DISSERTATION

Evaluation of Downtimeless System Evolution in Automation and Control Systems

How to decide whether a system under operation can be changed without disturbances?

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Em.O.Univ.-Prof. Dipl.-Ing. Dr. techn. Gerfried Zeichen /
Univ.-Prof. Dipl.-Ing. Dr. techn. Bernard Favre-Bulle
Institut für Automatisierungs- und Regelungstechnik (E376)

eingereicht an der Technischen Universität Wien,
Fakultät für Elektrotechnik und Informationstechnik

von
Dipl.-Ing. Christoph Sünder
Matr.Nr.: 9925300

Korneuburgerstraße 2
2003 Wiesen
ÖSTERREICH

Wiesen, im August 2008

"It is not said,
that it will get better,
if things change the way they are;
but if it should get better,
then things have to change the way they are."

Georg Christoph Lichtenberger (1742-1799)
German physicist and author

„Es ist nicht gesagt,
dass es besser wird,
wenn es anders wird;
wenn es aber besser werden soll,
muss es anders werden."

Georg Christoph Lichtenberger (1742-1799)
Deutscher Physiker und Schriftsteller

Abstract

This thesis concentrates on downtimeless evolution of the functionality of automation and control systems for production processes. These processes have to be adapted during the whole life cycle of the plant because of changing requirements, for instance new products or product variants. In addition, the free competition forces the companies to do not stop their production during the application of such changes. During operation the whole production plant has to be continuously adapted. These changes are introduced as downtimeless system evolution of automation and control systems in this thesis.

Current state of the art automation and control systems already provide the possibility to apply changes during operation to some extent. But these changes of the control logic introduce disturbances to the production processes, which may lead to reduced product quality or even damage of the plant machinery. In order to avoid these failures, the engineer has to be in the position to explicitly coordinate the evolution of the automation and control system to the special needs of the application.

Within this thesis we introduce a new engineering cycle for downtimeless system evolution. Thus the user is capable to freely program the kind of changes which will be applied during the evolution execution. Therefore, existing programming languages will be used, which additionally include the possibility for dynamic reconfiguration. Due to these applications, the so called evolution control applications, it is possible to change the current control logic during its operation. By additional interaction of the user also hardware components may be changed during operation.

The most important aspect for the execution of system evolution is the question regarding correctness. The new system state of the plant may be checked by conventional methods. But the check for the transition to the new system state needs a new methodology, as there is no concept available in literature up to now. This work provides as its main contribution an evaluation method for downtimeless system evolution.

The new methodology combines the engineering method and the basic properties of a correct downtimeless system evolution, which leads to a set of necessary measures for the evaluation. As main concept the current system state—a comprehensive description of all involved elements called KAPPA vector—is put into the center of investigations. The dynamically changing character of the system state during execution of the evolution control application is the basis for the overall evaluation method. As evaluation means two different concepts are applied: rule-based calculations and verification by model checking. Both concepts are based on the information provided in the KAPPA vector. Therefore the evolution of an automation and control system can be checked sufficiently and the new methodology of downtimeless system evolution can be applied to real systems.

Kurzfassung

Diese Arbeit beschäftigt sich mit der unterbrechnungsfreien Änderung der Funktion von Automatisierungssystemen für Produktionsprozesse. Diese müssen über den ganzen Lebenszyklus einer Anlage an die ständig veränderlichen Anforderungen wie etwa neue Produkte oder Produktvarianten angepasst werden. Da Änderungen aber durch den hohen Druck des freien Wettbewerbs zu keinem Stillstand der Anlage führen dürfen, muss das gesamte System kontinuierlich während des Betriebs adaptiert werden. Diese Änderung des Automatisierungssystems wird hier mit unterbrechungsfreier Systemevolution bezeichnet.

Aktuelle Automatisierungssysteme bieten zwar großteils die Möglichkeit, Umschaltungen der Steuerungslogik während des Betriebs durchzuführen, dabei werden aber Störungen in das System eingebracht, die z.B. zu verminderter Produktqualität oder auch Beschädigung der Anlage führen können. Um dies verhindern zu können, muss der Anwender in der Lage sein, die Evolution des Automatisierungssystems explizit auf die jeweilige Situation abzustimmen.

Diese Arbeit führt einen neuen Engineering-Zyklus für die Evolution von Steuerungssystemen ein, der es dem Anwender erlaubt, die Art und Weise der Durchführung einer Evolution frei zu programmieren. Dazu werden die vorhandenen Programmiersprachen genutzt und um die Möglichkeit zur dynamischen Rekonfiguration erweitert. Durch die Ausführung dieses sogenannten Evolutionssteuerungsprogramms kann das aktuelle Steuerungsprogramm (und unter Mitwirkung des Anwenders auch die dazugehörige Hardware) während des Betriebes verändert werden.

Die wichtigste Fragestellung in diesem Zusammenhang ist die Korrektheit des Evolutionssteuerungsprogramms. Ein neuer Zustand des Steuerungsprogramms kann mit herkömmlichen Mitteln auf Richtigkeit geprüft werden, aber für die Übergangsphase während der Änderung des Systems im laufenden Betrieb gibt es derzeit noch keine adäquate Überprüfungsmethode. Deshalb legt diese Arbeit ihren Schwerpunkt auf die Evaluierung von unterbrechungsfreien Systemevolutionen von Automatisierungssystemen.

In dem neuartigen Ansatz wird die Engineeringmethode mit den Eigenschaften für eine fehlerfreie Evolution in Zusammenhang gebracht und daraus die notwendigen Maßnahmen zur Evaluierung abgeleitet. Als wesentliches Grundkonzept wird der aktuelle, umfassend dargestellte Systemzustand hervorgehoben, der KAPPA Vektor genannt wird. Diese sich entsprechend der Evolution ändernde Größe ist die Grundlage für den gesamten Evaluierungsprozess. Auf Basis dieser Systembeschreibung werden zwei Arten von Überprüfungen durchgeführt: regelbasierte Berechnungen und Verifikation durch Model Checking. Beiden Fällen liegen die umfassenden Informationen aus dem KAPPA Vektor zugrunde. Dadurch kann die Veränderung des Automatisierungssystems vollständig überprüft und somit erst die neue Methodik der unterbrechungsfreien Evolution in der Praxis angewendet werden.

Acknowledgement

First of all I would like to thank Professor Dr. Bernard Favre-Bulle, who conducted my thesis as supervisor during a long period of time. Unfortunately he was not able to finish this work, and I am very grateful that Professor Dr. Gerfried Zeichen took this job during the final phase of my thesis.

I want to thank also Dr. Valeriy Vyatkin from Auckland University, who supported my work in many fruitful discussions and with enhancements of his verification tools. Finally, I am glad that he was willing to co-supervise this work.

This dissertation came to standing during my employment as a research assistant at the Automation and Control Institute at Vienna University of Technology. I want to thank all colleagues for providing a highly innovative and productive environment. My special thanks are dedicated to Dr. Alois Zoitl and Oliver Hummer for their support and fuitful discussions during my work. Wilfried Lepuschitz provided me with proof-reading the manuscript.

This thesis was highly supported by the research project εCEDAC. As second research partner, company Profactor GmbH, and especially Dr. Thomas Strasser, Martijn Rooker, Gerhard Ebenhofer, Mario Schüpany, and Roland Mungenast provided me with their ideas and discussions for my thesis. Even as we were located in different affiliations, we have been one big team. Dr. Franz Auinger took a central role during the preparation of this project, many thanks also to him.

The industrial members of the εCEDAC consortium have provided important incentives for my work. I want to express my gratitude to Josef Fritsche, Siegmar Thomas, Gerold Kerbleder (Bachmann electronic GmbH), Heinrich Steininger, Mario Semo, Thomas Baier (logi.cals Austria), Robert Schranz, Georg Keintzel (Siemens VAI), and last but not least Dr. Dietmar Loy (Loytec electronics).

Many warm thanks go to my parents, Brigitta and Karl Sünder, for their support during the long period of my education.

Finally and above all, I would like to thank my wonderful wife Monika for her devoted love and her incessant patience when my time schedule needed prolongation several times. Her meticulous attention to details has significantly improved my work for instance by proof-reading the manuscript. This thesis is dedicated to you my dear darling.

Content

Chapter 1

The general tasks for downtimeless system evolution

An *automation and control system* (ACS) for plant automation consists of different components such as actuators, sensors, communication systems, and appropriate control devices, as for instance depicted in Favre-Bulle (2004, Chapter 2). The most important type of control devices, which will be considered within this thesis, are *programmable logic controllers* (PLCs). Since its invention 40 years ago [45] many technologies have been added to the classical design of a PLC, their foundations have been defined by the *International Electrotechnical Commission* (IEC) in the IEC 61131 standard. Especially the software architecture and the programming languages of part 3 (IEC 61131-3, 2003) build the fundamentals of each PLC. The application fields for PLCs are manifold—the process under control concerns any kind of continuous, discrete, or mixed plant—and the ACS customers have very different background knowledge according to their role in the ACS industry: plant operator, field engineer, electrician, computer scientist.

The needs of the ACSs customers have been the topic of various studies and surveys. In order to give an impression of upcoming and new technologies in this field some key studies will be mentioned. A study has been conducted by the Iacocca Institute (1991), which identified nowadays well known requirements such as "production to order" or "lot/batch size equal or greater than one" as future demands for manufacturing systems. It has been stated that ACSs will be information intensive, reprogrammable, reconfigurable, and continuously changeable. These requirements can be summarized by the key word agile manufacturing, which means highly flexible adaptations to the current needs. The study also triggered standardization work within the IEC, and since 2005 the IEC 61499 standard is available as international standard, providing a basis for the requirements stated in this early study. A more recent study has been done by Favre-Bulle (2005) concerning the future directions in manufacturing science in Europe. Within this study a large number of European experts from industry and academia have been interviewed. The different views have been interrelated with existing international studies, and six future key technologies have been extracted. Three of them are directly related to ACSs, namely simultaneous production, 100% reconfigurable adaptive production systems, and semantic systems (introduction of reasoning and self-learning into control devices). Without going into detail about the results of this study it can be seen that these directions are very close to the vision of the early study of the Iacocca Institute. Even more impressive is the strategic research agenda of the European High-Level Group Manufuture (European Commission, 2006). This industry driven platform provides future research directions within the European Union frame programs in the field of ACSs. The report identifies advanced manufacturing engineering as one of the future challenges, with the topics reconfigurable technical systems and integrated processes/systems as important pillars.

The current situation of the overall time-to-market cycle within a process or production plant is depicted in the schematic in Figure 1. The bar diagram in the middle of this figures shows an average production process, starting from enterprise control until sales and after sales activities. The main time consumption is related the production process, which is denoted as standard production time TP_{St}. In case of changing production requirements (e.g., new variants of products) the capabilities of the production plant and especially the production automation in order to support these changes significantly influence the resulting time-to-market for the reconfiguration. The worst case situation is given in the top bar diagram. After using rapid prototyping and digital engineering for dynamic changes in product development functions of an enterprise the time for reconfiguration of manufacturing execution *TPC* is the bottle neck for quicker time-to-market.

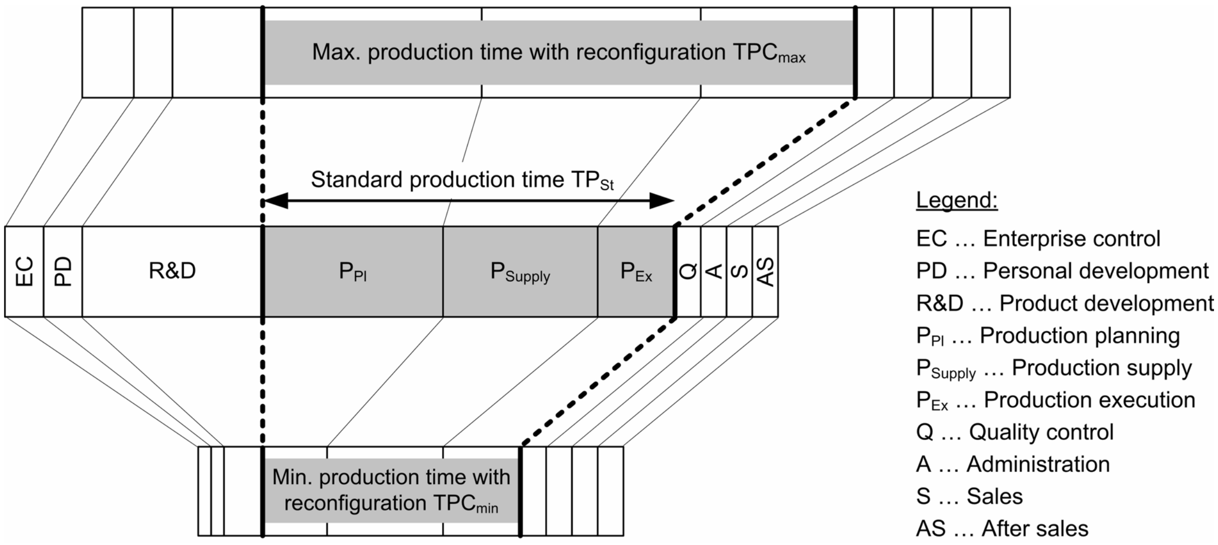

Max. production time with reconfiguration TPCmax
Standard production time TPSt
EC
PD
R&D
PPl
PSupply
PEx
Q
A
S
AS
Min. production time with reconfiguration TPCmin
Legend:
EC … Enterprise control
PD … Personal development
R&D … Product development
PPl … Production planning
PSupply … Production supply
PEx … Production execution
Q … Quality control
A … Administration
S … Sales
AS … After sales

Figure 1: Minimal and maximal time for reconfiguration of manufacturing execution[1]

With *downtimeless system evolution* (DSE), as it will be introduced in this thesis, we purpose to reduce the time for production to a minimum (see TPC_{min} in the third bar diagram of Figure 1). DSE aims at reconfiguration and changes to a system without the need to stop its operation. It enables the engineering of smooth transitions to new system states, whereas these changes are not limited to functionalities for the pure exchange of software parts. In literature, the term *dynamic reconfiguration* is used for changes of software during run-time. DSE does have a more comprehensive view, as a plant does consist of hardware and software. The plant evolves in its overall system characteristics over time and the ACS customer needs an appropriate means to model this behavior. As most important aspect of this new methodology, the evaluation of DSE will be developed: as means for the ACS customer to decide whether it will be downtimeless or cause a system break down (or something in between).

In order to make the challenges of DSE more descriptive, application scenarios from current industrial practice can be considered as for instance presented in Baier *et al.* (2007). One common prerequisite is that the plant has to be fully functional all the time, although the plant requires permanent maintenance like fault analysis and repair, incorporation of new functionality, or refactoring in reaction to environmental or changed requirements. For instance production plants for steel (e.g. rolling mills, hot dip galvanization lines) need to be operated continuously due to requirements of the process. A line stop would cause a lot of scrap and costs. A typical scenario for program changes is the optimization of the production process. Or a wind mill as example for an energy production plant, which cannot be shut down easily as this needs to be managed within the overall wind farm and wears out the mechanics of the wind turbine, too. Updates of erroneous software parts or added/changed functionality of the control program needs to be applied without downtime. Another application field of DSE is

[1] The different tasks and their correlation within the time-to-market cycle are based on Zeichen and Fürst (2000).

building automation. A modern building consists of a network of hundreds or even thousands of controllers. And the building changes its functions for instance when new tenants are moving in and out, when special events take place, or due to energy saving programs. The special challenge of such a building is the event triggered character of the usage of the building. A reconfiguration within the control program cannot be scheduled reliably to an uncritical time, since a switch-on of a light or a fire alarm may occur at any time.

The critical question of this thesis "How to decide whether a system under operation can be changed without disturbances?" aims at the use of basic means for evaluation of complex systems[32] applied especially for the methodology of DSE. According to the introductory chapter of Clarke *et al.* (1999), there are four principal methods available for complex systems: *simulation*, *testing*, *deductive verification* and *model checking*.

- Simulation means making experiments on a model of the system.
- Testing means making experiments on the real system.

Simulation and testing are well known and often applied for ACSs. In both cases it is rarely possible to check all possibilities of interactions and pitfalls.

- Deductive verification means the use of axioms and proof rules to check the correctness of the system.
- Model checking means the automatic and exhaustive search of the state space of the model of a system in order to determine the behavior of the system.

Deductive verification and model checking are known as formal methods, as their foundations are based on mathematical principles. Deductive verification is done by verification experts (usually mathematicians or logicians) with considerable experience. Although there is some support by software tools, this kind of verification is very extensive, takes long time and is only used for highly sensitive systems such as security or safety-relevant systems. Model checking is applicable also for non-experts due to its automatic character. Model checking consists of three tasks. First of all a model of the system has to be generated. Many different kinds of formalisms are available for this purpose. In a second step, the specification needs to be defined which describes the properties of the system under observation. Typically, a specification is expressed in any logical formalism. The use of temporal logics enables the definition of the system's behavior evolving over time. The third and last step within the process of model checking is the verification itself. Ideally this step is done completely automatic in ideal. The result is a Boolean value that states whether the model of the system fulfills the specification or not. If not, a counterexample is given which describes a path within the state space of the system where the specification is violated.

Based on these introductory comments on ACS, the DSE requirement and the current state of evaluation methods—especially model checking—the upcoming question is: How do these different topics fit together? The subsequent section describes the motivation for this work.

1.1 Motivation

The engineering process of an ACS is error-prone, as depicted by the statistics presented in Kropik (2005). By the use of extensive testing and simulation with existing engineering tools, the error rate can be kept rather low. Although, outstanding applications as depicted for instance in Bani Younis and Frey (2003) already require the use of more powerful methods such as verification by model checking in order to avoid failures of the system. The special parameters of DSE add further demands to the evaluation in contrast to "standard" applications:

- The DSE process happens only at once. There are no iterations that lead to a successive transition from the old system state to the new one. DSE means a hard switch to a new application.

- Therefore, DSE means an interruption of the control program's execution. Even when the new system state has been checked for correctness successfully, the transition to the new system state may cause a failure in the plant.
- DSE has to be performed without any—or at least as few as possible—disturbances to the total process, because any disturbance may lead to bad product quality or damage to the plant and its machinery. Even if it may be unlikely, the worst case is a break down of the plant.
- The single action "downtimeless system evolution" and the requirement of negligible disturbances faces the engineer with the challenge to think of all possible environmental conditions for the application of a DSE at any time in the production cycle of the plant.

The engineer has to decide, whether he wants to apply the changes by triggering the DSE at a certain time or not. This means that the engineer is responsible for a very complex operation to the operating plant and needs some means to support this decision. The choice of one of the above discussed evaluation methods may provide such kind of support. If we do have a closer look to the prerequisites of the different methods, a clear situation appears:

- Testing cannot be applied because the plant generally is not available for any experiences as it is in operation.
- Simulation is possible and may give an impression on the effects of the DSE. But it is not able to incorporate all possible scenarios and different situations that may happen in the plant.
- Deductive verification lacks the usability for ACS customers for different reasons, for example knowledge of the engineers as well as time consumption of the method.
- Model checking offers an automatic methodology that leads to a true/false decision. If an appropriate model of the control program and the plant is available, the engineer can decide about the system evolution by defining the specification. The model checking tool verifies the correctness of the specification for the whole state space of the model, and gives a counterexample in the case of a violation.

Verification by model checking promises to be the basic methodology that successfully provides support for the engineer when considering the application of DSE to the operating plant. But model checking does only provide answers with regard to the given specification. The engineer is responsible to define an appropriate specification in order to decide whether the DSE will be successful or not. This work has to provide a guideline on what are the important aspects that need to be incorporated in the specification, in order to make a decision about the correctness of a DSE. Because if necessary aspects are missing within the specification, the model checking tool will reply that the model may satisfy these requirements. But the system may produce failures during the DSE.

The problem itself may be examined in a more abstract way as depicted in Figure 2. Herein, an *ambigram* of Scott Kim is presented, which includes both the words true (written in lower case letters) and false (written in capital letters) in the same word. Depending on the perspective of the viewer, one of the two words appears (in Figure 2 this is clarified by different shadings of the words). In the case of system evolution, the engineer is in a very similar situation. He has to ask the right questions in order to receive a satisfactory answer. Even if all possible behaviors of the plant and the control program are taken into account for the execution of DSE, the result of the evaluation process will be the wrong answer (depending on the view point, which is given by the specification for successful DES defined by the engineer).

Figure 2: "TRUE/FALSE", Scott Kim, 1981[2]

1.2 Purpose of this thesis

This thesis aims at the providing of a methodology for the engineering process of DSE. Therefore, three topics have to be addressed:

- A new methodology for modeling of DSE, the transition from a current system state to a new system state during operation of the plant, without any disturbances to the plant.
- A new methodology for checking the correctness of DSE, if the modeled changes to the system under operation do not violate the plant's operation. This part represents the main novelity, which has to investigate the different aspects that need to be taken into consideration in terms of specifications for DSE. The ACS customer has to be guided for his decision on applying DSE to the system under operation. Clear and concise properties need to be developed in accordance with the means necessary for their evaluation. Model checking has been the first choice, but it does not have to be the best fitting means for all kind of properties.
- Provision of models of the system incorporating DSE for appropriate evaluation means. Next to a detailed specification also the detailed model as basis for verification by model checking has to be developed. Only if all relevant aspects are part of the model, the check for specifications can be executed satisfactory.

The presented methodology requires a system environment which provides basic means of dynamic reconfiguration. Further it has to be applicable for the specific standards used in ACSs. These are IEC 61131 and IEC 61499, whereas only IEC 61499 does provide a defined interface for dynamic reconfiguration. Consequently this work will use IEC 61499 as basis, but the use of IEC 61131 based systems needs to be supported, too (see Chapter 10 for this purpose).

Next to the list of aims for this work some excluded targets need to be mentioned clearly also. First of all, we will not investigate on a new formal modeling language or a new algorithm for model checking. Several modeling languages and model checking algorithms already exist, which have proved their benefits for certain application fields. We will utilize existing approaches and focus on an methodology for the application of these means for DSE. Additionally, this work does not aim at the development of a fully functional automatic verification tool. An appropriate tool framework needs to be based on the specific characteristics of a concrete system environment. The variety of ACSs is very broad, therefore we will concentrate on general considerations which may be applied to a concrete system environment and then integrated into the engineering tool.

[2] Copyright ©2007 Scott Kim [37]

1.3 Guideline through the thesis

We will start the discussion about the new methodology for evaluation of DSE with an analysis of requirements in Chapter 2. Herein the aim of the thesis is presented in more detail by definition of concrete tasks which have to be faced. The discussion about related work in this field as well as the state of the art concerning this work in Chapter 3 will provide the necessary background knowledge for this thesis. Furthermore the novelty of the approach will be described.

As starting point for our investigations we will introduce a new modeling method for DSE in Chapter 4. This is the necessary starting point, which provides a structured methodology based on the use of dynamic reconfiguration within a system environment for DSE.

Chapters 5 to 7 present the new methodology for evaluation of DSE. First of all, we will describe the general framework for evaluation in ACSs with DSE. According to the engineering methodology, the evaluation concerns will be split up into two parts: calculations based on the current system state and verification by model checking. The reason for this is on the one hand that many questions concerning the success of a DSE can be checked by rules with detailed knowledge of the system and its characteristics. On the other hand, the scope for the verification by model checking will be cut down and complexity is decreased for the ACS customer. Next to the definition of properties for the specification of DSE, also appropriate models for the evaluation process will be presented.

The results of the evaluation methodology are discussed in Chapters 8 to 10. In order to give a more practical impression of the evaluation process, different examples are given for demonstration. Another interesting aspect is the effect of DSE and its evaluation to industrial practice. Herein two concrete implications are discussed: the role of companies in the value-added chain of ACSs on the one hand and a new paradigm for engineering of ACSs on the other hand. These investigations will be concluded by the consideration of an industrial engineering tool based on IEC 61131 and the application of the proposed modeling and evaluation methodology to this system.

An outlook on further enhancements and developments of this thesis is given in Chapter 11. The work is summarized in Chapter 12. Some interesting topics are prepared in the appendix, as they would overload the main content of this work.

Chapter 2

Analysis of Requirements

The overall reasons of providing a methodology for the evaluation of DSE have been described above. When we take a closer look at such an evaluation method, we will find several requirements that need to be satisfied. An important prerequisite for this analysis is the situation in ACSs. The customers in ACSs are commonly only skilled in those fields which are related to the process under control, but they have to use programming languages as interfaces to the ACS components. The means of ACSs component providers need to be adapted to this special situation, and usability for the ACS customer has to be kept in mind in general. On the other hand, an ACS component is a highly sophisticated, programmable device that interacts steadily with its environment.

We start our consideration with very general requirements which are related to any kind of control device, even if pure control logic needs to be verified. Secondly we will give a list of additional requirements for the evaluation of DSE. As a third view point we will investigate the needs of the ACS customer. Herein, some issues concerning the usability of such a new methodology are taken into consideration.

2.1 Execution requirements for control devices

The practical work with ACSs is dominated by testing and simulation as means for evaluation of the functionality of a control device. When we think of verification by model checking for the pure control functionality of a control device, the following requirements have to be handled.

(1) Temporal behavior: In an ACS each control device typically is a *real-time computer system*. According to Kopetz (1997) such systems are characterized by functional requirements (these belong to the task that is solved by the control device), temporal requirements (correctness of the calculated results and actions), and dependability requirements (herein reliability, safety, maintainability, availability, and security are summarized). The main goal of an evaluation method is to prove the correctness of the computations of a given system. This belongs to the first item, functional requirements. Additionally, it is very important to take into consideration the temporal behavior of the control device, too. Only if both items are mentioned together, appropriate evaluation results will be achieved. When we apply this temporal behavior to verification by model checking, different specialized requirements emerge:

- ***Modeling:*** The modeling language needs to provide appropriate means for characterizing temporal behavior of the system.
- ***Specification:*** The use of temporal logic for the definition of specifications already includes temporal behavior. Nevertheless, there are different extensions available for improved handling of real-time behavior in specifications as discussed for instance in Clarke *et al*. (1999, Chapters 16 and 17).

- ***Model checking algorithm:*** Obviously, the algorithms and techniques for model checking must be able to handle these specializations for temporal behavior.

Dependability requirements of the control system as declared above will not be mentioned in this work. Of course, evaluation methods will also be useful for evaluation of safety or security of a control device. But this is not relevant for the process of DSE at a first glance. Further investigations (see outlook in Chapter 11) may be started as a next step based on the initial results of this work.

(2) Execution semantics: The way of executing control logic, which is implemented in a given control device, needs to be modeled exactly. The behavior of the control device obviously depends on the concrete implementation of the control logic and the runtime environment. This also applies when the runtime environment is compliant to a specific standard. For instance, early approaches for the verification of the IEC 61499 standard—Vyatkin and Hanisch (1999) presented a modeling approach for IEC 61499 function block applications the first time —purely concentrated on the definitions of the standard. Implementation details have not been considered and the approach was applicable for any kind of runtime environment. But in recent publications diverse examples are given that the concrete implementation of the standard, the so-called execution semantics, definitely influences the behavior of the control device in certain cases. Sünder *et al.* (2006a) discuss different questions about the execution semantics of the IEC 61499 standard in general, a precise answer is only possible with respect to a given implementation. Sünder *et al.* (2006a) especially focus on the *Function Block Run-Time* (FBRT), which is included in the *Function Block Development Kit* (FBDK), the first IEC 61499 engineering tool. Another impressive example is given by Čengić *et al.* (2006) by a comparison of the *Function Block Execution Runtime* (FUBER) and the ISaGRAF engineering tool [26]. The described situation of so-called contiguous events shows that events may be lost depending on the used runtime environment.

(3) Underlying system configuration: The consideration of implementation issues of a runtime environment may be obvious, because there may be different demands according to the wide application field within ACSs. But the situation is even worse, if we assume one runtime environment in different system configurations. The underlying system configuration has to be taken into consideration for the evaluation process, too. The most important aspects are the operating system and the computational power of the hardware platform. Zoitl (2007) describes a runtime environment for IEC 61499 standard, which enables real-time execution of control logic. The implementation includes an abstraction layer for the underlying operating system in order to provide similar behavior on various platforms. As proof of concept, the runtime environment was adapted to three different operating systems. By the use of diverse scenarios and appropriate measurements Zoitl concludes that characteristics of the operating system influence the execution behavior of the runtime environment, and therefore also the behavior of the control logic.

2.2 Requirements for downtimeless system evolution

The evaluation process for changes during operation of ACSs adds further specific requirements the above mentioned general claims to control logic evaluation. These aspects concern to the overall system evolution process—the engineering—and especially the evaluation process.

(4) Modeling dynamic reconfiguration: DSE tends to changes of the ACS in a bigger context, utilizing dynamic reconfiguration as one basic methodology. It is necessary to define these basic actions of dynamic reconfiguration in detail for the implementation within a runtime environment as well as for a formal description. Zoitl *et al.* (2006) provide a categorization of such so-called *basic reconfiguration services*, which are necessary to model any kind of changes in the control logic of a control device. As a basis the management commands of the IEC 61499 standard, defined in (IEC 61499-1, 2005), are used and enhanced compliant to the standard. The above mentioned runtime environment discussed in Zoitl (2007) implements a full set of basic reconfiguration services and provides a detailed description of their behavior.

(5) Free programmable downtimeless system evolution: Many different reasons exist for the development of DSE in ACSs, and the changes may influence even large parts of the control logic. Baier *et al.* (2007) describe also situations where the DSE is spread over several control devices within the distributed ACS. As concrete example the reconfiguration of a communication channel is given. The scope of a DSE—and correspondingly also the scope for the evaluation of the DSE—needs not to be restricted to any kind of area in the control logic or in its size. Furthermore it is an important aspect to be able to freely program the process of DSE because of the widespread application fields in ACSs. This is in contrast to other approaches. For instance Steffen (2005) claims minimal invasiveness as one main requirement for the reconfiguration of control systems at run-time. As this approach is focused on closed-loop control systems such a restriction is possible to reduce the complexity problem.

(6) Extensive engineering support: The acceptance of new technologies by the ACS customer is a very difficult process as already denoted above. Hall *et al.* (2007) consider the very slow adaptation of the IEC 61499 standard by ACS vendors and describe challenges that must be met in order to encourage more active use and support. One of the main points is the availability of appropriate engineering tools. On the one hand, ACS customers need good tools to use the standard in industrial practice. On the other hand, it is hard to provide a tool without appropriate market response. For the methodology of DSE and especially its evaluation, one very important requirement is the extensive engineering support for this new technique. Of course, this work does not aim at the development of such an engineering tool. But the methodology for evaluation of DSE needs to be based on mechanisms which enable simple integration to an engineering tool. Especially, there should not be any principal hindering reason for this integration.

2.3 Usability requirements

The last mentioned requirement already suggests that the usability for the ACS customer needs to be kept in mind for the whole approach. The situation is very complicated due to the different kinds of ACS customers—see the discussion in (Hanisch, 2004)—, which are skilled very differently. And especially the necessary skills for evaluation and mathematics are very often not present.

(7) Provision of formal models: The ACS customer is not able to provide the formal models[3] for the overall system. There are various hindering reasons, for instance lack of knowledge, time effort, and unknown details about the underlying system. Therefore it is necessary to provide the formal models to the ACS customer in such a way that they can be easily used within the engineering process. Vyatkin and Hanisch (2001a) give an example for such an integrated engineering support in the *Verification Environment for Distributed Applications* (VEDA) tool. Herein, the IEC 61499 application is automatically transformed into formal models. Furthermore the model of the plant is an integral part of the tool. This enables VEDA to easily generate the model of the overall system. Validation of single traces within the state space of the system is animated in a graphical visualization of the plant. According to the above given Requirements (1) "Temporal behavior", (2) "Execution semantics", and (3) "Underlying system configuration", this process of automatic generation of formal models is

[3] We will use the term formal model for a mathematical description, which is necessary for instance as input for a model checking algorithm.

much more complicated than realized in VEDA, but there needs to be a methodology for the generation of the formal models with only very little interaction by the ACS customer. Provision of formal models by the different ACS vendors included in the overall system is necessary.

(8) User-friendly definition of specifications: Verification by model checking is a technology used in research, but the transition of this technology to practice has been slow even in the field of computer science. Dwyer *et al.* (1998) state that one of the main reasons is that practitioners are not familiar with the specification processes, notations and strategies. As a consequence, they propose *Property Specification Patterns* as a generalized description of the specification for a certain property. Dwyer *et al.* (1999) present a survey on specifications used in literature, which reports that 92 percent of the specifications can be categorized in their system of property patterns. As the background knowledge of people working with ACSs concerning verification by model checking is even less, the use of a user-friendly definition of specifications is necessary. A pattern may be described using natural language which is automatically adapted to a temporal language. As a consequence also ACS customers may be able to define also complex properties of the system evolution process without being faced with any kind of temporal language. For successful application of a property specification pattern system for the evaluation of system evolution it is necessary to adapt the general patterns to the special needs of DSE.

2.4 Summary

This chapter provides a detailed analysis of the requirements which need to be satisfied by a methodology for evaluation of DSE. This is part of an overall engineering methodology for DSE. These requirements build also the basis for modeling of DSE. Table 1 gives a short overview on the different requirements.

Control device	(1)	Temporal behavior	In addition to functional behavior it is very important to recognize also temporal behavior for evaluation.
	(2)	Execution semantics	Implementation details about the execution semantics of the used runtime environment must be mentioned.
	(3)	Underlying system configuration	The overall system configuration has to be considered for a full featured model of the control device.
Downtimeless system evolution	(4)	Model dynamic reconfiguration	The basic actions of dynamic reconfiguration have to be described and modeled in detail.
	(5)	Free programmable DSE	DSE must not be restricted in order to enable its use in any application field of ACSs.
	(6)	Extensive engineering support	Evaluation of DSE needs to be accompanied by extensive engineering support.
Usability	(7)	Provision of formal models	Formal models of the overall system need to be generated automatically or provided by ACS vendors.
	(8)	User-friendly definition of specifications	Specifications have to be defined by ACS customers in natural language (without knowledge in temporal logics).

Table 1: Requirements for the evaluation of DSE for this thesis

Chapter 3

State of the Art

Already the introduction of this thesis in Chapter 1 gives a very brief description of the related fields of technology. This chapter will provide more detailed information about the three main topics:

- A general view on automation and control systems is depicted with particular interest on the lower levels of control in production industries, their programming languages, and description languages.
- Dynamic reconfiguration, as outstanding feature of these systems, will be considered within various architectures. Based on a more general view on methodologies known from computer science and embedded systems design, the usage within ACSs is emphasized in detail. This includes also the way of how to manage the transition from one system state to another.
- The aspect of evaluating the functionality of a given system is discussed with respect to the model checking methodology. There has been very much progress in this field in the past decades. Although practical relevance is rather low for industrial applications—this especially applies for ACSs—several approaches exist for the use of verification by model checking in the current state of the art.

There is no claim for completeness of the presented material, since the variety of possible methodologies and technologies is overwhelming. Researchers have been active in many fields of ACSs to face the challenges of industry, and as already depicted by different studies (e.g., European Commission, 2006) investigating new solutions for today and future challenging requirements will continue.

3.1 Automation and control systems

The initial aim of an automation and control system is to provide control over a physical process by the use of some control devices. As a general source of information concerning ACSs we will use Favre-Bulle (2004). Herein a principal differentiation between product automation and plant automation is given. The first one is related to the control of physical processes within a product, the latter investigates complex technical processes within a plant. Our focus is clearly on plant automation, although similar or equal methodologies may be used for product automation, too.

The general structure of an ACS for plant automation is given in Figure 3 as a very abstract schematic. There are different roles of vendors that can be identified. This kind of illustration stems from Vyaktin *et al.* (2005), who argue that this structure characterizes both manufacturing plants as well as process plants. The responsibilities of the different players can be described roughly as follows:

- ***Component vendors:*** The basic building blocks of an ACS are actuators and sensors. These are the direct interfaces to the process under control. The range of these elements is very wide according to the large application field of ACSs. For integration of components into the overall ACS, a defined interface is applied for instance by the use of a *field bus* as communication system.

- ***Machine vendors:*** Based on the various components machine vendors build functionally complete production machines. Herein also handling of products as well as logistics infrastructure is included. The integration of a machine may be based again on field buses, but also more complex communication is possible.
- ***System integrators:*** The overall production plant is assembled by a system integrator, who takes care of coordination of machines, production flow, and supervision.
- ***Industrial enterprises:*** The industrial enterprise integrates the production plant into the overall enterprise. Herein various aspects like product life cycle planning, customer requirements, or customization of products are taken into consideration.
- ***Tool/Controller vendors:*** All levels mentioned above use different tools, runtime environments, or complete control devices in order to fulfill the required functionality. On the lower levels of Figure 3 simple microcontrollers or PLCs will be used for instance as control devices for components and machines. On the upper levels *Industrial Personal Computer* (IPC) will provide the necessary computational power for *Production Planning and Scheduling* (PPS) or *Enterprise Resource Planning* (ERP).
- ***Service vendors:*** In addition, service vendors exist who are specialized on specific topics within the life cycle of the ACS, for instance diagnostics, maintenance, or optimization.

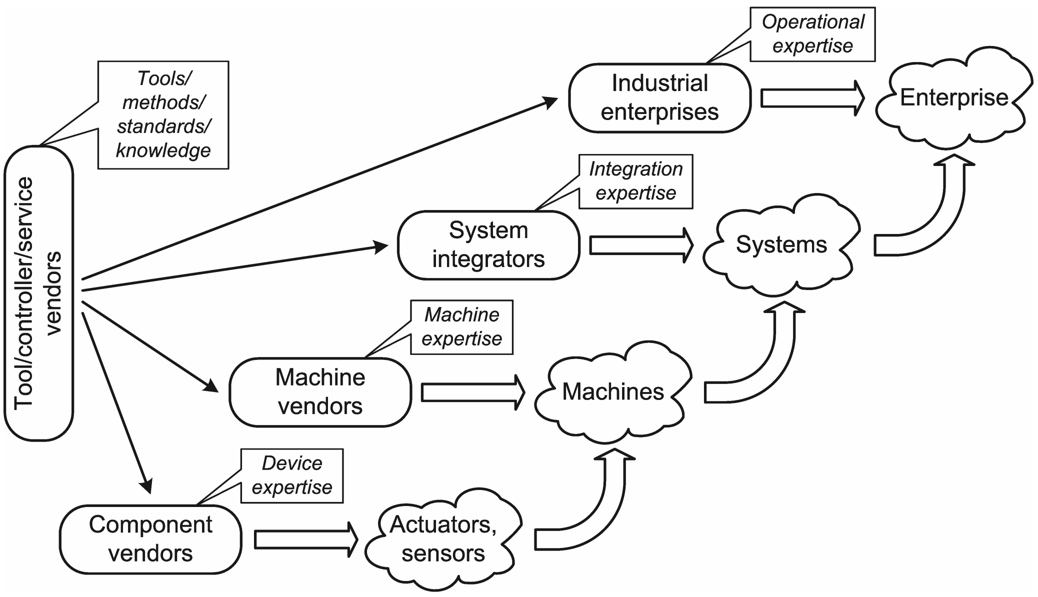


Figure 3: General structure and roles of vendors in ACSs, based on (Vyatkin *et al.*, 2005, Fig. 1)

Figure 3 also indicates the basic business model of the different vendors within an ACS. For instance, a component vendor provides his special expertise on the component. The machine vendor creates additional value as he appends his expertise on the machine and its behavior and so on. Vyaktin *et al.* (2005) claim that if each of the different players is able to add his knowledge to his product on the particular level of the ACS in an open manner, a new level of increased efficiency and simplified engineering can be achieved. The main requirements for such a new open knowledge economy in the ACS are encapsulation and protection of *Intellectual Property* (IP) in software components, an open architecture of interoperable control devices, and open software tool integration platforms.

Automation objects

One main concept presented in (Vyaktin *et al.* 2005) is characteristic for current research in ACSs, the so-called *automation object*. A proper description of an automation object includes apart from the control logic also information concerning layout, electric wiring, diagnostics, visualization, simulation, and so on. Therefore an automation object represents mechanical devices associated with software functionality and additional data. Also a computational unit is part of the mechatronic devices, and these basic building blocks are used to establish machines and systems in a hierarchical manner. The concept of a component, that includes mechanics and electronics, is well known as mechatronic device (e.g., a sensor or actor with a field bus interface). The idea of free programmable mechatronic devices with an extensive description of the different aspects of the device is the topic of new work presented in recent years. Vyatkin (2003) gives a description of such an automation object as a product that unifies three items:

- ***Mechatronic component:*** A physical functional device with sensors, actuators and electronic circuits.
- ***Embedded control device:*** A computing device with interfaces to the sensors and actors as well as to the network.
- ***Software component:*** A set of data and control logic implementing various automation functions. These elements provide the autonomy and cooperation of the automation object.

There are several reasons for the definition of such an automation object. Sünder *et al.* (2006b) describe the composition of ACSs based on these elements (the work uses the term automation component as it does not include all views of an automation object) for simplification in engineering. The aspects logic, diagnostics, and *Human Machine Interface* (HMI) are mentioned explicitly. The structure for all these elements is oriented to the functional structure of the ACS, which is commonly already consistent with the hardware structure. This unified hierarchical architecture increases efficiency for both engineering and maintenance. Each automation component is represented by a universal interface towards the system, which additionally increases exchangeability of components. This provides the basis for reconfiguration of the ACS on the level of automation objects, as it is also discussed by Vyatkin (2003). Ferrarini *et al.* (2003) describe a very similar approach. Herein the motivation is to deal with complex systems by a hierarchical decomposition of the system behavior and possible modularization. The overall system is considered as an object, with the expected behavior as its main method. This method makes use of the functionality (again methods) of sub-modules. Ferrarini *et al.* (2003) refer to these elements as intelligent mechatronic modules, similar to the definition of automation objects. For further work in this field the reader may consider for instance also Lee *et al.* (2004), which present a component-based distributed control systems for assembly automation, or Thramboulidis (2005), which additionally includes the analysis and design phases of the development process to establish so called model-integrated mechatronics.

Another interesting aspect—especially for this work—is included in the approach presented by Bonfe and Fantuzzi (2003), the evaluation of mechatronic object-oriented models. The basic elements for the design of ACSs are again similar to the above given definition of automation objects, although they are called mechatronic objects in this work. These objects are modeled by the use of adapted class diagrams and state charts from the *Unified Modeling Language* (UML). For each mechatronic object the control action as well as the hardware behavior (the uncontrolled plant) are modeled. The interaction of the mechatronic objects is described by means of collaboration diagrams (also specified by the UML). The authors present the transformation of these models into the input language of the verification tool

Symbolic Model Verifier (SMV) in order to provide verification by model checking. We will refer to this aspect of the work in more detail in section 3.6.4.

Reconfigurable manufacturing systems

The aspect of reconfigurability in ACS is also topic of a new type of manufacturing systems, the *Reconfigurable Manufacturing Systems* (RMSs). Koren *et al.* (1999) claim that RMSs are needed for cost-efficient response to the fast changing market. In contrast, dedicated manufacturing systems have a fixed machine structure and are able to produce high volumes at low prices, but they lack fast response to market changes. Flexible manufacturing systems are limited due to the use of universal machines, e.g. *Computer Numerical Control* (CNC) *machines*, and they lack high throughput. Setchi and Lagos (2004) also mention cellular manufacturing systems, that are also inflexible to market changes. According to the definition of RMSs in Koren *et al.* (1999, Section 2), RMSs "are designed at the outset for rapid change in structure, as well as hardware and software components". The use of automation objects, herein defined as the elements of a modular machine, achieves the ultimate goal of RMSs: "a systems approach in the design of the manufacturing process that allows simultaneous reconfiguration of (1) the entire system, (2) the machine hardware, and (3) the control software." (Koren *et al.*, 1999, Section 3).

This work will focus especially on the third goal of RMS on the lower levels of control logic. This is in contrast to other approaches related to the upper levels of a machine or system. Herein often agent-based approaches are introduced. To give an example, Lopez Orozco and Lastra (2007) describe their approach of a control model for RMSs, that utilizes so called mechatronic modules as basic building blocks (these are again similar to automation objects). They differentiate between two types of control for a system based on mechatronic modules, the logic control application and the coordination control. The logic control encapsulates the interaction with the hardware, its functionality is fixed. The coordination control is realized by the use of agent technology, which is responsible for communication and coordination of the mechatronic modules in order to fulfill the overall task of the RMS. Reconfigurability of the RMS is achieved by the coordination logic.

Software components in computer science

The above discussion about the use of components in ACSs needs to be examined also from the viewpoint of methodologies from computer science. An important source for an insight into component-based software development is Szyperski (2002). The situation about terms and definitions is even worse for software components as already described for the idea of automation objects. In order to structure the following discussion, we will refer to the definition given by Szyperski (2002, Section 4.1.5): "A *software component* is a unit of composition with contractually specified interface and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."[4] There are three main characteristic properties of a software component according to this definition (Szyperski, 2002, Section 4.1.1):

- ***Unity of independent deployment:*** This property aims at a clear separation between the software component's environment and other software components. Further, a software component will never be deployed partially.
- ***Unity of third-party composition:*** A third party means someone who has no access to the construction details of all the components involved. Nevertheless, a third party has

[4] This definition is not in conflict with the definition of an automation object or its software components in general. It strongly depends on the used programming languages and methods in order to decide this question. We will discuss this aspect in more detail in Section 3.2.

to be able to combine a software component with other software components. Therefore, a software component needs to be self-contained.[5]

- ***No (externally) observable state:*** It is required that a software component cannot be *distinguished from copies of its own.*

The component-oriented approach is very popular in information technologies and represents a main technology for current software architectures. This also applies for a special class of computer systems related to ACSs, the so-called e*mbedded systems*. The term embedded systems is again widely used and—expectedly—even contradicting definitions are mentioned in literature. We will follow the definition used within the European Union funded project *Accompanying Measure on Advanced Real-Time Systems* (ARTIST). Herein Bouyssounouse and Sifakis (2005, Section 1.2) refer to embedded systems as “electronic programmable subsystems that are generally an integral part of a larger heterogeneous system”. The interaction with the physical plant is the source of the real-time constraints, therefore embedded systems in general belong to the class of real-time computer systems (as already mentioned in Section 2.1). Consequently, any kind of computer controlled device within ACSs, for instance PLCs or automation objects, is an embedded system. But the application field of embedded systems is very broad and includes for example also automotive, aeronautics, consumer electronics, or telecommunications. This has an important impact to the techniques that are in use for the design and development of embedded systems. The programmer of an embedded system, for example a specific controller for a car or an airplane, is in common a specialist in computer science. At least the tools and proposed methods require distinctive skills in this area. For ACS this is only true for those people who are involved in the design of a control device such as a PLC. The end-user or customer of ACSs has skills concerning the process under control, but typically not in computer science as described by Hanisch (2004). The interface visible for the customer of an ACS control device is a programming language, whereas for embedded systems in general the interface is described by a set of parameters or some kind of HMI, if at all.

Another aspect that needs to be mentioned is the design of distributed embedded systems. Within all topics mentioned in the ARTIST roadmap (Bouyssounouse and Sifakis, 2005) it is obvious that a system consists of more than one single embedded system. Even more a large number of embedded systems have to cooperate in order to fulfill the system requirements. Kopetz (1997) presents the time-triggered architecture as main element for such a distributed real-time system. Herein, the communication between the different single embedded systems is determined by a schedule fixed at design time. In ACSs different means for handling this problem are available according to the special requirements. We will discuss these solutions in the following section.

3.2 Programming languages

The majority of ACS customers uses programming languages that are based on the standard IEC 61131-3 (2002). Herein four different languages for programming as well as one modeling language are defined, that can be found (at least in some dialects) in any PLC system. In addition, the IEC has defined the standard IEC 61499 in 2005, which is expected to become the successor of IEC 61131-3. Nevertheless, the practical usage of IEC 61499 is at the moment very low. Next to these two IEC standards, there may be used also any kind of programming or modeling approach from the wide field of embedded systems and computer

[5] The claim for protection of IP from Vyaktin *et al.* (2005) fits exactly with the claim for third-party composition. In detail the general structure of ACSs presented in Figure 3 requests such a property as there are several players involved for the establishment of production plants.

science. But from an overall perspective, these approaches are insignificant, as this is reflected also by Bouyssounouse and Sifakis (2005).

We will discuss these two IEC standards in contrast to the definitions of software components in order to bring them in the context of the component-oriented programming paradigm. This is especially useful, as for the purpose of dynamic reconfiguration the use of software components is very advantageous due to its characteristic properties. Additionally, the following definitions from Szyperski (2002, Section 20.3) may support the discussion:

- "A *component framework* is a dedicated and focused architecture, usually around a few key mechanisms, and a fixed set of policies for mechanisms at the component level."
- "A *component system architecture* consists of a set of platform decisions, a set of component frameworks, and an interoperation design for the component frameworks."

The first definition points out clearly that it is very important to mention also the component framework when considering software components. We will discuss this topic in detail for the programming languages of ACSs. The second definition aims at the overall configuration of a system, in our case a control device. As already mentioned in Requirement (3) in Section 2.1, the underlying system configuration needs to be taken into consideration for the evaluation of control logic. In detail, typical control devices are based on a *real-time operating system*, which represents a component framework itself. Additionally, some kind of *middleware* may be integrated (again a component framework), and on top of this the component framework for a programming language may be applied. Bouyssounouse and Sifakis (2005) provide an overview on current available real-time operating systems and middleware architectures, which is representative also for ACS control devices.

3.2.1 IEC 61131-3

The international standard IEC 61131 provides a set of eight parts concerning PLCs and their associated peripherals. Part one (IEC 61131-1, 2003) defines the principal characteristics of a PLC as digital, electronic system designed for industrial environments, which uses internal memory and user-oriented instructions in order to control and command machines and industrial processes. The most important part is IEC 61131-3 (2003), which defines programming languages and data types as well as a software architecture for PLCs. There are several sources for detailed information available. The most important one is the standard itself, next to several books as for instance Lewis (1998) or John and Tiegelkamp (1995). To the author's best knowledge there exists no book that is related to the actually valid, second edition of IEC 61131-3, although John and Tiegelkamp (2000) provide an outlook on upcoming changes to the first version of the standard.

The programming languages of IEC 61131-3 are widely used in industry. This is also a result of the work of the *PLCopen* organization, which is a user and vendor-driven platform to support the usage of IEC 61131-3 in industry. In recent years some specifications for add-ons in form of explicitly defined function blocks (e.g. motion control or safety) have been published that have strengthened unified usage of the IEC 61131-3 for certain application fields.

Software model

The main elements of the software architecture defined in IEC 61131-3 are given in Figure 4. The top element is called configuration and represents a programmable controller (this term is used in the standard for a PLC). A configuration consists of one or more resources, which represent a signal processing function. A resource consists of tasks and programs. A task is an execution control element that provides periodic or triggered execution of associated *program organization units* (POUs). This association is depicted by the execution control paths in

Figure 4. In general POUs may be programs, *function blocks* (FBs), or functions. Programs can only be instanced within resources, while FBs can be instanced within programs and FBs. The difference between functions and FBs is that functions shall not contain internal state information.

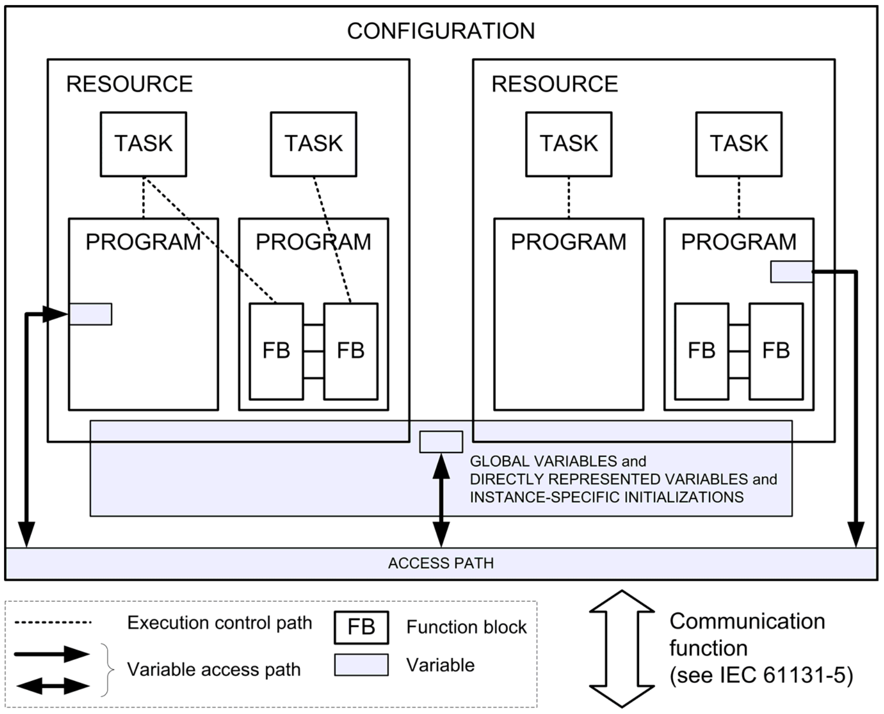


Figure 4: IEC 61131 Software model, (IEC 61131-3, 2003, Figure 3)

Data types

(IEC 61131-3, 2003) defines a set of elementary data types, such as integers, strings, or real numbers, a hierarchy of generic data types in order to enable overload mechanisms, and so-called derived data types. The last one describes user-defined data types that are derived from the existing data types, for instance an enumeration or a structure.

Variables

Variables are data objects associated with the inputs, outputs, or memory of the PLC. A variable can be declared to be one of the elementary or derived data types. There exist several variants of variables, which have important influence on the way of how to program a PLC. A detailed list is given in (IEC 61131-3, 2003, Table 16).

- Directly represented variables provide access to data elements with physical or logical locations in the PLC's input, output, or memory structure.
- VAR and VAR_TEMP identify internal variables (the second one means temporary storage).
- VAR_INPUT, VAR_OUTPUT, and VAR_IN_OUT define the interface of a POU, whereas the last one can be seen as a reference instead of a storage element.
- VAR_GLOBAL defines variables that can be used within the whole scope of a configuration or resource. By the use of the VAR_EXTERNAL construct a POU can yield access to a globally defined variable.
- VAR_ACCESS defines variables that can be used for remote access via the communication services specified in (IEC 61131-5, 2000). An access path associates each such variable with a global variable, directly represented variable, or input, output, or internal variable of a program or FB. The direction of the access path can be specified (i.e., read/write or read only).

- VAR_CONFIG is used to provide a means to assign a location or an initial value to symbolically represented variables. For this special case the use of an asterisk notation is possible, which enables the utilization of a “*” in a type definition. The full specification for an asterisk notation has to be expressed in the configuration which uses an instance of this type.

Structuring of programs and FBs

A POU is considered as a single action which is executed under control of the invoking entity. Additional structuring of programs and FBs is possible by means of the *Sequential Function Chart* (SFC) construct. Herein actions are associated with states, and the active state evolves according to interrelation of states via transitions and evaluation of the transition conditions. SFC represents a kind of state machine, with the possibility of several active states at the same time. The execution of actions is additionally parameterized by action qualifiers, so that for instance an action may only be executed once when the corresponding state becomes active. As a result programs and FBs can be structured in a sequential manner.

Programming languages

The programming languages of the IEC 61131-3 standard are defined in order to describe the control behavior of a POU. Nevertheless, any other programming language may be used instead. The standard defines two textual languages, *Instruction List* (IL) and *Structured Text* (ST), and two graphical languages, *Ladder Diagram* (LD) and *Function Block Diagram* (FBD). LDs come from the relay ladder logic diagrams, which have been replaced by the PLC. This programming concept provides a similar visual interface with additional features such as calls of FBs and functions. IL is very similar to the assembler language and provides a very simple instruction set. In contrast ST is similar to a higher level programming language such as Pascal or ADA, but with limited functionality (e.g., restricted specification of procedures or no use of pointers and memory access). The FBD programming language stems from Boolean logic diagrams, but additionally supports non-Boolean data types. A general difference to programming languages from computer science that needs to be kept in mind for all IEC 61131-3 programming languages is their execution behavior. The execution element is a task which can be triggered cyclical (which is typically the case) or by occurrence of an event, and POUs are executed in the context of these tasks.

Concepts for distribution

Although the IEC 61131-3 standard is defined for one PLC, also the cooperation of PLCs is a matter of fact in real world applications. There are two means that have to be mentioned when considering programming of two or more PLCs within one application:

- ***Network variables:*** Many providers of runtime environments according to IEC 61131-3 have introduced the concept of network variables, which is based on global variables. The only difference is that a network variable is valid for several configurations which may be located on different PLCs. The mechanism for synchronization of the local representations of a network variable is implemented vendor specific. Petig (2000) describes this methodology in more detail.
- ***Communication function blocks:*** Part five of the IEC 61131 standard (IEC 61131-5, 2000) defines communication FBs for access to variables that are identified via the VAR_ACCESS construct from outside the configuration (see also Figure 4). Additionally, status information can be requested from a PLC. Although this standardized concept for communication exists it is not used in practical realizations of PLCs.

Software components within IEC 61131-3

In order to find a relationship of the IEC 61131-3 constructs and the paradigm of component-based software development we will consider different scenarios for a mapping with the above mentioned definitions for software components and component frameworks. A first idea is provided by Bouyssounouse and Sifakis (2005, Section 12.2, Example 3), which states that "function blocks can be viewed as components and interfaces between blocks are released by connecting in-ports and out-ports". In this context especially the definition of interfaces and their descriptions are very important. The interface has to be defined in a clear manner and (as denoted in the definition for a software component) represents a contract in order to separate the software component from the component framework and other components. There should not exist any hidden interface that influences the behavior of a software component. Kopetz (1997, Sections 4.3.1 and 5.5.1) describes such hidden interfaces for a component-based distributed real-time system. He claims that they lead to incorrect results when reasoning about the correctness of a composition.

- ***Function as software component:*** A function is the most restricted element within the IEC 61131-3 software model. It does not include state information and is not allowed to use the VAR_EXTERNAL construct. Its interface and behavior is clearly described and a function can be seen as a software component.
- ***FB as software component:*** An FB is able to have state information, which is not contradicting the mapping as a software component. But there exists additionally the VAR_EXTERNAL construct, which enables the relation to any other element within a configuration or resource. The interface is only described by the use of a data element without any further description of the behavior of this interface. If the counterpart (and even the number of these counterparts is not limited) is related to another task with different execution settings, the behavior of an FB is in conflict with the principles of a software component.
- ***Program as software component:*** Roughly speaking there is no difference between a program and an FB aside from the reusability of FB types. Therefore, the same argument concerning the VAR_EXTERNAL construct applies also for programs. Another problem occurs from the execution control of FBs within programs. As depicted also in Figure 4 an FB may be related to another task than the overall program, which is specified within the resource. This results in another problematic situation concerning the definition of software components, as the behavior of two program instances will be different.
- ***Resource as software component:*** A resource includes information about the control logic depending on the used program instances as well as the execution behavior—the included tasks and their parameters. The result is a rich description of a resource as a software component. Nevertheless, there exists also the VAR_EXTERNAL construct which enables the use of an interface without behavior description in the resource. Therefore a resource cannot be considered as a software component.
- ***Configuration as software component:*** The configuration is the highest level in the software model of IEC 61131. A global variable within the configuration depicts an internal state of a software component and would fulfill the requirements for a software component. The interface of a configuration is given by the VAR_ACCESS definition. In case of a read-only restriction for such an access path the output of a software component is defined. But an input is defined without clear behavior description, which is again a problem when considering the mapping to a software component. This is especially problematic when thinking of the use of network variables in order to establish communication between configurations.

3.2.2 IEC 61499

The aim of the IEC 61499 standard is to provide "a generic architecture (…) for the use of function blocks in distributed industrial-process measurement and control systems. This architecture is presented in terms of implementable reference models, textual syntax and graphical representations” (IEC 61499-1, 2005, Section 1). The trigger for this new standardization work for ACSs (standardization work started 1992 when the IEC 61131-3 standard has just been finished) came from visionary studies of the Iacocca Institute (1992) and research programs. Especially the *Holonic Manufacturing Systems* (HMS) project is very important in this context, as depicted for instance by Christensen (1994). Key requirements that have been considered during the development are next to distribution and reconfigurability:

- ***Portability:*** The ability of software tools to accept and correctly interpret library elements produced by other software tools.
- ***Configurability:*** The ability of devices and their software components to be configured (selected, interconnected, parameterized, and assigned to locations) by multiple software tools.
- ***Interoperability:*** The ability of devices to operate together in order to perform the functions specified by one or more distributed applications.

The standard consists of four parts. Part one (IEC 61499-1, 2005) includes all definitions and models which are required to describe the architecture of the standard. Part two (IEC 61499-2, 2005) aims at software tool requirements and includes a *Document Type Definition* (DTD) for the representation of data types and library elements. Part three (IEC 61499-3, 2004) is a technical report which contains explanations on the elements and methodologies of the standard. Part four (IEC 61499-4, 2005) provides rules for compliance profiles, which are the means of the standard to cope with implementation depended details that are neglected in the standard itself. Next to the standard there are up to now two books available that describe IEC 61499 in more detail. Lewis (2001) gives a good introduction to the aims of the standard, but due to its early publication date it is not conform to the current standard. However, this is the case for Vyatkin (2007a), which is the teaching material of a university course and introduces the concepts of IEC 61499 by the use of the FBDK tool.

The *Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation* (O^{3}neida) organization pays attention to the further development and industrial application of the IEC 61499 standard. Next to several other activities especially a working group [40] on the development of compliance profiles has been founded in order to specify open issues within the definitions in the IEC 61499 standard. Actually, the topics of the working group are concentrated on execution semantic issues in order to provide similar execution behavior in heterogeneous system environments (IEC 61499 devices from different vendors). In addition also a compliance profile for the communiation via the CIP protocol has been established in a first version.

Basic architecture

The architecture of the IEC 61499 standard is described as a list of models. Figure 5 includes several of these models which can be considered as basic architecture of the IEC 61499 standard:

- ***System model:*** A system consists of devices, which are interrelated by some communication network, and applications.
- ***Device model:*** A device includes resources and interfaces to the process and/or the communication network.
- ***Resource model:*** The resource[6] is the element which executes FBs independently. It makes use of the interfaces from the device.

[6] The definition of a resource in IEC 61499 is not equivalent to the one of IEC 61131-3.

- ***Application model:*** An application consists of an FB network, which is composed of FB instances, their parameters and connections between the FBs. The execution of applications is determined by the event and the data flow within the FB network.
- ***Distribution model:*** An application can be distributed to different resources, which may be located on different devices. In Figure 5 'Application A' is distributed to different devices, 'Application C' is distributed to two resources within the same device. 'Application B' is distributed to 'Device 2' and 'Resource X' in 'Device 3'. The FB network visible in Figure 5 is allocated to 'Resource X' only, the communication within 'Application B' is capsulated in FBs. As the execution flow of an application is given by events, a distributed application will behave in the same manner as if it is located within one resource (delays due to communication neglected).

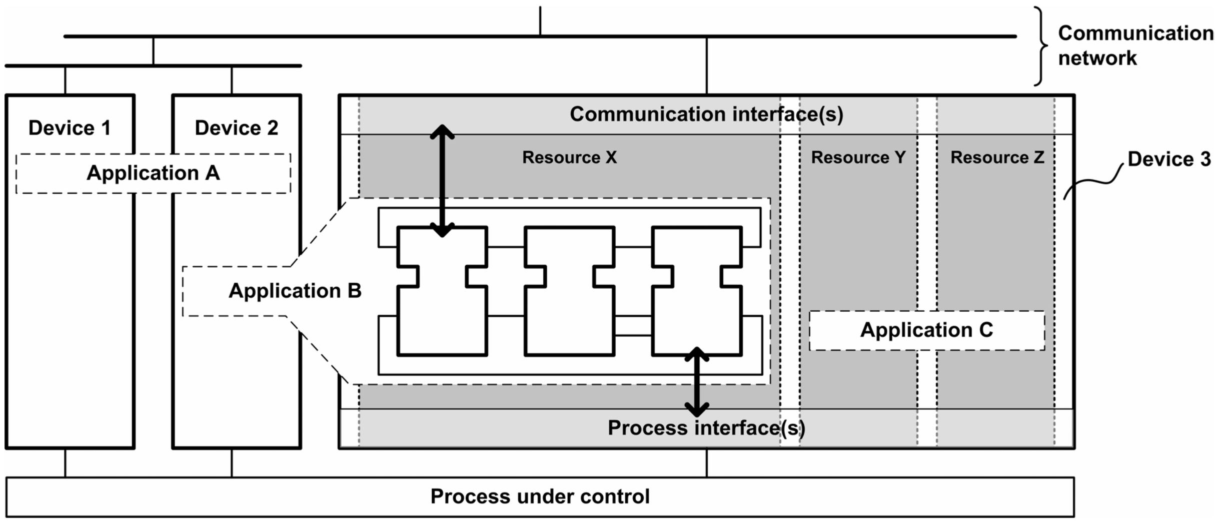


Figure 5: IEC 61499 architecture, based on (IEC 61499-1, 2005)

Function block model

The most important element of the IEC 61499 standard is the function block. In contrast to an FB defined by (IEC 61131-3, 2003) the interface is not only defined by variables but also by events. Figure 6 (a) depicts the graphical representation of an FB according to (IEC 61499-1, 2005). The upper part of the FB includes the event inputs ('EV1' and 'EV2') and event outputs ('EV3' and 'EV4'), the lower part contains the data inputs ('DI1' to 'DIx') and the data outputs ('DO1' to 'DOy'). The FB is executed as soon as it receives an event. In this case the data inputs related to this event are sampled. The means of the standard for the association of events and data is called WITH construct. Graphically the WITH construct is displayed as vertical line for an FB type, as for instance between 'EV1' and 'DI1' in Figure 6 (a). IEC 61499 does not define its own data types but makes use of the ones defined in (IEC 61131-3, 2003). After the execution of the internals of the FB output events may be initiated. There exists also an association between output events and output data, which denotes that these data outputs have been updated according to the FB execution before the corresponding output event has been produced. Therefore, the connection of an input event with an output event (including the corresponding data connections) will lead to a stable execution of FB networks with the latest data values. As an additional means the so-called adapter concept is introduced in the IEC 61499 standard. Herein, a bidirectional interface consisting of events and data can be specified and used as input or output for an FB. This can be used for simplification for instance if a specific interface is defined for some special purpose.

The internals of an FB can be categorized in three main types, which are called *Basic FB* (BFB), *Composite FB* (CFB), and *Service Interface FB* (SIFB):

- ***Basic function block:*** The behavior of a BFB is defined as event driven state machine, the so-called *Execution Control Chart* (ECC). Figure 6 (b) provides an example of an ECC. The ECC consists of states, that are connected via transitions. The transition condition can be an input event (e.g., 'EV1'), a Boolean expression (e.g., '1'), or the conjunction of both. When an input event triggers a BFB, all transitions from the active state are evaluated. There can only be one transition that fires (there exists an order for evaluation) and the active state of the ECC changes (there is only one active ECC state at one time possible). When a new state is entered, the corresponding actions are executed. An action consists of an algorithm, an output event (e.g., the action with 'EV3'), or both (e.g., the action consisting of 'Alg' and 'EV4'). There is more than one action possible for one state, and they are executed in a given order. If all actions of the active state have been executed, the transitions of the active state are evaluated. When a transition fires (e.g., '1'), than the ECC state changes and the corresponding actions are executed. Otherwise the execution of the BFB has finished. An algorithm within a BFB can be written in any programming language, but the IEC 61499 standard especially mentions the languages defined in (IEC 61131-3, 2003). An algorithm can use only input data, output data, and internal data of the FB type, therefore a BFB defines a very strong encapsulation of algorithms.

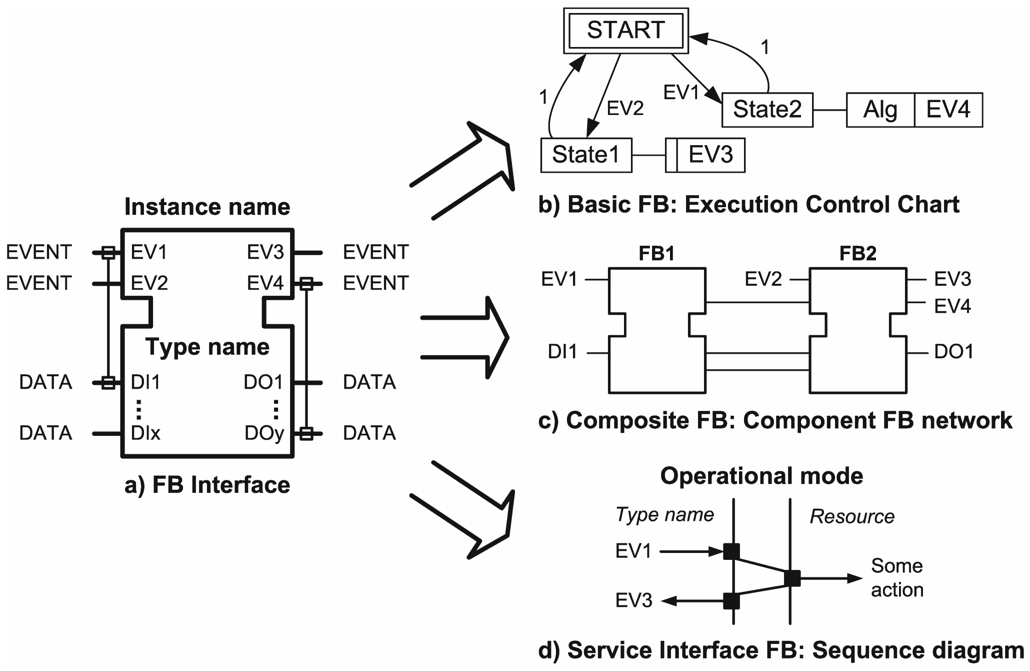


Figure 6: IEC 61499 FB model and FB types

- ***Composite function block:*** The behavior of a CFB is defined by an FB network. The FBs within a CFB are called component FBs, and there is no restriction to a special FB type. For instance, hierarchical structures can be designed by reuse of existing CFBs. The execution of a CFB is defined according to the event and data connections of the component FB network. A CFB cannot have internal data, since there is no possibility to use them within the component FB network. Figure 6 (c) depicts an example for a component FB network. In order to use the events and data from the CFB interface, their name can be used directly as inputs or outputs of the component FBs (e.g., 'EV1' or 'EV3').
- ***Service interface function block:*** The SIFB is a concept for the encapsulation of the interaction with external elements, that are not in the scope of the definitions of the IEC 61499 standard. Examples are interaction with the process or communication interface of a device, but also internal functionality of the underlying system (e.g., the

timer). In more general, within a SIFB any kind of functionality may be hidden. But the interface is equal to a BFB or CFB. In order to provide more information about the hidden functionality, a sequence diagram is defined by the standard. Figure 6 (d) depicts a simple example for one operational mode of a SIFB. If the event input 'EV1' is triggered, the SIFB will execute 'Some action' and emits the output event 'EV3' afterwards.

The IEC 61499 standard distinguishes two different types of SIFBs, the responder type and the requester type. The differentiation is based on the way the interaction between the FB network and the encapsulated functionality (which is called service) takes place. If the interaction is triggered by the FB network, then the SIFB is of requester type, and the interaction is called application triggered. This is the case for instance when outputs of the device can be written by the use of a SIFB. The sequence diagram in Figure 6 (d) describes exactly such an interaction. If the interaction is triggered by the service, then the SIFB is of responder type, and the interaction is called resource triggered. A typical example are time FBs, which are defined in (IEC 61499-1,2005, Annex A). For instance an E_CYCLE is triggered only once by the FB network to start its operation. Afterwards it emits events periodically triggered by the timing service.

Next to these FB types, the IEC 61499 standard defines the element *subapplication*. This element is similar to a CFB, but there exists no WITH construct and the component FBs and component subapplications may be distributed to different resources.

Management model

The IEC 61499 standard also includes a model for the management of resources and devices. It is stated that a management application may be modeled in order to facilitate the management of a device or resource. The management application itself is left open as an implementation-dependent part using SIFBs for communication and management. For the management SIFB a generic FB definition is included with a description of its functionality. The following management commands, see (IEC 61499-1, 2005, Table 6), are given based on a state machine for managed FBs, see (IEC 61499-1, 2005, Figure 24). Examples are provided in order to illustrate the management commands:

- ***CREATE:*** Creates an object such as an FB instance, resource instance, data connection, or event connection.
- ***DELETE:*** Deletes an object such as an FB instance, resource instance, data connection, or event connection.
- ***START:*** Starts an object such as an FB instance or an application.
- ***STOP:*** Stops an object such as an FB instance or an application.
- ***READ:*** Reads data from an access path[7], e.g., the data output of an FB instance.
- ***WRITE:*** Writes data to an access path such as the data inputs of an FB instance or resource instance.
- ***KILL:*** Makes an object unrunnable such as an FB instance.
- ***QUERY:*** Request for information on an object such as the FB types of a device, the FB instances of a resource or connections within a resource.

[7] An access path in IEC 61499 is a little bit different to the one defined in IEC 61131-3. It is defined as the association of a symbolic name with a variable, which is based on the concatenation of the instance names of the hierarchical elements, e.g., DEVICE_NAME.RESOURCE_NAME.FB_NAME.DATA_OUTPUT.

Compliance Profiles

A very important element of the IEC 61499 standard is the use of compliance profiles (IEC 61499-4, 2005) in order to specify open issues. The standard does not claim for completeness, but provides this element to unfold details of an implementation. A compliance profile is organized in three parts, which have been already mentioned above: portability, configurability, and interoperability.

Two practically relevant compliance profiles exist at the moment. The first one has been defined by James H. Christensen for his IEC 61499 implementation FBDK. It was the basis for the first inter-vendor feasibility demonstration of IEC 61499 within the HMS project and is called IEC 61499 Compliance Profile for Feasibility Demonstration [17]. Further IEC 61499 implementations have used this compliance profile, too, instead of using their own definitions. One example for a definition used in the compliance profile is the management application. The standard defines a generic management FB, but this compliance profile adds a concrete implementation of a management FB and its communication to a software tool. Also the semantics of this interface are defined as DTD for management commands. Communication FBs are another example, which are proposed in their interface within the standard. The compliance profile adds a definition for serialization of data streams in order to use simple Ethernet protocols for data exchange between devices. The second compliance profile has been established for the ISaGRAF tool [26], but it is not publically available.

Software components within IEC 61499

The main element of the IEC 61499 standard is the FB, which is treated very different in contrast to the ones discussed for the IEC 61131-3 standard. The first element for the discussion about software components within the IEC 61499 standard is therefore the FB in its different occurrences:

- ***Basic FB as software component:*** The interface of an IEC 61499 FB includes both data and events. Therefore also the execution of an FB is included in the FB interface. The behavior is given by the ECC and is well defined. There exist no hidden interfaces, as a BFB is only allowed to use input, output and internal data. A BFB can be considered as a software component.
- ***Service Interface FBs as software component:*** The SIFB hides its concrete implementation. But the IEC 61499 standard defines sequence diagrams in order to describe the functionality of a SIFB. If a SIFB is well described by the use of sequence diagrams, the SIFB can be considered as a software component. This is independent from the concrete type of SIFB. In case of a requester type SIFB, the execution behavior is defined by the interface as this type does not become active by itself. For the responder type SIFB, execution is triggered by the underlying services.
- There are two possibilities to undermine the representation of a SIFB as a software component. The first one depends on the underlying functionality encapsulated by the SIFB. There are no restrictions mentioned in the IEC 61499 standard for these services. Therefore it is possible to implement also some hidden interfaces, e.g. a similar construct as global variables. The second restriction concerns to the definition of the SIFB behavior by the use of sequence diagrams. This means may be not powerful enough to describe the SIFB behavior in all details (e.g., temporal order) or it is not used sufficiently.
- ***Composite FB as software component:*** The behavior of a CFB is defined by its component FB network. But the IEC 61499 standard lacks a concrete definition of the execution of FB networks, as this is discussed by the O^{3}neida working group [40] in general and especially for CFBs by Sünder *et al.* (2007). Another problem occurs by the use of SIFBs as component FBs within a CFB. If such a SIFB cannot be consid-

ered as a software component, then this is also true for the CFB. But also if each component SIFB fulfills all requirements of a software component, the CFB may lack a good description of its interface and behavior. The reason may be the hierarchy of composition levels within the CFB. A SIFB on the lower levels is hardly visible at the interface of the CFB, and only detailed analysis of the overall structure of the component FB network will clarify this situation.

- ***Subapplication as software component:*** A subapplication is very similar to a CFB and provides the possibility of distribution in addition. Already the considerations about CFBs have yield to the result that a CFB may be considered as a software component only in certain situations. The same can be stated for subapplications, if we neglect the impact of communication in case of a distributed subapplication.
- ***Application as software component:*** An application has no separate interface in form of an FB shape. Its interface emerges from the FBs used within the FB network, and especially by the SIFBs used. This fact and the above reflections about SIFBs within FB networks yield to the conclusion that an application cannot be considered as a software component.

The elements resource, device and even system may also not be considered as software components. For all these elements exist no separate interface descriptions as this is based on the FB network included.

3.3 Description languages

During the development of ACSs and especially PLC-based systems, a major improvement was achieved by the introduction of field buses. In the beginning, only very simple input/output field bus devices have been used, but their functionality increased more and more up to the point that nowadays each of such a component can be considered as a control device (see also the discussion about automation objects above). In order to handle the information about the various field bus devices, description languages have been developed for the engineering and operation of ACSs. We distinguish two different types of description languages—for simple field bus devices and enhanced system components. This determination is not very sharp because the functionality of also simple field bus devices increases.

With regard to the requirements stated in Chapter 2 there are two aspects that are strongly based on the description of control devices. Requirement (3) "Underlying system confiugration" aims at the intensive use of information about the overall control device and especially the system configuration. Therefore, these parameters need to be available in an appropriate manner by the use of a description language. Requirement (5) "Free programmable DSE" targets at the engineering support for DSE, which is the origin for description languages in general. The more information concerning a device is available in a standardized manner the more support can be included within the engineering tool by applying sophisticated algorithms on the available data.

Description of field bus devices

Many field bus systems, which have been developed in recent years, provide their own description language for the appropriate field bus devices. But there are also efforts for standardization of the description and integration of devices as for instance summarized in Niemann (2007), such as the *Electronic Device Description Language* (EDDL), *Field Device Tool* (FDT), and *Tool Calling Interface* (TCI). EDDL is defined in IEC 61804-3 (2006) and provides a generic language for describing the properties of ACS components. The main elements of EDDL concern device parameters, device functions, graphical representation, and interactions with control devices. The other two standards are defined by user organizations, in detail the FDT Group [11] and the Profibus and Profinet International organization [48].

Next to the description of the field device, which may again be based on the EDDL definitions, FDT aims at the integration of dedicated software for interaction with the field device into the engineering tool. Also the communication to the field device is part of the FDT specification, in order to achieve the flexibility of field device specific software elements within the engineering tool. The TCI specification concentrates on the integration of whole configuration tools into the engineering process of ACSs. Another important approach of a user organization is *OPC Unified Architecture* (OPC UA), specified by the OPC Foundation [41]. OPC UA defines the relations between clients and servers in order to achieve platform independence in spite of various kinds of systems and devices. One main aspect of OPC UA is the definition of information models, which define structure and semantics of data within the address space of the OPC UA server. Bender *et al.* (2007) present a concept based on OPC UA that integrates also the specific advantages of EDDL and FDT. This work shows that no description language exists that fulfills all needs of the customer.

The discussion above was driven mainly by specifications from user organizations that are often used in parallel for different types of applications. But there is also some more standardization work available in this field. A very basic definition of the elements of a device description is given in ISO 15745-1 (2003) next to more general rules on an application integration framework. The four elements of a device profile according to ISO 15745-1 (2003) are:

- ***Device identity:*** "The device identity object contains attributes which uniquely identify the device. Examples of such attributes are the manufacturer's identification, part number (…) and indication of the number and type of additional objects within the device."
- ***Device manager:*** "The device manager object represents the set of attributes (e.g., revision of the device identity object) and services (e.g., reset, configuration/run mode, retrieval of device manager objects' attributes) used to configure and to monitor a device integrated into the application system."
- ***Device function:*** "The device function object describes intrinsic function of a device in terms of its technology (e.g., mechanical limit switch, proximity sensor, ultrasonic sensor). The device function object differentiates the technology of the device from the application of the device. Examples of device function objects are analogue current inputs in milliamps, and discrete voltage outputs in volts."
- ***Application process:*** "The application process object represents a set of attributes and services that correspond to the application requirements captured in the attributes and services of the associated process profile. The application process object describes the behavior of the device in terms of the application, independent of the device technology."

The technical report IEC 62390 (2005) aims at the "development of device profiles for industrial field device and control devices, independent of their complexity". The basis is given by ISO 15745-1 (2003), whereas IEC 62390 provides a guideline for the profile development process. It is clearly mentioned that the scope is not limited to simple field bus devices. For instance programmable controllers and HMI devices are also targeted by this report. As general means for the establishment of a device description the *eXtensible Markup Language* (XML) [64] is recommended. XML is widely used also in ACSs during the last years, the two main application fields are given by Wollschlaeger and Wenzel (2005) as data exchange (e.g., within the OPC UA specification) and device description. Several user organizations have already proposed *XML Schemas* for the second purpose based on ISO 15745. Some are mentioned here briefly:

- ***XML@Profibus:*** This specification provides a technical guideline for the use of XML within the field buses Profibus and Profinet. Wollschlaeger and Wenzel (2005) describe its concepts as a common model that is able to integrate existing XML formats (from the various field buses covered by Profibus and Profinet International [48]) into this framework.

- ***CANopen:*** CANopen is one of the networks promoted by the user group *CAN in Automation* (CiA). Device descriptions have been originally developed on a textual basis, since 2007 a specification for the use of an XML schema for CANopen device descriptions (CiA DSP 311, 2007) has been published.
- ***Field Device Configuration Markup Language (FDCML):*** The FDCML specification (FDCML.org, 2002) follows a more general approach. It defines an XML schema which is again a markup language, but specialized for the description of field devices. (for a detailed description see also Appendix A)

Enhanced description of system components

The above considerations mainly concentrate on the communication aspects of a system component, more specifically field bus devices. But for an ACS consisting of several different components or automation objects it is important to yield an even more comprehensive description of these system components. The communication aspect is an important item within this enhanced description. But further topics need not to be neglected, too. The following discussion gives examples for an enhanced description of system components.

The *Total life cycle web-integrated control* (TORERO) project is next to the HMS project an example for the application of the IEC 61499 standard. The idea is based on devices that are capable to be self-configuring and self-maintaining. For example, a device can be introduced into the ACS by automatic plug and play mechanisms or it may be updated automatically by using information from a server of its manufacturing company. A basis for these capabilities is constituted by the TORERO device description, which includes all hardware and software information concerning the device. As described by Schwab *et al.* (2005) the predecessor of the FDCML specification, which is based on the same principles, has been used for the TORERO device description. Also in the case of the *Evolution Control Environment for Distributed Automation Components* (εCEDAC) project [8], which is closely related to this work, FDCML represents the basis for an extensive description of devices. Further details will be presented in Section 5.3 or are available in Strasser *et al.* (2007). Next to the support for the engineering process especially information concerning the properties of the hardware and software have been added to the device description in order to provide the basis for an enhanced engineering of systems, incorporating also the evaluation process.

Thramboulidis and Prayati (2001) analyze the current state of device description languages and argue, that there is no common model for the device specification which enables dynamic aspects and system management. Therefore they propose a field device specification based on IEC 61499 and IEC 61804 that adds the missing aspects and provides the basis for their function block oriented engineering support system. Their model is oriented to support both the development and the operational phase and consists of the main items network interface unit, resources, industrial process entity, and application management entity.

A very interesting approach is given in (VDMA 66430-1, 2006), the so-called *XML Interface for Robots and Peripherals* (XIRP). For the special application scenario of interaction between robots and processor-based peripherals (e.g., a vision system) XIRP defines a device description as well as a communication protocol, which specify a machine-to-machine interface and the rules for interaction. Several components (based on the discussion in Section 3.1 these are automation objects) are able to interact autonomously and provide a given functionality.

A device description is also an elementary part for sensor and actuator networks as addressed by the IEEE 1451 standard. Herein especially in part 3 (IEEE 1451.3, 2003) the plug and play aspect at the transducer level is depicted by the use of a common communication interface with an appropriate description file (the so called transducer electronic data sheet). Transducers according to this standard may be plugged into a compatible system and may be used without additional specific drivers, profiles or changes to the system. A similar approach is described in Kaiser and Piontek (2005), who investigate self-describing devices, which means that all information necessary for the use of such a device is stored within the device itself. The proposed device description includes three categories of information: general information, information about the physical connections and the temporal properties of the network, and information about the semantics of data provided by the device.

3.4 Software evolution and dynamic reconfiguration

The life-cycle of a software product consists next to the initial phase of software creation (programming) also of two further important activities: *Software Maintenance* and *Software Evolution*. Whereas software maintenance is defined by the IEEE 1219 (1998) standard[8] as "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment", there is no equivalent definition available for software evolution. Bennett and Rajlich (2000, Section 1.1) argue, that—although sometimes these terms are used interchangeably—"we shall use maintenance to refer to general post-delivery activities, and evolution to refer to a particular phase in the staged model" of the software lifecycle. "Software evolution takes place only when the initial development has been successful. The goal is to adapt the application to the ever-changing user requirements and operating environment." (Bennett and Rajlich, 2000, Section 2.2).

Several studies in the last 30 years concerning software evolution in computer science have yielded to the so-called laws of software evolution, which are discussed with respect to component-based software engineering in Lehmann and Ramil (2000). "The laws provide a phenomenology, a description of systematically observed patterns. The term 'laws' was used to indicate that they emerge from sociological, organizational and cognitive phenomena which appear to be to a large extent beyond the control of individual software developers and even managers." (Lehmann and Ramil, 2000, Section 2). In the following some laws with special relevance for this thesis are given:

- "***Continuing change (I):*** An E-type[9] system that is used must be continually adapted else it becomes progressively less satisfactory." (Lehmann and Ramil, 2000, Section 4.2)
- "***Increasing complexity (II):*** As an E-type system evolves its complexity increases unless work is done to maintain or reduce it." (Lehmann and Ramil, 2000, Section 4.3)
- "***Continuing growth (VI):*** The functional capability of an E-type system must be continually increased to maintain user satisfaction over its lifetime; where, in the context of the present paper, the term user applies to both component customers and end-users." (Lehmann and Ramil, 2000, Section 4.2)
- "***Declining quality (VII):*** Unless rigorously adapted to take into account changes in the operational environment, the quality of E-type systems will appear to be declining." (Lehmann and Ramil, 2000, Section 4.6)

[8] This standard has been superseded by ISO/IEC 14764-IEEE 14764 (2006), which focuses in more detail on the software life cycle process. Maintenance is considered in much more facets and its definition is more comprehensive accordingly. For our purpose the definition of IEEE 1219 (1998) is more useful.

[9] „An E-type program may, for simplicity, be defined as one whose acceptability depends on the perception, judgment and degree of satisfaction of appropriate stakeholder(s). Software used to solve a problem or address an application in a real world domain is in general of this type.“ (Lehmann and Ramil, 2000, Section 3.1)

Although these laws have been stated already long time ago, and the process of evolution was intended as an interception of operation of the software program (e.g., an update to a newer version), they are also applicable to today's practice in software engineering. Nowadays the need for software systems that are available without any downtimes is more and more claimed. Herein, the situation is equal for business software as well as control logic used in ACSs. The challenges for software evolution have been summarized in Mens *et al.* (2005) with the main statement that "the only way to overcome or avoid the negative effects of software aging is by placing change and evolution in the center of the software development process" (Mens *et al.*, 2005, Section 1). In the following some of these challenges with special relevance for this thesis are given:

- ***Preserving and improving software quality:*** "The challenge is to provide tools and techniques that preserve or even improve the quality characteristics of a software system, whatever its size and complexity." (Mens *et al.*, 2005, Section 3.A)
- ***Supporting model evolution:*** "Software evolution techniques should be raised to a higher level of abstraction, in order to accommodate not only evolution of programs, but also evolution of higher-level artifacts such as analysis and design models, software architectures, requirement specifications, and so on." (Mens *et al.*, 2005, Section 3.C)
- ***Formal support for evolution:*** "In order to become accepted as practical tools for software developers, formal methods need to embrace change and evolution as an essential fact of life". (Mens *et al.*, 2005, Section 3.E)
- ***Evolution as a language construct:*** "Programming (or even modeling) languages should provide more direct and explicit support for software evolution." (Mens *et al.*, 2005, Section 3.F)
- ***Need for better versioning systems:*** "Version control is a crucial aspect in software evolution, especially in a collaborative and distributed setting. (…) Therefore, the challenge is to develop new ways of recording the evolution of software that overcome the shortcomings of the current state-of-the-art tools." (Mens *et al.*, 2005, Section 3.J)
- ***A theory of software evolution:*** "It is necessary to develop new theories and mathematical models to increase understanding of software evolution." (Mens *et al.*, 2005, Section 3.Q)
- ***Post-deployment runtime environment:*** "There is an urgent need for proper support of runtime adaptations of systems while they are running, without the need to pause them, or even to shut them down." (Mens *et al.*, 2005, Section 3.R)

Dynamic reconfiguration

Especially the last noted challenge above, post-deployment runtime environment, has been the topic of research in the last years, called *Dynamic Reconfiguration*. This term is especially used for the change of architectures at run-time. Following the discussion presented by Wermelinger (1999), different issues need to be taken into consideration. By focusing especially on dynamic reconfiguration, the issue time is already defined to be meaning of "at run-time" (in contrast to off-line).

- ***Source:*** We distinguish between a change that may be triggered by the current state of components, topology of the architecture, or it may be asked for by the user.
- ***Operations:*** The fundamental operations of dynamic reconfiguration are adding and removing components and connections as well as querying relevant system properties at run-time.

- ***Constraints:*** Dynamic reconfiguration is constrained by any kind of property if it has to be preserved during the change process.
- ***Specification:*** The process of dynamic reconfiguration should be verifiable against the constraints.
- ***Management:*** The reconfiguration process may be managed in an explicit and centralized manner or management is implicit and distributed among the components.

Downtimeless System Evolution

This thesis aims at a combination of the above described topics software evolution and dynamic reconfiguration. Therefore, the term *Downtimeless System Evolution* is introduced, which explicitly faces the challenges of software evolution at run-time for the wide range of ACSs.

- ***Downtimeless:*** Changes have to be applied to the running system with as less disturbances to the process under control as possible.
- ***System:*** Although software is considered to be the central element that is under change, also changes in hardware—or more general changes of the overall system—are taken into account.
- ***Evolution:*** In order to keep a system satisfactory, the system has to be changed continuously during the whole life-cycle.

The remaining section presents a reference architecture for dynamic reconfiguration, which can be used to distinguish different scenarios that may take place within a component-based system and describes their interdependencies. Furthermore, several existing approaches of frameworks for dynamic reconfiguration with special focus on embedded systems and ACSs are briefly summarized.

3.4.1 Reference architecture for dynamic reconfiguration

In order to provide an almost general view on dynamic reconfiguration it is valuable to examine a reference architecture. We will use the conceptual framework presented by Walsh *et al.* (2007b) for this purpose, who systematically and consistently address problems and solutions related to dynamic reconfiguration. Although this work comes from classical computer science as their case study is a financial analysis system, the architecture is useful also for embedded systems and especially ACSs. The work is based on component-based system development, as it has been mentioned in Section 3.1.

Change types

The action of dynamic reconfiguration can be categorized into different change types, which describe how a system is adapted to the new situation. An evolution of a system may be conform to one or more of these change types. Walsh *et al.* (2007b) distinguish six different types of changes. Figure 7 depicts these change types and their interdependencies. There are two significant determinations for these change types:

- ***Effect on the component interdependency:*** For the characteristic of a change type it is important to separate inter-component change and intra-component change.
- ***Effect on the system signature:*** A change type may affect a structural change, a behavioral change, or both, with respect to the signature of the system.

The change types are defined by Walsh *et al.* (2007b) in general. In the following also their relevance for the application in ACSs is mentioned:

Internal change: Changes are applied to the internals of a software component. This addresses clearly intra-component changes and does not require any other types of change. A component may be changed regarding to its internal operations or state elements. Although the elements of ACS programming languages do not fit in all details to the software component paradigm, internal change can be applied equally.

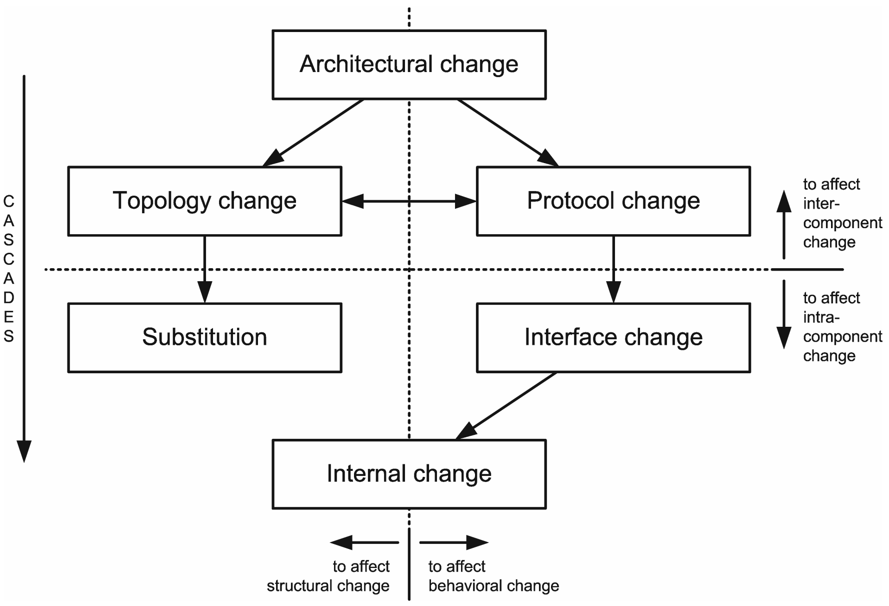

Architectural change
Topology change
Protocol change
Substitution
Interface change
Internal change
CASCADES
to affect inter-component change
to affect intra-component change
to affect structural change
to affect behavioral change

Figure 7: Change types and dependencies, based on Walsh *et al.* (2007b)

Interface change: Interface change is defined as "changing the interface of a component and therefore means changing provided and/or required services" (Walsh *et al.,* 2007b, Section 3.1.5). It affects only the behavior of a component and is an intra-component change. With regard to ACSs, the term services is limited to data and/or event interfaces as well as SIFBs within a component. An interface change needs to be considered always in the context with the protocol change, mentioned below. An internal change may be required due to the interface change.

Protocol change: A protocol change concerns the control and/or data flow between components. In the terminology of computer science this means change to service protocols. Again this kind of change affects only the behavior, but it affects at least two components and belongs to an inter-component change. The interconnection of components is the way of modeling control applications in IEC 61499 and at least for graphical languages also in IEC 61131-3. Therefore protocol change does not mean changes in communications of for instance a field bus, as this would match better with the terminology in ACSs, but of control applications themselves and changes to their behavior. According to the level of consideration within the elements of ACS programming languages, it has to be considered whether changes of interconnections belong to an internal change (e.g., in case of the component FB network of a CFB) or to protocol change (e.g., for applications in an IEC 61499 system).

Protocol change may require interface change and/or topology change. For the second issue this may be the case also in the other direction. Protocol change may be required also due to an architectural change.

Substitution: A substitution means the exchange of one component by another one. Therefore it belongs to structural changes, as the type of one component is changed. But it does not belong to inter-component changes, as there are no other components influenced. One component is removed and another one is added in order to fulfill similar requirements. Substitution may be required due to a topology change. Substitution can be considered equally also for ACS programming languages.

Topology change: The addition and/or removal of components is defined as topology change. Therefore this type of change is characterized by structural change and inter-component change. For topology change also substitution or protocol change may be necessary. Topology change may be part of an architectural change. The meaning of topology change can be projected in an equal manner also to ACSs.

Architectural change: "Architectural change means changing global or local system properties. A change to global or local properties can imply pervasive behavioral and/or structural system evolution to conform to these changed properties. This may mean changing whole configurations of components of a greater system." (Walsh *et al.,* 2007b, Section 3.1.1)

If we apply this definition to ACSs, the content can be transferred without adaptations. An architectureal change correspondingly means changing the control application of the ACS in general.

Integrity characteristics

A second important item of a reference architecture for dynamic reconfiguration concerns the management of integrity during the process of dynamic reconfiguration. Walsh *et al.* (2007b, Section 3.2) mention different kinds of integrity characteristics:

Global consistency: Any global property defined for a system (e.g. implied by its specification) needs to be preserved during dynamic reconfiguration.

Local consistency: If local properties of a system are defined they need to be preserved for the application of dynamic reconfiguration. Local properties need to be reconciled with global properties.

Active references: A component may have established communication paths as bindings of services. If these bindings are affected during dynamic reconfiguration, they need to be managed accordingly to do not violate any local or global consistency specification.

Dependent operation: There may occur dependencies between the operation of different components due to data flow interrelation. A change to an operation needs to be considered with regard to such dependencies.

Composite component: Similar to dependent operation, also the composition of components is affected by changes to one component. It has to be taken care that a changed component does not violate properties of composite components.

Constrained operation: A change to a component may be constrained by a dependency to the state of another component. Therefore, the process of dynamic reconfiguration has to be managed according to such constraints.

State management: In case of changes to or substitution of a component, a synchronization of the state within the components may be necessary.

Dynamically reconfigurable component-based system

Walsh *et al.* (2007b) also describe how to design a dynamically reconfigurable component-based system in order to fulfill the above mentioned change types and integrity characteristics. A domain model of dynamic reconfiguration needs to be set up, defined in three steps: (1) model of the primary concepts of the component-based system, (2) model of primary concepts of the context of dynamic change, and (3) combination of these two models in order to achieve the desired domain model of dynamic reconfiguration. As a very important item of the domain model also fault tolerance modes are introduced.

Walsh *et al.* (2007a) present the implementation of the above described model of dynamic reconfiguration by the use of explicit metaclass programming techniques. It is illustrated how global and local properties can be encoded and reconciliation of existing system properties and new change properties can then be resolved through a constraint solver.

3.4.2 Further approaches to dynamic reconfiguration

Many different approaches exist for dynamic reconfiguration. As a brief overview, only a few are mentioned. The first approach towards dynamic reconfiguration known to the author has been presented by Kramer and Magee (1985), who used the term dynamic configuration for “the ability to modify and extend a system while it is running”. They present the CONIC system, which consists of a configuration language (describing systems consisting of inter-connected modules), a programming language (for module types), and a distributed operating system. Their model for dynamic reconfiguration is based on a configuration manager, which is capable to translate requests for configuration changes expressed in the CONIC configuration language into commands to the operating system. The configuration manager is part of the target system and validates the change specifications against the current system configuration specification. In Kramer and Magee (1990) they enhance their methodology by separation between structural concerns and application concerns. They claim that in order to perform configuration changes it is important to do not lose application information and leave the application in a consistent state. For consistency during the change they introduce the quiescence property, which expresses that a node is both passive and has no outstanding transactions which it must accept or service. This is the basis to decide whether a change can be applied or not.

Appavoo *et al.* (2003) use the technique of hot swapping in order to support self-diagnosing and self-healing abilities of autonomous computer systems. Hot swapping is known also for dynamic reconfiguration in general and is accomplished either by the interposition of code (inserting a new component between two existing) or by a replacement of code (switching an active component to a new implementation). The infrastructure for hot swapping has to take different actions into consideration in order to perform a hot swap, namely triggering, choosing the target, swapping components, transferring the state, and potentially adding object types. Appavoo *et al.* (2003) depict a general description of such an infrastructure and the involved methodologies as well as a reference implementation in the open source research kernel K42.

Whisnant *et al.* (2003) describe a methodology for formally expressing the dependencies among processes in order to analyze whether dynamic reconfigurations are compatible with the existing configuration or not. Their system model consists of code blocks (they perform a computation triggered by events called operations), state variables (only accessed via executing code blocks), and threads (execute sequentially invoking code blocks). Reconfigurability is achieved by either adding or removing single operation bindings. The decision whether a reconfiguration will lead to a failure or not is based on the analysis of the dataflow within the system model. If an unsafe situation occurs due to the reconfiguration, the user is notified of the existing broken dataflow dependency.

Dynamic reconfiguration in embedded systems

Especially in the field of embedded systems, the topic of dynamic reconfiguration has been investigated by various researchers. The following list is again only a small excerpt of this very active field.

Yu *et al.* (2002) present a framework for a so-called live software update (this exactly matches with dynamic reconfiguration) in order to support mission- and safety-critical software applications. Their dynamic architecture is characterized by indirect addressing of modules and flexible communication via the publisher and subscriber model. The module proxy is essential for dynamic reconfiguration, as it is responsible for the management of e.g. the substitution of a module. For substitution, the upgrade protocol defines three phases: (1) uploading the new module, (2) switching operation to the new module implementation, and (3) removing the old module. Herein also the state of the module can be transferred, which must be implemented in the module by itself. By the use of a software upgrader element, the coordination of reconfiguration requests via a command line interface and the module proxies is achieved. The software upgrader is also used to synchronize the dynamic reconfiguration of several modules in parallel.

Rasche and Polze (2005) describe a framework for the run-time adaptation of component-based applications—Adapt.NET—based on the commercial component framework .NET [34]. In order to achieve reconfigurability, each component has to comprise a specific interface containing methods for connecting components, setting component parameters, or transferring the component state. Additionally they use an XML-based configuration description language to describe components in order to be able to identify components and connections involved in the reconfiguration process and to perform appropriate reconfiguration commands. The realization of adaptive applications, which can be adapted to certain environmental settings (e.g., state or attributes of components), is based on so-called adaptation policies. An adaptation policy defines the mapping of a monitored parameter and application configurations. If significant changes are detected, a reconfiguration request is generated and the framework executes the related dynamic reconfiguration. The Adapt.NET framework is able to achieve application consistency and furthermore also deadlines of application tasks can be met when necessary processor resources for potential reconfiguration commands are provided. Rasche and Polze (2005) demonstrate how the required resources can be calculated before run-time and included into the design and implementation of applications.

Angelov *et al.* (2006) present two versions of the COMDES framework for distributed embedded systems. The frameworks are defined as a set of executable models, whereas executable models are ultimately implemented as reusable and reconfigurable components. The second version has been improved especially in order to support statically allocated function units onto network nodes and hybrid timed event-driven state machines. Communication is based on signals that are exchanged at precisely specified time instants. Angelov *et al.* (2005) especially describe their implementation approach for components that include a reconfigurable state machine. By separating the executable code of such a component from the transition state table (represented as multiple-output binary decision diagram) reconfiguration is possible without re-programming. Therefore, internal change of a component is achieved on basis of the reconfiguration of state machines.

Stewart *et al.* (1997) developed dynamically reconfigurable real-time software in order to support reconfigurable robots based on port-based objects. A port-based object is an independent concurrent process whose functionality is defined by methods. The interface of a port-based object is given by input and output ports (used for the interconnection between port-based objects), resource ports (for the communication with sensors and actors), and configuration constants (used for the reconfiguration of generic components for specific hardware or applications). An application is modeled in the same way as a control engineer configures a system using transfer functions and block diagrams. Strictly speaking this was the original idea of port-based objects, in order to provide a simple modeling method for real-time software for control engineers. Communication between port-based objects is achieved by state variables that are synchronized by using local and global tables. Dynamic reconfiguration is supported by the framework, but it is clearly stated by Stewart *et al.* (1997) that there are no mechanisms integrated to perform a safe reconfiguration. It is mentioned that policies that ensure a stable execution during the reconfiguration are usually application specific. In their experiments they used a conservative approach for dynamic reconfiguration, namely the robot was temporally set at rest (velocity and acceleration are both zero) before the dynamic reconfiguration started.

3.4.3 Dynamic reconfiguration in automation and control systems

For the special purpose of dynamic reconfiguration in ACSs the above described approach cannot be applied directly as there are different means for system's programming in use. It is important to distinguish between vendors of ACS components and users of ACSs, as depicted for instance in Bouyssounouse and Sifakis (2005, Section 19.2). The first one face the problem of providing tools and runtime environments that are adequate also for dynamic reconfiguration, but with the restriction of an interface to the user in form of the programming languages mentioned in Section 3.2. None of the above mentioned approaches can be applied directly for ACSs. We will survey the approaches for dynamic reconfiguration based on the ACS programming languages:

IEC 61131-3

The IEC 61131-3 standard does not provide any means for dynamic reconfiguration. Nevertheless, nearly each IEC 61131-3 compliant runtime environment and engineering tool provides the possibility to change the elements of the IEC 61131-3 software model during operation. Of course, there are differences depending on the implementation of the runtime environment as well as the granularity of the reconfiguration, but commonly POUs can be changed dynamically. Hummer *et al.* (2007, III.B-1) give a description of the used methodology. The main idea is to use the cyclic execution of the control logic, which is the common way to program a PLC (although IEC 61131-3 enables also the triggering of tasks via the rising edge of a Boolean signal). There is a point in time when it is possible to change a POU without influencing the current execution of the control logic: This is between the finishing of the current cycle and the trigger for a new cycle of execution. Simply speaking this is equivalent to the quiescence property introduced by Kramer and Magee (1990). Based on the ratio between cycle time and execution time, there may be a big amount of time to execute the dynamic reconfiguration, which can be reduced to a readjustment of the pointer to the start address, if the new POU is already available. A typical methodology for state recovery is that the value of similar variables is copied to the new POU. Otherwise the variables of the new POU are set to their default values. Considering the approaches mentioned above this methodology for dynamic reconfiguration in IEC 61131-3 compliant systems is very similar to hot swapping.

There are several problems that occur when dynamically reconfiguring an IEC 61131-3 control logic without mentioning an appropriate engineering methodology (see Chapter 10 for an IEC 61131-3 system utilizing the εCEDAC methodology for dynamic reconfiguration), which have been depicted by Sünder *et al.* (2006c, Section 1). Fundamental problems exist based on the IEC 61131-3 software model:

- "The switching point in time cannot be determined because of the cyclic way of execution and the lack of information about the state of the system or application."
- "The reconfiguration of one task of an application interferes with all tasks of this application since all tasks have to be stopped because of the asynchronous cyclic execution of tasks. This leads to jittering effects."
- "The lack of fine granularity (task level) introduces high complexity in communication, memory management and re-initialization."
- "The reconfiguration of elements may lead to inconsistent states, e.g. deadlocks or token-proliferation in Sequential Function Charts (SFC)."
- "New elements start with their cold start initial value."

IEC 61499

The management model of the IEC 61499 standard directly addresses dynamic reconfiguration. There are of course no implementation techniques included in the standard, but the interface as well as the behavior for dynamic reconfiguration are mentioned. According to the short description of the management model in Section 3.2.2, the standard defines a state machine for managed FBs (Appendix B, Figure 63). Correspondingly, an FB is in an idle state after its creation. The START management command forces the FB into the running state, which means that the FB is operating input events. The FB may be stopped or killed by a management command, depending on whether a possible active operation should be finished or not when the management command occurs. By the use of an IEC 61499 compliant runtime environment such as the FBDK [15] and a compliance profile describing the concrete interface of the management application [17] an IEC 61499 control logic can be dynamically reconfigured. Hummer *et al.*, (2007, III.B-2) depict a test application which provides the possibility to simple send management commands to an IEC 61499 device. Demmelmayr and Zafari (2007) have used the FBDK for dynamic reconfiguration of a simple application under supervision of the author. In detail, the well-known programming exercise "Towers of Hanoi" has been adapted during operation by simply using management commands via an engineering station. These results have shown that dynamic reconfiguration can be achieved with the IEC 61499 standard, although the handling of pure management commands was very complex and means for state recovery are missing.

There are some approaches available that implement and enhance the dynamic reconfiguration capabilities of the IEC 61499 standard, which are mentioned chronological.

Brennan *et al.* (2002a) examine reconfiguration based on IEC 61499 already in a very early state of the standard during the HMS project. Their work is based on a draft version of IEC 61499, nevertheless their work can be applied in a similar way also to the final version. Brennan *et al.* (2002a) describe an enhanced model for function blocks that enables also the modeling of reconfiguration. Figure 8 depicts the general idea of two different kinds of control paths within an IEC 61499 application. Horizontally there exists the execution path which is responsible for operating the normal control flow modeled via the event and data connections of FBs. Vertically there is a configuration control path that can be used to model the reconfiguration of the control application. For the implementation of these two paths, they introduce two agent types: (1) the execution agent is primarily concerned with the FB execution and (2) the configuration agent is primarily responsible for implementing reconfiguration plans. In order to synchronize the execution of these two agents, a state machine is proposed. An FB therefore does not only provide an interface for data und events for control execution but also for reconfiguration execution. The configuration management application mentioned in Figure 8 again can be considered as an FB application. Brennan *et al.* (2002a) mention two different kinds of configuration management applications as the key to achieve an reconfiguration, which are discussed in more detail in Brennan *et al.* (2002b):

- ***Contingencies approach:*** "Within this form of reconfiguration control, contingencies are made for all possible changes that may occur. In other words, alternate configurations are pre-programmed based on the system designer's understanding of the current configuration, possible faults that may occur, and possible means for recovery." (Brennan *et al.* , 2002b, Section IV.B)
- ***Soft-wiring approach:*** "The basic idea behind this approach to reconfiguration is to enable higher layers to use higher-level reasoning to analyze the current configuration and plan for reconfiguration when required." (Brennan *et al.* , 2002b, Section IV.C)

For both cases, the configuration management application has to take care for a smooth transition from one configuration to another. For the contingencies approach this has to be modeled by the system designer. In the soft-wiring approach the higher layers reasoning about the reconfiguration also need to take care of the transition. The authors propose the use of agents for these higher layers. An implementation of this model for reconfiguration has been described by the use of real-time Java in Brennan *et al.* (2002a) and an FB operating system in Brennan *et al.* (2002b).

Thramboulidis and Zoupas (2005) present an implementation of an IEC 61499 runtime environment also based on real-time Java. This framework provides support for dynamic reconfiguration according to the interface defined by the standard. For the process of reconfiguration two different phases are proposed: In the first phase (low priority) all actions for the preparation of the dynamic reconfiguration should be included. These are for instance the download of a new FB type and the creation of a new FB instance. The second step has to be executed with high priority. Herein all actions that directly influence the active application have to be executed. The authors also provide timing characteristic measurements for the execution of management commands for different platforms. Thramboulidis and Zoupas (2005) present data for two different configurations of a personal computer, and Thramboulidis and Papakonstantinou (2006) mention data for an embedded platform. This is of special interest as the temporal behavior of reconfiguration is as important as its functional behavior, as already stated in Requirement (1) "Temporal behavior".

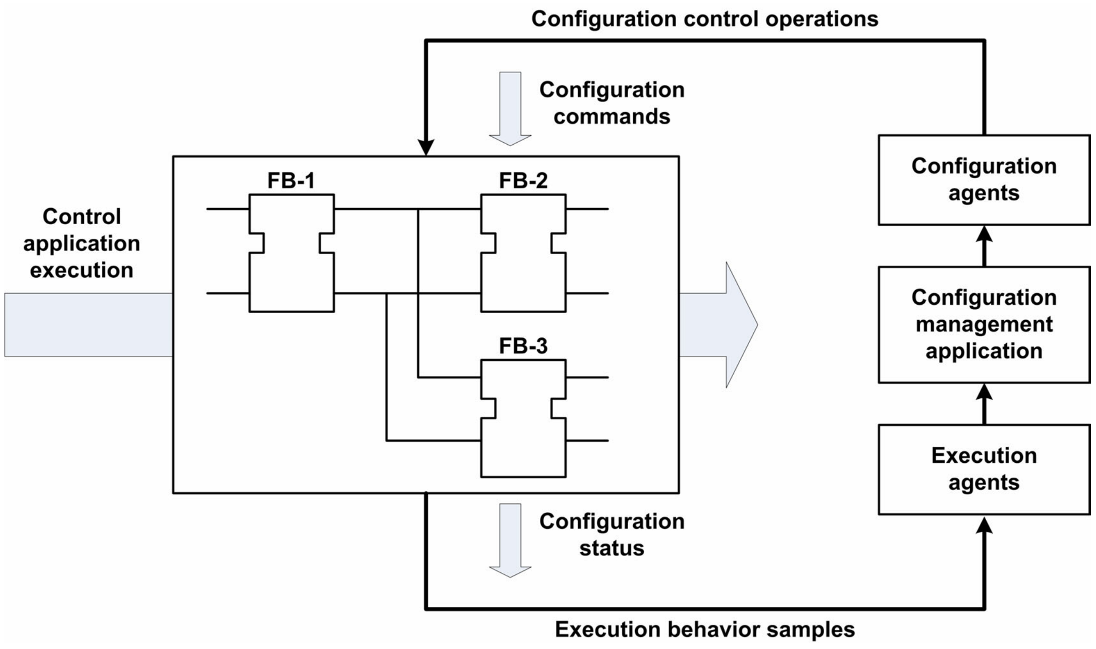

Configuration control operations
Configuration commands
Control application execution
FB-1
FB-2
FB-3
Configuration agents
Configuration management application
Execution agents
Configuration status
Execution behavior samples

Figure 8: Conceptual model for configuration/reconfiguration, (Brennan *et al.*, 2002a, Fig. 5c)

Zoitl (2007) investigates an IEC 61499 runtime environment utilizing two major characteristics: real-time execution of IEC 61499 applications and enhanced support for dynamic reconfiguration. The first one, real-time execution, is a necessity to provide full support for dynamic reconfiguration, as the control logic is constrained by the real-time characteristics of the process under control. Therefore, also the dynamic reconfiguration needs to be executed with appropriate real-time constraints in order to do not disturb control applications. Zoitl (2007) develops his concept on the basis of Kramer and Magee (1985). The interface to the device management is given by FBs which represent a certain management command. Similar to the idea of Brennan *et al.* (2002a) an FB application can be modeled in order to program a dynamic reconfiguration process. This runtime environment builds the basis for the implementation of the modeling approach for DSE and will be discussed in more detail in Section 4.2 and Appendix B.

The modeling of dynamic reconfiguration by using the above mentioned FBs has been described in Hummer *et al.*, (2007) as next steps towards downtimeless ACSs. The author of this thesis was part of the related research project εCEDAC [8]. A general description of the idea and a requirements analysis for the εCEDAC project is given for instance in

Strasser *et al.* (2005). The results have been described in a brief overview for instance in Rooker *et al.* (2007). The modeling method for DSE from the εCEDAC project will provide the general framework for this thesis (see Chapter 4 for a detailed discussion).

Programming language independent approach

Almeida *et al.* (2007) present a different approach in order to program ACSs. They utilize *Event-Condition-Action* (ECA) rules as formal method for defining the reconfigurable logic control. Almeida *et al.* (2007, Section I) describe the way of executing an ECA system as follows: "The occurrence of the event triggers the rule, which will start a query to check the condition, which determines if the system is in a particular state. The actions will fire if the conditions are satisfied." A monolithic structure of ECA rules is similar to a program consisting of a list of if-statements. Almeida *et al.* (2007) propose modular structures of ECA rules and trees (an enhanced rule with several conditions that build a tree). By applying changes to the ECA system, reconfigurability can be achieved. An important aspect for the reconfiguration as well as execution of ECA systems in general is the synchrony hypothesis, which states that the reaction of the controller takes negligible time with respect to the plant. The ECA system is independent of a programming language. Almeida *et al.* (2007) provide two examples for the implementation: modular finite state machines and IEC 61499 applications.

3.5 Transition management

A very important point in the above given description of the reference architecture for dynamic reconfiguration discussed in Section 3.4.1 are integrity characteristics. There are certain properties that need to be achieved during the process of dynamic reconfiguration in order to do not disturb the control logic during operation. These can be split up into two different kind of properties:

- ***General properties of the overall system:*** Commonly there are several applications running on a control device, and dynamic reconfiguration is applied only to a limited part of these applications. This kind of properties refers to those parts of the applications that are not affected by the reconfiguration. Therefore, the system environment as well as the dynamic reconfiguration's implementation need to take care that the integrity characteristics of the unaffected application parts are preserved.
- ***Properties of the application under reconfiguration:*** There are also integrity characteristics that apply to those parts of the application that are changed by dynamic reconfiguration. Therefore it is very important to integrate special mechanisms in order to retain these properties also when changes happen. These special mechanisms are usually called transient or transition management.

As already mentioned for instance by Stewart *et al.* (1997) the policy for ensuring a stable execution and as little disturbances as possible to the process is application specific. Therefore, a transition management policy cannot be stated generally and needs to be considered always in combination with the application. A field with a big amount of literature for such policies is control theory, where adaptive structures of controllers and plants are the topic of research since many years. The following discussion is affected by the methodologies from control theory, which may be adapted also to other application fields. As a starting point, we use the definition of transients from Kovacshazy *et al.* (2001) as

$$f_{tr}(n) = f(n) - f_{id}(n), \tag{1}$$

where	$f_{tr}(n)$	transient of the variable;
	$f(n)$	observed variable in the investigated reconfigurable system;
	$f_{id}(n)$	same variable observed in an ideal reconfigurable system.

For control loops the observed variable would be for instance the plant output. The ideal reconfigurable system would match with the plant output when the new controller has been used already for a long time. Kovacshazy *et al.* (2001) also provide measures in order to quantify the efficiency of a transition management policy based on Equation 1 as for instance the average energy of the transient or the absolute maximum of the transient.

Guler *et al.* (2003) provide an overview on different transition management policies, which can be summarized in four main topics. The principle idea is based on the substitution of a component (e.g., the controller):

- ***Output blending:*** Herein the old and the new component work in parallel, and their outputs are merged by some functional relations, whereas the transition starts with the old configuration and ends with the new configuration. The functional relation can be arbitrarily complex, a simple example would be a linear function. The method is meaningless especially for the substitution of controllers, as during the transition both controllers are not fully integrated in the closed loop.
- ***Parameter blending:*** This method concerns internal change, especially of a controller. When the new controller's structure is very similar to the old one, the parameters of the controller can be blended during the transition. Therefor, a functional relation between the parameter settings of the two controller settings is applied (similar to output blending).
- ***Transient management:*** Simon *et al.* (2001) provide a methodology in order to add a so-called anti-transient signal to the controller's output (similar to disturbance variable compensation), which is calculated based on the models of the plant and the controller. They claim that transient management can compensate both changes—of the controller as well as the plant. Simon *et al.* (2001) strongly restrict this approach to the prerequisites steady state of the control loop and a constant reference signal.
- ***State initialization:*** This methodology calculates the initial state of the new controller according to special algorithms in order to reduce transients during dynamic reconfiguration. Simon *et al.* (2000b) describe several approaches. Two very simple strategies have been already mentioned in Section 3.4.3 for IEC 61131-3 compliant systems. In the state zeroing method all state variables are set to zero, and in the state preserving method similar state variables are copied from the old controller to the new one. A more sophisticated method with a significantly higher reduction of transients is the output fitting method. Herein the state variables of the new controller are calculated so that it produces the same output as the old controller. An analytic solution for this problem may also include deviations of the output signal. Simon *et al.* (2000a) provide considerable simulation results of these different strategies by using a two link planar robot.

There are manifold results available from the field of transition management research. For instance Simon *et al.* (2000a) investigate the influence of the control logic's structure regarding to the occurrence of transients and show that there is a significant effect during the execution of similar dynamic reconfigurations within different structures. Kovacshazy *et al.* (2001) apply similar methodologies for transition management to reconfigurable signal processing channels. Guler *et al.* (2003) provide a generic pattern for transition management as well as graphical modeling for the above mentioned scenarios. These patterns may also be structured hierarchically enabling the coordination of several dynamic reconfiguration processes as well as transition management in distributed systems. Steffen (2005) investigates reconfiguration based on failures of actors and sensors in control loops. He introduces a so-called reconfiguration block which is in between the nominal controller and the faulty plant. The purpose of the reconfiguration block is twofold: On the one hand the faulty plant faces a reconfigured controller and on the other hand the nominal controller faces a reconfigured plant (the faulty plant including the reconfiguration block models the nominal plant).

3.6 Verification by model checking

The technique of verification by using model checking has been invented independently in the early 1980's by Clarke and Emerson (1981) in the United States and Queille and Sifakis (1981) in France. Both approaches utilize temporal logic (in detail branching time) in order to specify the desired system behavior. Since that, a lot of research and progress has been achieved in order to improve the capabilities of verification by model checking. First industrial applicable results have been presented for verification of hardware design and communication protocols, as the complexity in these fields is quite limited. Nowadays research is focused on software design and code, or also on the combination of software and hardware design.

A lot of literature exists about the field of verification by model checking. We will use two references for this work, Clarke *et al.* (1999) and Huth and Ryan (2004). There is a big variety of symbols and notations in use within the literature, and also the two works mentioned use different nomenclatures. We will follow the symbols and notations used in Clarke *et al.* (1999).

Model checking is a process that consists of three main steps: modeling, specification, and verification. These steps can be described briefly as follows:

- ***Modeling:*** The basis for model checking is a model of the system, which can be given in any description language of a model checker. Generally the model is given in some sort of transition system. The model may be compiled from a given design, but due to limitations in time and memory abstraction may be used to eliminate irrelevant or unimportant details.
- ***Specification:*** Model checking is based on temporal logic. In detail the combination of temporal logic with automatic algorithms for verification of a given model was the starting point for the research in model checking. A specification is the summary of properties that need to be comprised by the model. A high number of various dialects and languages of temporal logic exist (see Section 3.6.2), whereas in practice their use is restricted to the given model checker.
- ***Verification:*** Verification means the execution of a model checking algorithm with the specification and the model as input. The result is the answer, whether the model satisfies the specification or not. If the answer is no, the model checker provides the user in most cases with an error trace. The error trace is a counterexample for a checked property; a path in the state space of the model where the specification is violated. The generation of a counterexample is an important aspect for the (re-)design and debugging of a system.

The model checking problem (Clarke *et al.*, 1999, Chapter 4) can be described as follows. Given is a model M that represents a finite-state concurrent system. The model includes a set of states S. A specification that is given as temporal logic formula f expresses some desired properties of the model. Then model checking aims at finding the set of states in S that satisfies f:

$$\{ s \in S \mid M , s \vDash f \}. \tag{2}$$

Normally some states exist within the model that are designated as initial states. Then the model satisfies the specification if all initial states are in the set. The definition of the satisfaction relation $\vDash$ depends on the used temporal logic and is given for CTL* (see Section 3.6.2) for instance in (Clarke *et al.*, 1999, Section 3.1).

Following the classification criteria given in (Huth and Ryan, 2004, Section 3.1) for the determination of the verification approach, model checking fulfills the following characteristics:

- ***Model-based:*** The system description is represented by a model *M*. This is in contrast to proof-based approaches, where the system description is given as a set of formulas.
- ***Automatic:*** The degree of automation is another criteria for verification approaches. Model checking can be executed completely automatic.
- ***Property-verification:*** Model checking verifies whether a given system satisfies a given specification or not. But it does not determine whether the given system covers all the properties the system should satisfy (this would be called full-verification).
- ***Concurrent, reactive systems:*** The intended application fields for model checking are systems that may be hardware and/or software. Their characteristics are concurrent (instead of sequential) and reactive (instead of terminating) behavior.
- ***Post-development:*** The earlier verification is used in the course of system development, the greater are the benefits in terms of reduced rectification costs. Model checking is a post-development methodology, that means the model is built from a given design.

A general model: Kripke structures

Many possibilities exist to model concurrent, reactive systems. Concurrent systems are often given by the text of a program, utilizing shared variables and communication via message passing. They may be of synchronous or asynchronous type. Reactive systems are characterized by frequent interaction with the environment. They usually do not terminate. For both kinds of systems Clarke *et al.* (1999) propose to use a general type of state transition system called *Kripke structure* in order to capture this behavior. The general characteristics of such Kripke structures are that states exist (a snapshot of the system that captures values of variables at a particular instant of time), changes of the state are described by transitions, and computations within the system are depicted as an infinite sequence of states (the change from the previous state is given by some transition). A formal description of a Kripke structure is given in (Clarke *et al.*, 1999, Section 2.1) as follows:

“Let *AP* be a set of atomic propositions. A Kripke structure *M* over *AP* is a four tuple

$$M = (S, S_0, R, L) \tag{3}$$

where
1. S is a finite set of states.
2. $S_0 \subseteq S$ is the set of initial states.
3. $R \subseteq S \times S$ is a transition relation that must be total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $R (s , s')$.
4. $L : S \rightarrow 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

Sometimes we will not be concerned with the set of initial states S_0. In such cases, we will omit this set of states from the definition. A path in the structure *M* from a state *s* is an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$ such that $s_0 = s$ and $R (s_i , s_{i+1})$ holds for all $i \geq 0$.”

State explosion problem

When we think of an automatic algorithm for deciding on a given problem, the theory of *computability* needs to be kept in mind. "In particular, it shows that there cannot be an algorithm that decides whether an arbitrary computer program (written in some programming language like C or Pascal) terminates. This immediately limits what can be verified automatically." (Clarke *et al.*, 1999, Section 1.2) But as model checking is a technique for verifying finite state concurrent systems, an appropriate algorithm will terminate (theoretically). In order to determine a given specification, an exhaustive search of the state space of the system has to be performed by such an algorithm. The efficiency of the algorithm is an important measure for the applicability of model checking to practical problems (see the discussion below on enhanced model checking techniques in Section 3.6.1).

Nevertheless, the size of the model is a critical issue in model checking. Herein also the number of variables as well as the number of components of the system which execute in parallel are important measures. "The tendency of state space to become very large is known as the state explosion problem." (Huth and Ryan. 2004, Section 3.6.1).

This especially applies for the application of model checking in embedded systems design. Boyounnouse and Sifakis (2005, Section 1.4) state that "formal methods have scaled up drastically in the last decade, and this process is going to continue even faster. (…) Still, skilled engineers managed to use them by properly phrasing or decomposing their validation or analysis problems into traceable parts. Nevertheless, it is a constant and stringent need that formal methods and tools scale up to follow the increasing complexity of designs."

3.6.1 Enhanced model checking techniques

A common methodology for human beings to visualize the model checking problem is unwinding the given Kripke structure, which means starting from the initial state of the model and representing the overall system behavior by an infinite tree of all computation paths. The first model checking algorithms used such an explicit representation of the Kripke structure, but obviously these approaches lack efficiency in contrast to the state explosion problem. Many enhanced algorithms for model checking have been developed in recent years. We will give a brief overview describing the main ideas in improving the performance of model checking algorithms.

Efficient data structures—symbolic model checking

A key step in increasing the possible number of states in model checking was done by a symbolic representation of the state transition graphs. In detail, such a representation has been proposed by using *Ordered Binary Decision Diagrams* (OBDDs). OBDDs provide a very compact, canonical form for Boolean formulas. As a simple example, the transition relation can be expressed as a Boolean formula using two sets of variables, one for the old state and the other encoding the new one, which is represented by an OBDD. The model checking algorithm is based on computing fixpoints of *predicate transformers* that are obtained from the transition relation.

Abstraction

The technique of abstraction is described in Clarke *et al.* (1999, Chapter 13) as a methodology applied before the model of a system is constructed. In detail it aims at a reduction of states on a high level description of the system. There are two concrete techniques mentioned: cone of influence reduction and data abstraction. The cone of influence reduction analyses the influence of variables to the ones mentioned in the specification. If the influence of variables can be neglected, they can be eliminated for the creation of the model of the system. Data abstraction aims at a mapping from actual data values to abstract data values. For instance, a real number may be represented by three attributes: smaller than zero, equal to zero, or greater than zero. Again, a reduction in size of the original system can be achieved.

Simulation

In the sense of modeling a system, the simulation relation is an important means in decreasing the number of states. Clarke *et al.* (1999, Chapter 11) define that two given Kripke structures *M* and *M'* according to Equation 3—we say that *M'* simulates *M* (denoted by $M \preceq M'$)—if a simulation relation exists which associates each state in *M* a corresponding state in *M'*. Furthermore it can be shown that simulation is a preorder and given an ACTL* formula *f* (a detailed description of ACTL* is given in Section 3.6.2), $M' \models f$ implies $M \models f$. The model checking problem can be solved with the reduced structure *M'* instead of the original structure *M*.

Partial order reduction

This technique especially focuses on the verification of software. Concurrent software often consists of different processes which are performed independently—that is also called asynchronous (without global synchronization clock). This property often can be used to substantially reduce the size of the model. In detail, "it exploits the commutativity of concurrently executed transitions, which result in the same state when executed in different orders." (Clarke *et al.*, 1999, Chapter 10)

On-the-fly model checking

This technique is used in conjunction with model checking with automata (Clarke *et al.*, 1999, Chapter 9). Herein the automaton for both the model and the negation of the specification are generated and the emptiness of the intersection is checked. If the intersection is empty, the model satisfies the specification. On-the-fly model checking only generates the automaton for the specification. This automaton is used to guide the generation for the system automaton. It has been shown that this often leads to the construction of only a small portion of the state space before finding a counterexample for the properties being checked.

Bounded model checking

This technique uses the construction of a Boolean formula that is satisfiable if a counterexample exists. For the counterexample the length of the path is bounded. Starting from length *0* it is incremented until a proof is found. In certain cases the number of iterations of this procedure can be bounded for instance by the diameter of the finite state systems in case of safety properties.

Compositional reasoning

In many cases the overall model is represented by a composition of smaller parts. If it is possible to decompose the specification of the overall system into properties that describe the behavior of such smaller parts, model checking can be applied to much larger systems.

3.6.2 Formal specification by temporal logic

Model checking is used to verify properties of concurrent and reactive systems. Therefor it is necessary to specify also the dynamic aspects of these properties. It is not sufficient to investigate only fixed properties of the model, as this would be possible by using propositional and predicate logic. An appropriate means for specifiying dynamic system properties is temporal logic. Various dialects of temporal logic exist that differ in the provided operators and the semantics of these operators. We will start our considerations with a very powerful logic called *Computation Tree Logics* (CTL) CTL*, which can be seen as superset for the most common temporal logics CTL and *Liner-time Temporal Logic* (LTL). In addition we will investigate further derivatives such as ACTL*, real-time and stochastic time temporal logic. Next to various dialects of temporal logic there are also investigations on simplified understanding and application of temporal logics available, which are of special interest for ACS customers who are in common no experts in computer science.

*The Computation Tree Logic CTL**

There are two different kinds of temporal logic that can be distinguished, branching-time logic and linear temporal logic. Branching time means that the model of time has a tree-like structure. There are several paths possible within this structure and the path that is realized in the future is not determined. In linear temporal logic the model of time is a sequence of states, extending infinitely often in the future. As the future is not determined in general, several paths are taken into consideration representing different possible futures.

CTL* combines both models of time. The following description is based on Clarke *et al.* (1999, Section 3.1). In CTL* formulas can use two different kinds of quantifiers: path quantifiers and temporal quantifiers. As principle model for reasoning with CTL* a computation tree is examined, which represents the unwinded Kripke structure with one state designated as the initial state. Path quantifiers can be used in a particular state to specify whether all or some of the paths starting in that state have a given property. The two path quantifiers are

- **A** which means “for all paths” and
- **E** which means “for some paths”.

Temporal quantifiers describe properties of a path through the computation tree. Five basic operators exist:

- **X** (next time) specifies that the property holds in the second state of the path.
- **F** (future or eventually) specifies that the property will hold at some state on the path. This may be also the first state of the path.
- **G** (globally or always) specifies that a property holds at all states on the path.
- **U** (until) is based on two properties of a path. If the second property holds at some state of the path, the first property has to hold at every preceding state.
- **R** (release) is dual to **U**. The second property has to hold along the path up to and including the first state where the first property holds. The first property is not required to hold eventually.

The precise definition of CTL* formulas is split up into two kinds of formulas: state formulas (for a specific state) and path formulas (for a specific path). According to the definition of a Kripke structure in Equation 3 we assume *AP* to be a set of atomic propositions. Then the syntax of CTL* is given by the following rules (Clarke *et al.*, 1999, Section 3.1):

1. “If $p \in AP$, then p is a state formula.”
2. “If f and g are state formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulas.”
3. “If f is a path formula, then $\mathbf{E}\,f$ and $\mathbf{A}\,f$ are state formulas.”
4. “If f is a state formula, then f is also a path formula.”
5. “If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}\,f$, $\mathbf{F}\,f$, $\mathbf{G}\,f$, $f\,\mathbf{U}\,g$, and $f\,\mathbf{R}\,g$ are path formulas”.

The Computation Tree Logic CTL

Whereas CTL* included both branching-time and linear-time, CTL as a subset of CTL* only focuses on branching-time logic. In CTL each temporal operator has to be immediately succeded by a path quantifier. This means that there exist always pairs of one path quantifier and one temporal quantifier. CTL can be defined by exchange of the fifth rule of CTL* by the following restricted rule (Clarke *et al.*, 1999, Section 3.2):

5. "If f and g are state formulas, then $\mathbf{X}\,f$, $\mathbf{F}\,f$, $\mathbf{G}\,f$, $f\,\mathbf{U}\,g$, and $f\,\mathbf{R}\,g$ are path formulas".

Next to this basic definition also different extensions of CTL exist, for instance Starke and Roch (2002) include especially state transition information (see also Appendix C.2).

Linear-time Temporal Logic LTL

LTL is also a subset of CTL*, but in contrast to CTL it is focused on linear time. An LTL formula has the form $\mathbf{A}\,f$, where f is a path formula whereas only subformulas are permitted that consist of atomic propositions. In detail a LTL path formula can be defined according to (Clarke *et al.*, 1999, Section 3.2) by two rules:

1. "If $p \in AP$, then p is a path formula."
2. "If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X}\,f$, $\mathbf{F}\,f$, $\mathbf{G}\,f$, $f\,\mathbf{U}\,g$, and $f\,\mathbf{R}\,g$ are path formulas".

A very important aspect of temporal logic is the expressiveness of a given language. It is not possible to express any LTL specification in CTL and vice versa. For instance the LTL formula $\mathbf{A}(\mathbf{FG}\,p)$ cannot be expressed in CTL, and the CTL formula $\mathbf{AG}(\mathbf{EG}\,p)$ cannot be expressed in LTL. The disjunction $\mathbf{A}(\mathbf{FG}\,p) \vee \mathbf{AG}(\mathbf{EG}\,p)$ is a CTL* formula that is expressible neither in CTL nor in LTL. The choice of a specific temporal logic may be motivated also by its expressive power.

ACTL* and ACTL

An often used subset of CTL* is when only the path quantifier **A** is allowed. The restriction of CTL* to only utilize the **A** path qualifier is called ACTL*, and accordingly the same restriction of CTL is called ACTL.

Derivatives of CTL for real-time systems

Sveral extensions to temporal logics exist in order to support mentioning of time directly in the specification. Clarke *et al.* (1999, Section 16.3) mention for instance RTCTL which uses bounded operators such as $\mathbf{U}_{[a,b]}$, where $[a, b]$ defines the time interval in which the property has to be true. Another approach called TCTL has been introduced by Alur *et al.* (1990). In contrast to CTL the next operator **X** is omitted and all other temporal operators are extended by a timing condition such as $<c$, $\leq c$, $=c$, $\geq c$, and $>c$ (c as time value).

Derivatives for stochastic time

Another extension of temporal logics is the introduction of stochastic time. Again several approaches exist in this field as for instance probabilistic temporal logic PCTL, "in which a probabilistic quantifier of the form $\mathsf{P}_{\bowtie\lambda}$ is used in place of a path quantifier of CTL, where $\bowtie\, \in \{<, \leq, =, \geq, >\}$ is a comparison operator and $\lambda \in [0, 1]$ is a probability" (Sproston, 2004, Section 5.4). D'Aprile *et al.* (2004) discuss the use of *Continuous Stochastic Logic* (CSL) in model checking. CSL comprises the ability to specify qualitative and quantitative properties.

User friendly representation of temporal logic

The use of temporal logic for specification in the model checking process is highly supported by the various model checking approaches (see Section 3.6.3). But also a problem occurs due to the lack of good understanding of the expressiveness of temporal logics by engineers. There are two trends that can be observed in order to simplify the use of temporal logics:

- ***Timing diagrams:*** In Clarke *et al.* (1999, Chapter 18) the use of timing diagrams instead of temporal logic is mentioned for the specification of hardware design. Typically circuit designs are considered and timing diagrams are the natural way to express the behavior of the system. The timing diagram as specification may be used directly by adapted model checking algorithms or they are translated into temporal logic automatically. Vyatkin and Hanisch (2001b) and Vyatkin and Bouzon (2008) depict the use of timing diagrams for the specification in ACSs. In detail, they develop a specification language for timing diagrams which is used to generate a model of possible input behavior for a given system.

- ***Pattern:*** Similar to patterns used in software engineering for recurrent problems also patterns may be used for recurrent types of specifications. Dwyer *et al.* (1998, Section 3) introduced the term property specification pattern, which "is a general description of a commonly occurring requirement on the permissible state/event sequences in a finite-state model of a system". They set up a hierarchy of such patterns with additional characterization of the pattern scope. Each pattern consists of a textual description, temporal logic formulas for different languages, examples, and the relationship to other patterns. Dwyer *et al.* (1999) present a study on available specifications in literature whereas most of them have been instances of their proposed patterns. The patterns are publicly available via [53]. Another survey of patterns has been presented also in Meolic *et al.* (2001). A special kind of patterns for safety requirements with different classification scheme is presented in Bitsch (2001). Herein a selection process is proposed which leads to the selection of the best fitting pattern.

3.6.3 Approaches to model checking

Concerning the wide field of applications for verification by model checking and the different kinds of problems that may be evaluated, many different model checkers and model checking algorithms exist. This section aims at a brief overview by spotlighting some of these approaches. A more comprehensive overview is given for instance in Boyounnouse and Sifakis (2005, Chapter 7).

Finite state model checking

As described above model checking initially is concentrated on finite state models. The first model checker that was capable to manage a large amount of states for a practical application was developed in the PhD thesis of Ken McMillan (1993) and is called SMV. It is based on OBDD symbolic model checking and uses an input language that is based on the decomposition of a system into modules. Hierarchically structured designs are possible. The modules can be composed synchronously or using interleaving, and state transitions can be modeled either as deterministic or nondeterministic. The principles of SMV as well as an application are presented for instance in Clarke *et al.* (1999, Chapter 8). SMV supports the specification in CTL, LTL as well as further dialects of temporal logics. There are several versions available as for instance the original version from Carnegie Mellon University [52], a re-implementation with extensions as an open source project NuSMV [35], or TSMV [55] for the verification of timed Kripke structures by using TCTL.

A second important approach in finite state model checking is based on the application of model checking algorithms in the framework of automata. Herein an automaton is used as the model, and by using LTL formulas again in the form of automata very efficient on-the-fly model checking algorithms can be applied as depicted in Clarke *et al.* (1999, Chapter 9). The corresponding tool is called SPIN [51] which uses its own input language PROMELA in order to build formal models.

Continuous time model checking

There are different approaches available in order to use time within models. The notion of time has to be distinguished, whether discrete or continuous time is utilized. For discrete time model checking existing finite state model checkers can be utilized and enhanced such as for instance in TSMV. In case of a continuous time, the use of so-called timed automata as introduced by Alur and Dill (1992) has become the standard methodology. A timed automaton is a finite automaton augmented with a finite set of real-valued clocks. The automaton consists of locations (set of states) and labeled edges (set of transitions). Clock constraints can be used as guards on edges, and when a transition is taken clocks may be reset. A recent survey on the semantics and algorithms for model checking with timed automata is given for instance in Bengtsson and Wang (2004).

The UPPAAL tool [58] is one of the most notable approaches to model checking with timed automata. The UPPAAL modeling language comprises networks of timed automata with some extensions such as integer values (in addition to clocks) or urgent channels (for synchronization). Specifications are expressed in TCTL in general. The UPPAAL framework provides a rich featured environment for modeling, simulation and verification of timed automata. Another important tool for verification of timed automata is KRONOS [30].

Petri net based model checking

The theory of Petri nets has been established in the PhD thesis of Carl Petri (1962) in order to model the communication between asynchronous components in computer systems. As depicted in Peterson (1981) a big amount of research work was already available before model checking has been invented. Moreover, Queille and Sifakis (1981) used a special class of Petri nets, so called interpreted Petri nets, as the internal representation of the model in their first approach to model checking. A basic Petri net consists of four parts: a set of places, a set of transitions, an input function that represents edges from transitions to places, and an output function that represents edges from places to transitions. The dynamic behavior of a Petri net is represented by markings and their flow due to rules defined via the edges between places and transitions. Popular extensions of these models are colored Petri nets, that include different colors in order to distinguish markings. A recent survey on the Petri net theory is given for instance in Priese and Wimmel (2003), a collection of online services such as a tool database is available in [42]. Next to the analysis methodologies developed within the Petri net theory also model checking has been incorporated by some tools.

One special extension of Petri nets are *Net Condition/Event Systems* (NCES), which are a module based modeling approach introduced by Rausch and Hanisch (1995). A module interface utilizes event and condition inputs/outputs, the internal behavior is represented as Petri net. A detailed description of NCES is provided in Appendix C, an appropriate tool chain capable to provide model checking based on NCES is available in [61].

Probabilistic model checking

Based on the different approaches to model checking different extensions exist in order to incorporate also stochastic models into the model checking algorithms. Bause and Kritzinger (1996) discuss additions to the Petri net theory, in detail they discuss the introduction of Markov processes and queuing theory, which is the basis for Stochastic Petri Nets, *Generalized Stochastic Petri Nets* (GSPN), and Queuing Petri Nets. D'Aprile *et al.* (2004) depict the use of several tools for model checking of a GSPN model. These are ETMCC [18] (model checker for Continuous Time Markov Chains), PRISM [46] (a tool for analysis and model checking of different types of stochastic models), and GreatSPN [21] (a graphical editor and analyzer for timed and stochastic Petri nets). A survey on different approaches to model schekcing of probabilistic timed automata is presented in Sproston (2004).

An application of probabilistic model checking in ACS has been presented in Greifeneder and Frey (2007). Herein especially the situation in networked automation systems is taken into consideration, which consists of cyclic executed PLCs and sensors and actuators connected via some communication network.

Source code model checkers

For the model checking of software, the source code is the initial representation of the system. Several approaches exist already that are capable to handle source code of different programming languages as an input. For instance, Bandera [4] enables model checking of concurrent Java software by automatic conversion into the input languages of SMV or SPIN. A different approach is implemented in Java PathFinder [27], which provides a systematical exploration of all potential execution paths of a Java program in order to verify a given specification. A similar approach is used by VeriSoft [60] but without the restriction to a certain programming language. The C programming language is the basis for the tool SLAM [50] which is used for the verification of device drivers for Microsoft Windows. Herein the behavior according to the description of the application programming interface is checked. A more general approach for C programs is given in BLAST [5], which uses a counterexample-driven automatic abstraction refinement in order to construct an abstract model of the C source code for model checking. A similar approach is also utilized in the MAGIC framework [31] described for instance in Chaki *et al.* (2004). The abstract model simulates the model of the source code. If the specification is satisfied in the abstract model, the properties hold also for the original model. Otherwise, a refinement of the abstract model is calculated based on the information from the counterexample.

Model checking and dynamic reconfiguration

The discussion on model checking given above has one main prerequisite: there is a static model of the system in order to check whether it fulfills the specification or not. But also approaches exist that focus on dynamic reconfiguration of systems and therefore also dynamic reconfiguration of the model. We will concentrate our discussion of approaches to the field of embedded systems and especially ACSs.

Tešanović *et al.* (2005) present a model checking algorithm that is capable to verify properties of reconfigurable components. In detail, their approach is based on aspect-oriented software development which modifies given components during the establishment of a system by applying certain aspects. A component incorporates a set of reconfiguration locations where code may be changed during the aspect weaving. The verification is based on timed automata and the presented model checking algorithm checks whether properties of components are preserved upon the reconfiguration or not.

In ACSs especially the field of RMS initiated different approaches for the verification of dynamic reconfiguration. Herein formal models are used in the design process in order to generate the control logic based on these models. Kalita and Khargonekar (2002) present a methodology that combines both theorem proving and model checking based on timed transition models. The reconfiguration is described as the change of configurations which include models of the plant and the controller. Li *et al.* (2005) aim at the design of reconfigurable logic controllers by rewriting Petri net based controllers. Instead of carrying out a redesign and a new verification a method for rewriting the existing Petri net based controller is presented. A similar approach with Petri net rewriting rules is given in Alcaraz-Mejía and López-Mellado (2006). The dynamic reconfiguration is expressed directly as rewriting of the model.

A very important aspect within the process of dynamic reconfiguration is the behavior of the system during the execution of the changes. In Park *et al.* (2001) this is also taken into consideration for a controller capable to change within three pre-given modes. As a consequence next to the formal model of the different controllers and their control modes also the mode-switching logic needs to be included into the model of the system. This approach is based on Petri nets and automatic code generation from these models. The formal models of changes induced by dynamic reconfiguration are main elements of this work, as already stated in Requirements (4) “Modeling dynamic reconfiguration and (5) “Free programmable downtimeless system evolution” in Section 2.2.

3.6.4 Model checking in automation and control systems

The model checking problem in ACSs has to be enlarged especially due to the close interconnection to the plant and its requirements of controlled behavior. Hanisch (2004) describes this situation of a closed-loop modeling of the plant and the controller in more detail. The plant behavior dictates the design of the control logic. The interface between plant and controller is given by actuator and sensor signals. An integrated approach for the modeling of plant and control logic based on automation objects (the authors use the term mechatronic objects) is given in Bonfe and Fantuzzi (2003). Herein an automatic transformation into the input language of SMV is included and also the execution semantics of the control logic (the synchronous execution model of IEC 61131-3 and the asynchronous execution model of IEC 61499) are taken into consideration.

In the following we will survey several approaches for the analysis of existing code from the ACS programming languages IEC 61131-3 and IEC 61499 by model checking, with special focus on IEC 61499. According to Frey and Litz (2000) further reasons for the use of formal methods in PLC programming are the design of the control logic (with integrated automatic code generation) and the re-implementation of existing code on different platforms. They include also a survey on examples for the application of evaluation in ACSs. Further reasons for the use of formal models in ACSs are coordination activities and scheduling in manufacturing systems. Herein the high-level control is modeled and analyzed. Recalde *et al.* (2003) provide an overview on the use of Petri nets for this field of application as for instance a car manufacturing plant.

IEC 61131-3

The various elements of the IEC 61131-3 standard may be part of a verification approach by model checking. One important element with regard to verification is the structuring of POUs by the use of the SFC modeling languages. As described in IEC 61131-3 (2003, Section 2.6.5) the rules for the execution of SFC elements do not exclude failure situations such as unsafe SFCs (for instance uncontrolled behavior due to proliferation of tokens) or unreachable SFCs (the token may be locked and parts of the SFC may be unreachable). As the usual methods for validation—testing and simulation—cannot be applied easily in order to detect such situations also in complex SFCs, the method of verification by model checking has been used in different situations. As an example, Bauer *et al.* (2004) describe the translation of untimed SFCs into the SMV input languages and timed SFCs into Timed Automata. Based on these models and the dynamic model of the plant Bauer *et al.* (2004) depict the identification of errors in the control program by the use of model checking.

As this work utilizes NCES as modeling language we will concentrate on approaches in this field. One of the first applications is given in Hanisch *et al.* (1997) for the IEC 61131-3 programming language LD. The structure of LD is analyzed and a transformation into NCES models is presented. This transformation also incorporates timer function blocks. A more recent approach is presented in Hanisch *et al.* (2006) on basis of the practical example of a lifter. There are two different implementations of this example taken into consideration: by the use of LD control logic and visual flowcharts (a proprietary PLC programming language). They describe in detail the modeling of the plant in a hierarchical architecture of NCES modules and especially take into account the execution behavior of a PLC by execution cycles. Furthermore this approach uses data abstraction in the models in order to handle non-Boolean values by utilizing discrete thresholds. Lobov *et al.* (2006b) describe the translation of the IL dialect from the company Siemens (the language is called statement list) into NCES. Next to the fundamental transformation of IL commands into NCES models this approach takes into account also a very detailed model of the execution behavior of the PLC such as the scheduler of the operating system or organization blocks. Comprising also a model of the plant detailed analysis is possible. The modeling of the plant is done in a special manner, which is described in more detail in Lobov *et al.* (2006a). Here again a transformation is used for generating NCES models from plant models given in UML. Therefore a very powerful framework is established in order to simplify a closed-loop modeling of an ACS application.

IEC 61499

There are also many approaches available for the verification of IEC 61499. A survey on these approaches is given for instance in Frey and Hussain (2006, Section III) or Sünder *et al.* (2007, Section II). We will focus in the following brief introduction to the state of the art in formal modeling of IEC 61499 especially on the Requirements (1) "Temporal behavior", (2) "Execution semantics", and (3) "Underlying system configuration" mentioned in Section 2.1.

The first approach for a formal description of FBs according to IEC 61499 has been published by Vyatkin and Hanisch (1999). They use NCES which has a number of direct similarities with IEC 61499. NCES modules can be interconnected by event and condition arcs to bigger modules. The formal model of BFBs is based on the IEC 61499 standard, without taking into consideration the execution semantics of a given runtime environment. Event propagation is modeled directly by event arcs, the runtime scheduling is assumed to be concurrent and instantaneous. Further work based on this approach uses closed-loop verification of the controller and the plant. Enhancements of this early approach are for instance given in Vyatkin (2006), who describes especially the modeling of execution semantics of IEC 61499 FBs. In detail, the correct order of actions within an FB as well as the propagation of events over the network by the use of a scheduler which provides sequential operation of events is incorporated in the formal models. Pang and Vyatkin (2007) investigate on the representation of data and algorithms in NCES for the formal verification of IEC 61499. The work from Lüder *et al.* (2005) is based also on the concepts of Vyatkin and Hanisch (1999).

Wurmus and Wagner (2000) depict a formal description of IEC 61499 FBs and FB networks by using Petri nets. The event flow is represented by the flow of tokens, and especially the representation of the ECC within BFBs is taken into account. The approach also incorporates SIFBs utilizing timing services in a very simple manner, but all considerations are based directly on the standard without regard to a concrete runtime implementation.

Schnakenbourg *et al.* (2002) propose to model FBs using the synchronous language SIGNAL. They use clocks in order to assure the synchronization between the Execution Control Chart (ECC) and the input events. There is no model included for the propagation of events according to a concrete runtime implementation. Physical time is also not included, but the authors claim that this can be overcome by giving a value to the gap between two instances of a clock.

Khalgui *et al.* (2004) propose a state machine model compliant to the standard IEC 61499. To avoid unpredictable behavior in the case of a simultaneous occurrence of events, they propose to design an offline scheduling of an FB execution. They verify the scheduling correctness using a state machine model. By using this scheduler, a hard-coded execution model of a runtime environment can be implemented. Khalgui *et al.* (2006) include considerations also for distributed applications based on a temporal specification of exchanged messages between devices.

Zhang *et al.* (2004) consider the verification of IEC 61499 applications in contrast of safety-related system development. They propose a transformation of the IEC 61499 standard into finite state models without a concrete runtime environment or the physical time in mind. Based on the verification of BFBs, the verification of FB networks and CFBs is reduced to the verification of the connections between the pre-checked elements. In Zhang *et al.* (2005) the formal language for the overall specification of the software design cycle is UML. Herein, the special focus is on an integrated approach starting from system requirements till code implementation by using similar means, in detail the different UML models. For the verification a transformation from UML into finite state machines is given.

Stanica (2005) provides a study on modeling BFBs and FB networks together with a very simple model of the run-time behavior of a virtual IEC 61499 execution platform. His approach is based on Timed Automata and takes into account the physical time of algorithm execution. The formal description restricts the execution of algorithms to only one algorithm at the same time. But there are no models included to describe the propagation of events and further runtime behavior.

A rather new approach has been presented by Dubinin *et al.* (2006) using the verification engine of Prolog language, whose implementations contain a built-in deductive inference engine. Therefore, the class of properties that can be checked is extended to more substantial queries providing in return not only “yes” or “no”, but also the parameters explaining the reasons. For instance, questions like “at which values of parameter *X* does parameter *Y* belong to the interval [*a*, *b*]” can be formulated and checked. This approach is limited to BFBs in the current version.

Čengić *et al.* (2006) describe their formal model of the runtime environment FUBER [13], which they have developed based on interacting finite automata in Supremica. In this case the formal description includes many aspects of the runtime behavior. For instance, the event execution model specifies that each FB instance must wait for another instance to finish its event handling before it can begin its own event handling. Incoming events of an FB instance are stored in a queue; all FB instances waiting for execution are also handled in another queue. By the use of such a detailed formal description of the runtime behavior, they are able to prove in many details the behavior of the FUBER implementation. Physical time is not mentioned in their approach. As the implementation of FUBER is based on Java, the virtual machine as well as the underlying operating system need to be included to the models for the consideration of physical time.

3.7 Summary

The state of the art for this thesis consists of three main parts. First a general description of computer-based systems for automation and control purposes is presented, which can be characterized as embedded, real-time systems. The ACS customer is skilled in the abilities necessary for the operation of the plant and uses special programming languages (the widely used IEC 61131-3 standard and its successor IEC 61499) in order to operate the plant. General means from computer science are in use within the products of ACS component’s suppliers, but these cannot be anticipated to be used by ACS customers. The analysis of ACS programming languages shows that their concepts are quite different from modern programming disciplines, in detail software components and the appropriate software development cannot be matched directly, which is a problem for the generalization of approaches from embedded systems design.

Dynamic reconfiguration describes the changes of software during operation, whereas the new concept of DSE incorporates the overall configuration of hardware and software over the lifecycle of a plant. For computer science and especially component-based software development a reference architecture gives an overview on interdependencies of the different types of changes as well as the influence to integrity characteristics. To some extent dynamic reconfiguration is already possible in ACS. Special work has to be done in order to minimize disturbances to the plant during dynamic reconfiguration. Appropriate means for such a transition management are currently available especially for closed-loop control systems.

Verification by model checking has been successfully applied to hardware design and is in recent years also used in software design. The main problem is to decide whether a given system satisfies some specifications. Therefore the system as well as the specification have to be transformed into the input language of a model checking tool. Various means exist for the description of a system as a formal model. Specifications are expressed in some kind of temporal logic. In order to apply model checking in ACSs the model of the system needs to be generated as much as possible automatically (according to a closed-loop modeling paradigm the system comprises the plant and the controller) and the specifications have to be encapsulated in some user friendly format such as patterns or signal diagrams. Next to these prerequisites a concept for the evaluation of the effect of the execution of dynamic reconfiguration is missing in the current state of the art.

Chapter 4

New Engineering method for Downtimeless System Evolution

An appropriate engineering method is the basis of an evaluation approach for DSE. This is a very critical part especially for ACSs. Vyatkin *et al.* (2005, Section IV.A) state as an additional restriction for the software architecture of new systems: "Maintainability of automation systems is determined by the training level of the factory floor personal. For this reason, the human interface (which also includes means for re-programming) should not be radically different from what is used in the field now." Due to the special situation of ACS customers it is not feasible to apply highly sophisticated approaches such as presented in Section 3.4 directly.

This chapter starts with the definition of an engineering cycle for DSE, which is based on a special application in order to model the transition from a current system state to a new system state. This application can be split up into three sequences, providing a clear structure for an evolution step. For practical use different evolution steps need to be coordinated, which is again modeled in an application. Up to now this process only focuses on changes to the control logic of an ACS at run-time. But also hardware changes can be integrated into the engineering cycle, as depicted in Section 4.4.

4.1 Evolution engineering method

The process of engineering in ACSs can be described in a very simplified manner as depicted in Figure 9a in three steps:

1. ***Planning new ACS:*** First of all the requirements for the new ACS as well as a rough schematic of the planned functionality need to be set up.
2. ***Application engineering:*** Then an appropriate application is modeled and tested in order to fulfill the specified requirements.
3. ***Start of operation:*** At last the engineer has to download the application to the control devices and start the operation of the ACS. This phase may last for months especially for complex plants and manufacturing systems.

This very rough representation can be used for an overall plant as well as small parts of an ACS. The characteristic item is that when new requirements need to be added or requirements are changing, the process starts again from the first step (dotted arrow in Figure 9a). The plant or single control devices need to be stopped in order to download the new application. Then the ACS has to be started again.

In case of an engineering process with DSE, the initial steps are similar to those stated above. But as soon as the operation of the ACS has started, the process changes (see arrow from Figure 9a to Figure 9b). From now on the engineering process is characterized by the following steps (Figure 9b):

4. ***Planning system evolution:*** If new requirements exist or the requirements are changing, the new situation is planned. This is rather similar to the application engineering mentioned above.
5. ***Evolution engineering:*** In contrast of modeling the new application, the transition from the current system state to the new system state needs to be modeled. Herein it may be necessary to split up the overall system evolution into smaller steps.
6. ***Downtimeless system evolution:*** In contrast to the start of operation of a new application the actions for the transition to the new system state need to be executed at runtime of the overall plant.

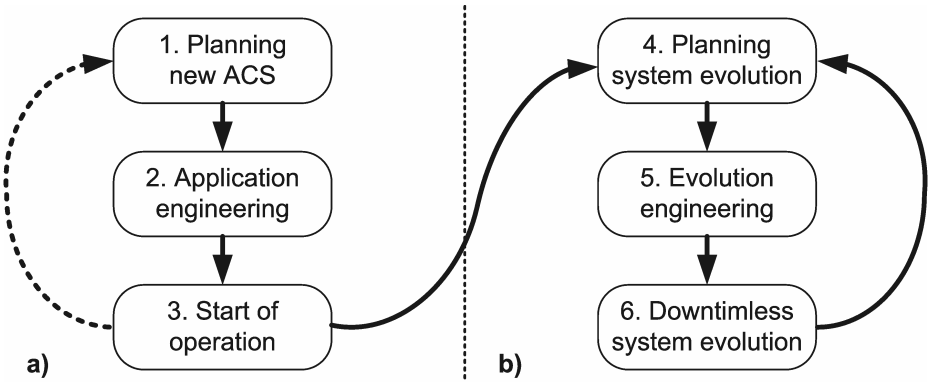

1. Planning new ACS
2. Application engineering
3. Start of operation
a)
4. Planning system evolution
5. Evolution engineering
6. Downtimless system evolution
b)

Figure 9: Engineering of ACSs (a) without and (b) with downtimeless system evolution

New or changed requirements can be handled by repeating steps 4 to 6 again and again. The engineering process for such an incremental enhancement of the overall functionality of the ACS is depicted in Figure 10.

Step 4, planning system evolution, is spilt up into two subtasks. First of all it is a prerequisite to have a detailed depiction of the current system state. Secondly the new application (the changed application) will be modeled rather similar to the application engineering mentioned in Figure 9a. There should be no difference for the ACS customer whether he models the initial application or a changed application. Then the evolution engineering takes into account the differences between the current system state and the proposed new system state. It is necessary to especially model the transition management in order to minimize the disturbances to the operating plant. The last step refers to the execution of the downtimeless system evolution. These steps describe the engineering process for one system evolution step, and as already depicted in Figure 9b the overall engineering process for an ACS consists of the recurrent application of these steps. A more detailed look at the tasks that have to be considered within this engineering cycle is given by the following description:

Acquire existing application

This first activity can be summarized as collecting all data available for describing the current system state. The used method depends on the possibilities of the used engineering tools and runtime platforms. For instance, all data of the ACS may be stored in some kind of data base or simply in the project file of the engineering tool. But as especially for bigger ACSs engineering takes place in larger teams, it may be necessary to countercheck these descriptions by directly interacting with the control devices. The collected data is used as input for the next step, the application modeling. In terms of an ACS based on the IEC 61499 standard the data consists of the system model including applications currently running in the system, the hardware configuration of the system (used devices and network structure), the mapping of the applications to the different devices and in addition a description of the hardware capabilities of the control devices.

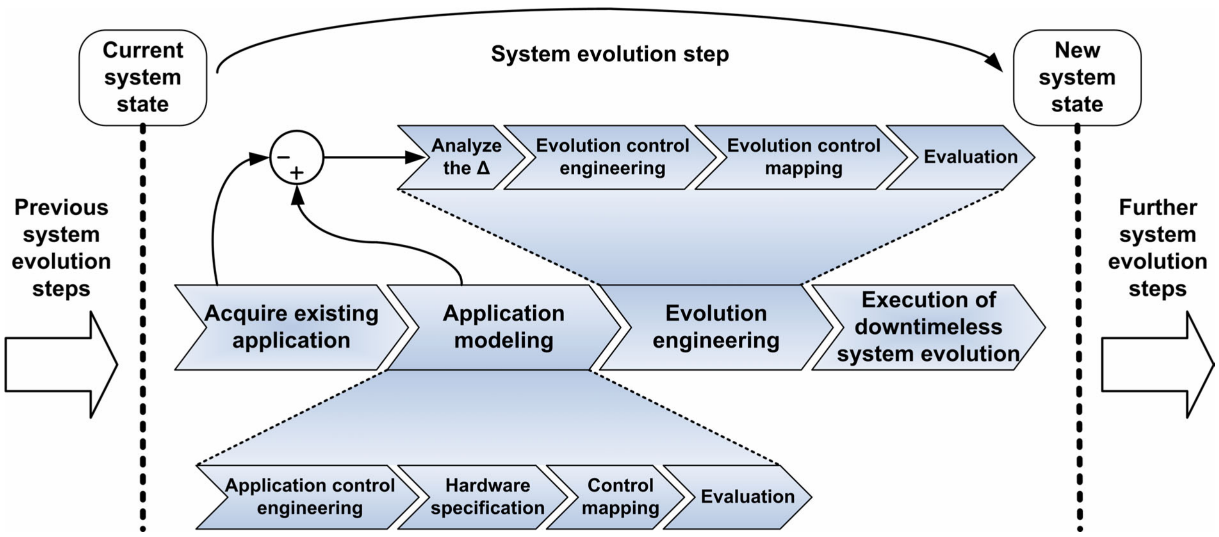

Current system state
System evolution step
New system state
Previous system evolution steps
Further system evolution steps
Analyze the Δ
Evolution control engineering
Evolution control mapping
Evaluation
Acquire existing application
Application modeling
Evolution engineering
Execution of downtimeless system evolution
Application control engineering
Hardware specification
Control mapping
Evaluation

Figure 10: Engineering cycle for downtimeless system evolution

Application modeling

The ACS customer models the new control application based on the existing application by adding/removing components and their interconnections. According to the application centered engineering paradigm of the IEC 61499 standard the application modeling consists of four steps:

- ***Application control engineering:*** The modeling of the control application is based on the existing application and focuses on the satisfaction of the new or changed requirements of the ACS. Furthermore the ACS customer has to specify application properties like real-time constraints.
- ***Hardware specification:*** As DSE focuses on both changes in hardware and software also the hardware configuration may be changed.
- ***Control mapping:*** The procedure of allocating applications or application parts to the available control devices is called mapping. A DSE may be also the relocation of application parts without changed software and hardware specifications.
- ***Evaluation:*** The evaluation process of the new application aims at checking the properties of the new system state. It is supposed that there is no DSE and only the properties of the new system state are examined. Based on the four evaluation methods presented in Chapter 1 especially simulation is used in industrial practice. Testing cannot be applied since the plant is still in operation, but is often used for the engineering of ACSs without DSE (Figure 9a). Deductive verification and model checking may be used as well.

Evolution engineering

The third step aims at the description of the transition from the current system state to the new system state. The main idea is that the ACS customer uses an application in order to model this transition. In detail, this evolution control application will be modeled utilizing the IEC 61499 standard. There are again four tasks that need to be applied:

- ***Analyze the Δ (Delta):*** The starting point for the evolution engineering is an analysis of the changes that have been modeled. A simple implementation may protocol all changes that have been applied to the current system state during the application modeling step. But there may be also more sophisticated algorithms that provide additional information in order to simplify the following tasks.
- ***Evolution control engineering:*** The ACS customer can use the same means for modeling the *Evolution Control Application* (ECA) as well as the control application. The

most important aspect is the provision of management commands in the shape of function blocks. The control properties and parameters for the DSE (especially for transition management) are specified in the same manner as control application properties.

- ***Evolution control mapping:*** Similar to the application centered engineering paradigm also the ECA is not restricted to a specific control device. It may be split up into parts and executed as a distributed application similar to a control application. This is especially important for synchronizing of changes on different control devices.
- ***Evaluation:*** Up to now no changes have been applied to the current system state. Within the next step, the DSE will be executed. In order to guarantee that the ACS will not break down due to the changes that will be applied, it is necessary to evaluate the process of DSE. As already discussed in Section 1.1 model checking is used for the evaluation process.

Execution of downtimeless system evolution

The execution of DSE consists of three tasks. Firstly the ECA has to be downloaded to the control devices. This is similar to the download of any control application. Secondly, the ECA has to be started, which means that it will execute the changes according to the constraints that have been modeled by the ACS customer. After the ACS has changed into the new system state, the ECA is useless and it can be deleted in order to leave the system in a clean state.

4.2 Basic evolution control engineering

As a first step we will investigate on an ECA as an IEC 61499 application for one system evolution step. As already depicted above, the process of DSE sets high demands on the underlying concepts and methodologies: Applications within the ACS have to be executed without disturbances. The system evolution has to be adapted to the special environmental conditions of the affected application part. Any failure during the evolution process has to be managed at least to such a degree, that the system is left in a defined state. The standard IEC 61499 already includes management commands for the configuration and reconfiguration of applications. But the standard lacks an engineering methodology for dynamic reconfiguration or even DSE.

The use of an application for modeling the DSE is closely related to the work of Brennan *et al.* (2002a), whereas we will use an IEC 61499 application utilizing basic reconfiguration services as described in Zoitl (2007) in order to model the interaction of the ECA with the control application. This topic has been presented in Sünder *et al.* (2006c), which provides the basis for this section.

4.2.1 Why is it necessary to freely program evolution control applications?

Before we start our detailed analysis of an ECA it may be helpful to reconsider the reasons for the need of a free programmable ECA. The Requirement (5) "Free programmable DSE" mentions that the scope of the system evolution may be very large and it is not restricted to any special field of applications. Based on the state of the art review presented in Chapter 3 there are two more reasons for this requirement:

- ***Transition management:*** One main requirement is the use of appropriate transition management strategies in order to minimize the disturbances to the control applications. As depicted in Section 3.5 various techniques especially exist for the field of closed-loop control. But the scope of dynamic reconfiguration in ACS is very broad as described for instance in Baier *et al.* (2007). Therefore, it is not sufficient to provide a defined set of transition management techniques in order to satisfy the ACS customer needs. The transition management strategy has to be modeled based on the special prerequisites of the concrete DSE.

- ***The nature of ACS programming languages:*** The component-based software development paradigm provides a good basis for dynamic reconfiguration as the different software components represent independent operational units. The rules for the interaction of software components within a given component framework can be used as basis for automatic scenarios of dynamic reconfiguration especially for inter-component changes (see the reference architecture in Section 3.4.1). Intra-component changes may be handled by using strict interfaces and software component specific implementations as mentioned in Section 3.4.2. But as the analysis of the programming languages in ACS with respect to the definitions of a software component have indicated, it is not possible to apply these concepts directly. As a consequence, the ECA cannot be established automatically and needs to be modeled freely according to the given control application.

4.2.2 Basic reconfiguration services

IEC 61499 already defines a basic set of commands to the enable management of resources, function blocks or connections. Based on the generic interface of the management function block a set of specialized function blocks should be available for modeling ECAs. But this set is not sufficient and has to be enlarged. The following gives an overview of missing instructions:

- ***Query of all internals of FBs:*** For instance the currently active Execution Control Chart (ECC) state or the value of an internal variable may be needed.
- ***Setting of all internals of FBs:*** A management FB should be able to set internal variables or to force the ECC to a dedicated state. In case of the latter action it must be possible to choose whether the corresponding algorithms or output events should be executed or not.
- ***Generation of events:*** The occurrence of an event at an FB input has to be controlled by a command for selective operation sequences. Such functionality may be simply modeled by using an event connection, but for engineering purposes also an appropriate FB may be useful.
- ***Sniffing of events:*** In order to synchronize the ECA with the control application events from the control application are an important input for the ECA. This functionality can be simply modeled by an event connection, but as already mentioned above it may be useful for engineering to provide a special function block for sniffing of events.
- ***Real-time execution of specific ECA parts:*** This is a general prerequisite also for the execution of control applications, as control applications are always constrained by the process under control. Any changes to the control application (modeled within the ECA) need to fulfill real-time constraints, too.
- ***Resolving of timing conflicts:*** In the best case no conflict will ever occur during the execution of control applications and ECAs. But this may not be possible in every situation, since for instance control devices have a limited amount of computational power. There has to be a means in order to define the procedure of execution for the competing application parts.

For this thesis we will use the IEC 61499 runtime environment which is described in Zoitl (2007). This runtime environment, we will use the term *Real-time Reconfiguration Runtime Environment* (R^{3}E), has been developed during the research project *Micro Holons for Next Generation Distributed Embedded Automation and Control Systems* (μCrons), which focuses on dynamic reconfiguration and real-time execution of IEC 61499. This runtime environment has been adapted in some special topics within the εCEDAC project and is available as part of the open source project *Framework for Distributed Industrial Automation and Control* (4DIAC). A more detailed description of the runtime environment is given in Appendix B, for our considerations we will use the terminology from Zoitl (2007).

The open points mentioned above can be summarized in two categories. On the one hand the real-time constrained execution of FB applications has to be included in general. Therefore Zoitl (2007) investigates the identification of event sources within an application. These event sources are always SIFBs, which are triggered by some external sources such as the timer or the network. In order to integrate a real-time execution concept, so called real-time event FBs (Zoitl, 2007, Appendix C), which provide parameters in order to define real-time constraints for the FB network that is triggered by these sources, have been defined. This concept enables the modeling of real-time execution within the control application without violating the concepts of the IEC 61499 standard.

The second enhancement is represented by basic reconfiguration services, which provide full access to the device management in order to control dynamic reconfiguration. The basic reconfiguration services are described in detail in Zoitl, (2007, Appendix A). Five categories of basic reconfiguration services exist:

- ***Structural services:*** The structural reconfiguration services provide mechanisms for changes to the structure of the control application. Herein creation and deletion of resources, FBs and connections as well as writing of parameters is summarized.
- ***Library services:*** The library reconfiguration services influence the library available within a device. The library includes resource, FB, and data types.
- ***Execution control services:*** The execution control reconfiguration services set the state of a managed FB or resource. The corresponding management commands are START, STOP, KILL and RESET.
- ***State interaction services:*** The state interaction reconfiguration services provide access to the internals of an FB by using the management command READ and WRITE (herein an enhanced functionality is necessary in contrast to the IEC 61499 standard).
- ***Query services:*** The query reconfiguration services can be used to establish the current system state by interacting with the control devices. For instance, lists of instanced FBs or connections can be polled.

In order to describe the interrelation between these basic reconfiguration services and the reference architecture for dynamic reconfiguration presented in Section 3.4.1, we have to define the association of the models defined in the IEC 61499 standard and a software component. For this thesis we will consider an FB as a software component[10]. The resulting types of changes and their dependencies are depicted in Figure 11. In comparison to Figure 7 all types of inter-component changes are available, as the IEC 61499 standard defines each FB as an entity. But for intra-component changes only internal changes are possible with the restriction to behavioral changes, as only state elements may be changed by their value. No basic reconfiguration services exist in order to change the type of an FB. In this case a new FB type needs to be created. Also substitution is not possible by using the basic reconfiguration services as single command. But it can be modeled within an ECA (see also the example presented in Section 4.2.3). The different basic reconfiguration services can be mapped to the possible change types as follows:

[10] Any kind of FB type will be considered as basic FB, although the analysis in Section 3.2.2 has identified different problems for CFBs and SIFBs.

- ***Protocol change:*** With respect to IEC 61499 a protocol change can be applied by changing connections and parameters within an application. The software components (FB instances) themselves will not be changed.
- ***Topology change:*** In contrast to protocol change, the topology change aims at modifications to the software components, the FB instances. Without influencing the behavior of the application there are two different kinds of changes possible: the substitution of an FB type without changing the interface and the relocation of application parts.
- ***Architectural change:*** The combination of both protocol and topology change yields to architectural change, which means any changes to an application within the ACS.
- ***Internal change:*** Based on the enhancement of R^3E in order to access also internal variables of an FB internal change is provided. But it is limited to behavioral change of the software component.

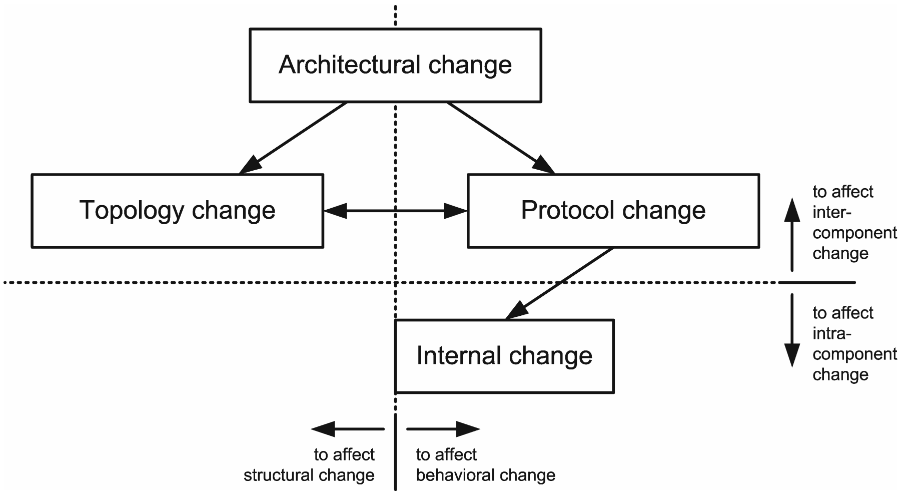

Architectural change
Topology change
Protocol change
Internal change
to affect inter-component change
to affect intra-component change
to affect structural change
to affect behavioral change

Figure 11: Change types within R^3E

Access to the device management

The interface to the device management is defined in the IEC 61499 standard by using a generic FB, which has been adapted within the IEC 61499 compliance profile for feasibility demonstration [17] as DEV_MGR FB type. This FB type is incorporated within a management application which simply defines a communication channel to the device management. A compliant engineering tool uses this communication channel in order to download applications to the device. Examples are the FBDK [15] or the 4DIAC platform [12], which utilize these definitions for the engineering tool as well as the runtime environment. Only one access mode to the device management of an IEC 61499 device exists: via the management application defined in the IEC 61499 compliance profile for feasibility demonstration.

The use of basic reconfiguration services sets much higher demands to the device management of an IEC 61499 control device. Each FB instance that represents a basic reconfiguration service includes an access mode to the device management. Figure 12 depicts this situation for an example device. The device includes several resources (‘MGR’, ‘Resource A’, ‘Resource B’) that execute function block networks. The management application described above is included within the ‘MGR’ resource (this is defined also in the IEC 61499 compliance profile for feasibility demonstration). The dotted arrows describe the access modes. For instance, the engineering tool uses the communication channel to send commands to the ‘DEV_MGR’ FB instance, which is an interface to the device management. According to the management commands sent to the device the device management acts within the resources of the device. Within ‘Resource A’ there is also an evolution control application which includes FB instances incorporating basic reconfiguration services. According to the execution of the ECA again the device management acts within the resources of the device. But now the device management is part of the execution of applications (in detail the ECA) and has to fulfill real-time constraints based on control applications the ECA acts on ('Application 1' and 'Application 2' are control applications).

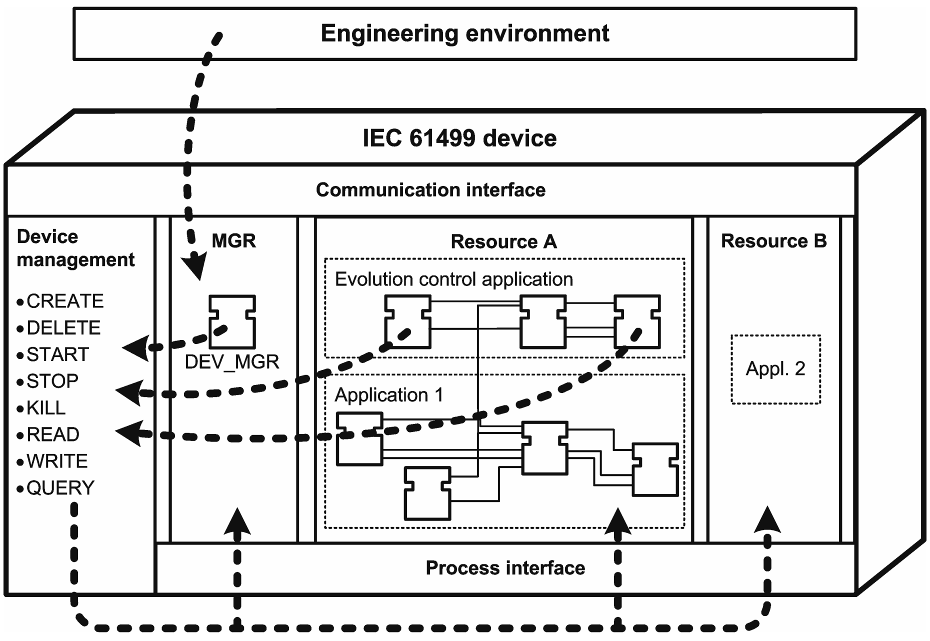


Figure 12: Different access modes to the device management of an IEC 61499 control device

4.2.3 Modeling evolution control applications

The main idea of this methodology is to control the DSE of control applications by an application, the evolution control application. This special application should make use of basic reconfiguration services in order to control another application. Furthermore the reconfiguration application can use any event and data flow in order to recognize the current system state of the application. For instance, the ECA may realize that the process has reached an idle state and would start the DSE. The event driven approach of IEC 61499 supports such a kind of synchronization with the control application in a very good manner. From a general point of view the following aspects should be mentioned for modeling ECAs:

- The ECA can be located on the local device. This enables a direct interaction to the concerned device/application without time delays due to communication networks. Real-time requirements of the system evolution can be fulfilled.
- The reconfiguration application has to interact directly with the corresponding application in order to react on the current system state and to coordinate the evolution process with the application behavior. By using event and data connections the ECA can be tightly coupled with the control application.
- Failure handling may be integrated directly within the ECA. A main requirement to the system evolution process is to leave the system within a defined state, even in the case of unexpected failures during the reconfiguration process, as otherwise the further operation of the ACS may be not possible.
- The system evolution process can be split up into characteristic sequences that represent typical sections within the execution of a system evolution step. Based on these sections patterns and libraries may be developed which will help the ACS customer to simplify the use of ECAs.
- Distributed ECAs are needed to model the interaction of the engineering tool and the devices and of course to synchronize the system evolution steps that need to be executed on several devices concurrently.

This general description provides some abstract guidelines for the modeling of ECAs. The concrete ECA highly depends on the control application and the required changes to this control application. In order to give a more detailed depiction of an ECA, we will use a closed-loop control circuit as an example for a typical control application. The proposed change to the control application is the exchange of the controller without disturbances to the process under control, which was a linear axis as described in Hanni (2007). Based on a short description of the control application we will describe the actions within the ECA in detail.

Example: Closed-loop control circuit

The control application is marked as grey shaded FBs within Figure 13 (lower part). The control cycle consists of four steps: write the output value from the previous cycle to the physical process (‘Set_Value’), read the current value of the control variable (‘Get_Value’), build the difference of current value and set point (‘Summing_point’), and calculate the control algorithm (‘Controller’). The additional FBs are used for the generation of the control clock (‘Clock’), receiving the set point (‘Get_Setpoint’) and the generation of the initial event for initialization (‘START’). The controller calculates the output value based on a proportional part and an integral part. The task for the DSE is to exchange the controller with a new type which includes also a limitation of the output value. This exchange should use appropriate transition management methods in order to minimize the disturbances to the control value.

Evolution control application

The appropriate evolution control application for this task is depicted in Figure 13 (upper part). The ECA consists of three typical sequences, which are available in any ECA. In order to execute the ECA, first of all the ECA has to be downloaded to the control device and all FB instances of the ECA need to be started (management command START) and, if necessary, they are initialized by the ‘INIT event’. After these actions have been executed successfully, the ECA is ready for the execution of the DSE.

Initialization sequence: The first sequence within the execution of the ECA is called initialization sequence and is responsible for preparation purposes. In detail no action within the initialization sequence should affect the execution of the current application. As a consequence these actions are not time critical and may be executed whenever there is spare execution time within the control device. The initialization sequence is started by ‘Start event’ mentioned schematically in Figure 13.

For the example given in Figure 13 the following actions are summarized within the FB instance ‘Initialization’. As the system evolution process dynamically changes the current application, Figure 13 does not depict a special situation within the system evolution step but a schematic of the overall process. All FBs or connections which are created within the system evolution step are drawn with dotted lines. Deletion of FBs or connections is not shown in Figure 13.

- Creation of the new controller (‘NewController’) as well as its input connections. The latter are the connections to the event inputs ‘INIT’ and ‘REQ’ as well as the data input ‘Delta’.
- Writing of the input parameters of the FB ‘NewController’.
- Starting of the FB ‘NewController’. This action influences the control application as the new FB needs to be executed as soon as a ‘REQ’ event is issued to the FB (this happens each control cycle).[11]

[11] In general the influence on the current application needs to be considered carefully. If it is not possible or intended to execute the new FBs the creation of the input connections or the START management command for

- The issue of the ‘INIT’ event to the ‘NewController’ FB. This action depends on the internal implementation of the FB. For the ‘NewController’ FB we expect that a ‘REQ’ event will only lead to a proper calculation if the FB has been initialized.

The second FB within the initialization sequence is called ‘Check_RINIT’ and has two responsibilities. On the one hand it checks whether all previous actions have been executed successfully. This can be done simply by the input qualifier ‘QI’, which is commonly defined to be true if the operation should be performed. On the other hand ‘Check_RINIT’ generates the starting event for the next sequence within the system evolution step. Information from the control application is necessary in order to synchronize the following actions with the execution of the system evolution. In case of a closed-loop control circuit a good starting point is the finishing of an execution cycle. The connection from ‘Controller.CNF’ to ‘Check_RINIT.CLK’ has been established during the download of the ECA. When all actions within the initialization phase have been executed correctly and the next execution cycle of the control application occurs, the output event ‘Check_RINIT.CNF’ will be fired in order to start the next sequence within the DSE.

Reconfiguration sequence: The second sequence within the execution of the ECA is called reconfiguration sequence and is responsible for the changes to the current application. Based on the preparations of the initialization sequence the current application is changed to the new application. Accordingly the actions within the reconfiguration sequence are time critical.

For the example of the closed-loop control circuit within the reconfiguration sequence the new controller ‘NewController’ needs to be initialized according to a transition management method. In this example the output fitting method (see also Section 3.5) is utilized. The following actions need to be performed:

- Reading of the internal state of the old controller ‘Controller’. The only value that needs to be considered for this kind of controller is the integral part, denoted as ‘Controller.I’.
- Then this value is used to calculate the appropriate internal value of the new controller ‘NewController’ in such a way that the new controller will produce the same output value for the current execution cycle. The FB ‘Transition’ performs this calculation by using data from the application. In detail the current output value (‘Controller.U’) and the current deviation (‘Summing_point.DELTA’) are used apart from the integral part of the old controller and the parameters of the new controller.
- Writing of the internal state of the new controller ‘NewController’. The WRITE management command in the ‘Set_Internal’ FB is used to provide this action.
- The FB ‘Rewire’ again includes several management commands for the purpose of moving the output connections from the old controller ‘Controller’ to the new controller ‘NewController’. By executing these actions the current application is changed to the new application as from now on the new controller ‘NewController’ calculates the output value ‘U’. In detail the connection from ‘Controller.U’ to ‘Set_Value.VALUE’ needs to be deleted and correspondingly a connection from ‘NewController.U’ to ‘Set_Value.VALUE’ has to be created. Additionally the INITO output event needs to be rewired.

The last FB ‘Check_RECONF’ again provides a check for the correctness of previous actions within the reconfiguration sequence. Next to a simple check of the ‘QI’ data input now also the correctness of the changed application needs to be taken into consideration. For the closed-loop control circuit the ‘Check_RECONF’ FB considers the current control value as

the new FBs may be moved to the second sequence, the reconfiguration sequence. If there is an initialization necessary for the new FBs the former option should be preferred.

well as the set point for a given number of cycles in order to recognize whether the new application is in a stable state. If yes, the output event ‘Check_RECONF.CNF’ will be fired in order to start the next sequence within the DSE.

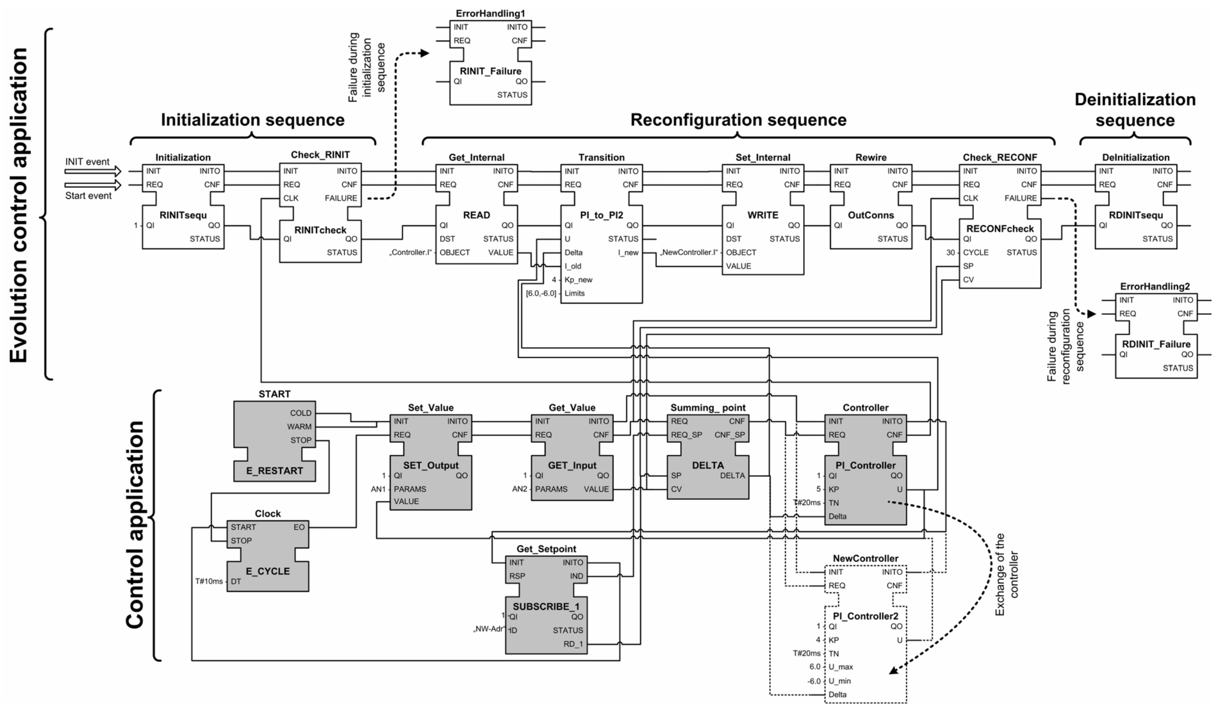


Figure 13: Downtimeless system evolution of a closed-loop control application

Deinitialization sequence: The third sequence within the execution of the ECA is called deinitialization sequence and is responsible for bringing the system into a clean state. As the reconfiguration sequence needs to be executed under real-time constraints, there is no time to delete FBs or connections which are not in use any longer and do not influence the behavior of the new application. These elements will be deleted within the deinitialization sequence. As the system is already in the new state, the actions within this sequence do not influence the behavior of the control application. The deinitialization sequence is not time critical.

The situation in the closed-loop control circuit at the beginning of this sequence is similar to the finishing state after the initialization sequence. Only the new controller ‘NewController’ and the old controller ‘Controller’ change their roles. Now the new controller is in operation, and the old controller is just present and of course also executed, but its output value is not used anymore. Therefore the actions within the deinitialization sequence are the same as for the initialization sequence, but with inverted order:

- Stopping of the old controller ‘Controller’.
- Deleting the input and output connections of ‘Controller’. It is not necessary to delete input parameters, but there may be some output connections that are still available and need to be deleted, too. For the example in Figure 13 this is true for the output connection of ‘Controller.CNF’, which has been used for the synchronization with the ECA.
- Deleting the FB instance ‘Controller’.

After the successful execution of the deinitialization sequence the new system state has been established without any unused elements from the old system state. The only thing that has to be done furthermore is to delete the ECA itself. This is again not time critical, as the ECA has no active interrelations with the current application.

Failure handling

As indicated in Figure 13, the ECA uses check points in order to trigger the different sequences. At these check points failure handling mechanisms can be applied, too. Depending on the application behavior, different algorithms are needed for failure handling. In this example 'ErrorHandling1' includes countermeasures if an error occurs during the initialization sequence, 'ErrorHandling2' reacts on a failure during the reconfiguration sequence. A failure may happen also during the deinitialization sequence. An appropriate FB is not depicted in Figure 13, also the two failure handling FBs are only indicated schematically. The actions that may be performed after each of the three sequences can be summarized roughly as follows:

- ***Failure during the initialization sequence:*** As at this point the current application has not been influenced the simplest action for error handling is to stop the ECA. A more sophisticated method may analyze the initialization sequence and retry those actions that have not been executed successfully. If the ECA is aborted, then those elements that have been already executed need to be canceled.
- ***Failure during the reconfiguration sequence:*** This is a very critical point within the ECA since the application is just in change. In most cases the error handling method needs to return the current status of the application into the old application. This may happen with some kind of transition management policy or without, depending on the kind of failure. Another possibility is to implement a retry for unsuccessful actions, but due to the real-time constrained execution of the reconfiguration sequence this may be critical.
- ***Failure during the deinitialization sequence:*** The DSE has successfully changed the ACS to the new application at this point. A failure influences only the process of cleaning up the old application. An error handling method may retry those actions that have not been successful.

Within this thesis we will investigate on a method in order to check that an ECA will not produce a failure during its execution. Therefore, we will not consider failure handling built within the ECAs.

4.3 Enhanced evolution control engineering

The above described methodology for establishing an evolution control application fulfills many requirements of a DSE. But there are some open aspects especially usability and clearness of this basic approach:

- The simple example of a controller exchange depicted in Figure 13 already visualizes the most important hindering reason for the application of such a methodology by the ACS customer: The ECA is rather big in contrast to the control application and adds considerable complexity to the overall ACS. The basic reason for this is of course that the DSE is a highly sophisticated action and changes without disturbances to the control application need significant efforts.
- The assignment of an ECA and the control application it concerns is not presented in a clear manner. The example given in Figure 13 shows a very simplified situation since only the interesting part of the overall ACS application is visible. Within control applications in industrial practice it may be much more complicated to consider a single system evolution step.
- The above described situation takes only one system evolution step into account. But it is necessary to coordinate several system evolution steps in their order or synchronize them in order to apply changes to different control applications simultaneously.

- The coordination and synchronization of system evolution steps needs to be possible also in distributed systems. The above described evolution modeling method does not limit the user to systems consisting only of one single device. But the introduction of a distributed ECA will again increase complexity.

In order to overcome these problems an enhanced evolution control engineering method has been developed which is based on the above presented basic methodology. This approach has been described for instance in Hummer *et al.* (2006).

Encapsulation of single ECAs

The execution of an ECA can be split up into three characteristic sequences: initialization, reconfiguration, and deinitialization. These sequences represent independent parts within the execution of a system evolution step. They are triggered at certain points in time and need separate consideration of successful execution and also failure handling. A first step within the enhanced evolution control engineering is to encapsulate these single ECAs within a defined containment. We call this containment the *Evolution Execution Control Function Block* (EECFB). Each of the three sequences should be operated and represented by the interface in an independent manner. The generic interface of an EECFB is depicted in Figure 14. A separate event input and output exists for each sequence as well as an input and output qualifier in order to represent status information. The generic interface can be described as follows:

- ***FB Initialization:*** As it is defined by the IEC 61499 standard SIFBs provide an interface for the initialization/deinitialization of the underlying services. The event input 'INIT' together with the data input 'QI' are used in order to start the initialization ('QI' is true) or deinitialization ('QI' is false) of the SIFB. The corresponding outputs are 'INITO' and 'QO', which state the end of the FB initialization together with its status ('QO' true for successful initialization and vice versa). As each basic reconfiguration service is a SIFB, the EECFB needs to provide such an interface, too.
- ***Initialization sequence (RINIT):*** The first sequence within the ECA is responsible for the preparation of the control application in order to reduce the effort for dynamic reconfiguration. This sequence has an interface similar to FB initialization, that is characterized by the key word RINIT (in order to describe that it represents the initialization sequence for the dynamic reconfiguration). Accordingly, the event input 'RINIT' and the data input 'RINIT_QI' can be used for starting the initialization sequence, and 'RINITO' and 'RINIT_QO' issue its result.
- ***Reconfiguration sequence (RECONF):*** The second sequence of an ECA includes the time critical dynamic reconfiguration. We use the key word RECONF in order to identify this sequence. The interface for starting and issuing its results is similar: event input 'RECONF' and data input 'RECONF_QI' as well as the event output 'RECONFO' and data output 'RECONF_QO'.
- ***Deinitialization sequence (RDINIT):*** The third sequence aims at the clean-up of the control application in order to remove unused FBs and connections. We use the key word RDINIT (deinitialization of dynamic reconfiguration). The interface is again similar and includes the event input 'RDINIT' and the data input 'RDINIT_QI' as well as the event output 'RINITO' and the data output 'RDINITO'.

This generic interface of an EECFB needs to be extended with parameters that may be useful for the ECA and of course also the event and data connections in between the control application and the ECA. It has to be mentioned that in contrast to an ECA modeled according to the basic evolution control engineering the EECFB does not include all elements necessary to execute the ECA. Each of the sequences is included as a separate part within the EECFB, but their interconnection to each other needs to be modeled outside of the EECFB.

This especially influences the 'CHECK_RINIT' FB within Figure 13, which has two different purposes. On the one hand it checks the correctness of the initialization sequence execution, which belongs to the internals of the EECFB. But on the other hand it is responsible to trigger the reconfiguration sequence, which now belongs to the external "wiring" of the EECFB, as otherwise there would be a dependency between the different sequences. But for the establishment of the EECFB there has to be taken care that such an dependency does not exist.

Another important aspect of the EECFB is its implementation. As far as we have described the EECFB it simply represents a containment for an FB network. Although we have used the word function block, the IEC 61499 standard provides two different means for such a containment: CFB and subapplication. It is a matter of the implementation, which version will be preferred (in Figure 14 the CFB version is depicted due to the used WITH construct). If the ECCFB is realized as a CFB, this FB type has to be available for execution within the runtime environment. This means, a CFB type declaration has to be created during operation. If the ECCFB is realized as a subapplication, only the component FBs have to be available for execution within the runtime environment. Although it can be assumed that most of the FBs used within the ECA are available (e.g., the basic reconfiguration services), there may be some FB types missing as for instance in order to provide an appropriate transition management method. These FB types need to be created within the runtime environment during its operation anyway.

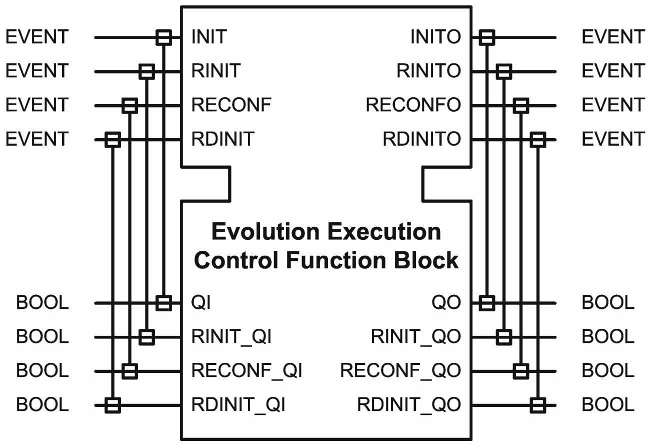


Figure 14: Evolution Execution Control Function Block type (implemented as CFB)

Region of interest taken into consideration within an EECFB

The second step for enhancing the evolution control engineering approach concerns to the association of an ECCFB to an application area it concerns. We call this application area *Evolution Region of Interest* (EROI), as it includes exactly those parts of the control application that will be affected by the EECFB. During the process of establishing the new application, the different EROIs occur based on the changes that are modeled within the current application. The following rules define the borders of the EROI:

- ***Creating/Deleting a function block:*** The EROI consists of the function block itself plus the corresponding halves[12] of the surrounding FBs it is connected with, because a significant temporal order of operations exists within the ECA, as for instance a deletion of an FB incorporates all connections of this FB.
- ***Creating/Deleting a connection:*** The EROI consists of the corresponding halves of source and target function blocks of the connection as well as the connection itself.

[12] For the consideration of the EROI we split up FBs into one half consisting of all event and data inputs and another half incorporating all event and data outputs.

- ***Creating/Deleting a parameter:*** The EROI consists only of the input half of the target FB and the parameter itself.

This association is very helpful for structuring the evolution engineering process. The different areas of DSE within the control application get their own EECFB that includes the necessary ECA for the intended changes. Figure 15b depicts this situation schematically. During the engineering process of DSE the ACS user may examine the control application and mark the changes that are necessary. These markings correspond to the EROIs (grey shaded circles) and as the next step the user designs an appropriate ECA in terms of an EECFB. Finally the coordination and synchronization of the changes to the different EROIs need to be modeled, which will be described below.

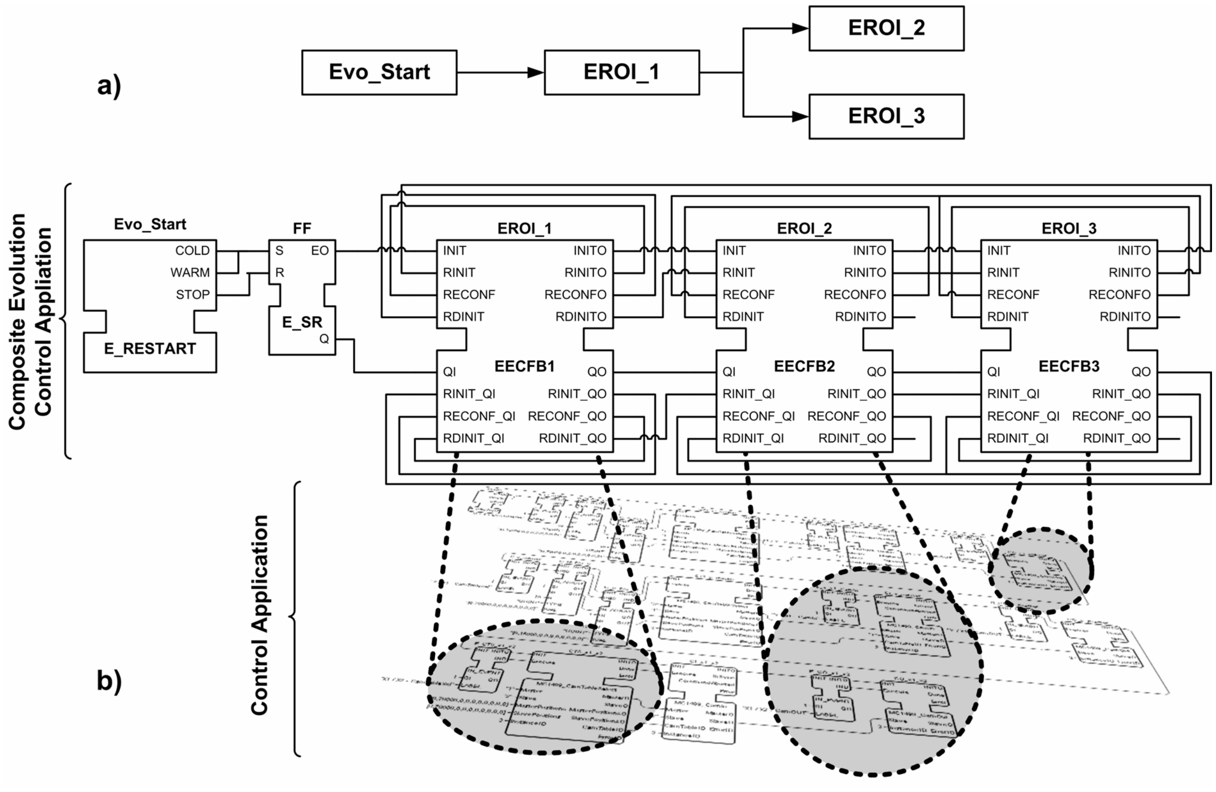


Figure 15: Composite ECA and its influence to the EROIs within the control application

The definition of EROIs opens up another important possibility for the simplification of evolution control engineering: the usage of templates for certain changes. It is obvious to design standardized procedures for recurrent activities such as the exchange of a controller FB with a distinct internal algorithm, or the exchange of an FB without transition management. This template EECFB needs to be adapted to the concrete control application, but most parts within the EECFB can be defined in advance. This is also an interesting aspect for companies which provide control applications or FB types to their customers. In order to update to a new version they are able to prepare an EECFB which may be applied by their customer without high work load.

Coordination of single ECAs

The third step for improving the evolution control engineering approach is the modeling of a *Composite Evolution Control Application* (CECA). The CECA consists of single ECAs as its main parts with an additional control logic in order to coordinate and synchronize their execution. Again the CECA is an IEC 61499 application and can be modeled with the same means as the control application. In detail the coordination is based on the three sequences within each ECA. The initialization and deinitialization sequences are not important since they do not influence the control application. But the reconfiguration sequence directly changes the control application and the actions within the reconfiguration sequence have to be taken into account. Additional interfaces (especially events for synchronization) may be necessary for a fine-grain synchronization between two EECFBs. The event FBs which are defined in (IEC 61499, 2005, Annex A) provide a first set of FBs that can be used within this synchronization.

Figure 15b depicts a CECA incorporating three EECFBs. Within these EECFBs no interdependencies exist, but their execution is modeled in a special temporal order. After the issue of 'COLD' or 'WARM' events from the 'Evo_Start' FB all three EECFBs are initialized sequentially. When this initialization has been successful (due to the connections of the input and output qualifiers the DSE will be stopped as soon as any action within the EECFBs is not executed successfully) the first EECFB 'EROI1' is executed. Herein all three sequences are executed sequentially without any further external synchronization mechanism. When this first system evolution step has been performed successfully, 'EROI2' and 'EROI3' execute their initialization sequence sequentially. As soon as both system evolution steps are ready for their reconfiguration sequence, the changes within both EROIs are applied simultaneously. When the reconfiguration sequence of these EECFBs has been executed successfully, deinitialization sequence will be executed independent from each other.

The execution order of EECFBs within a CECA can be easily visualized as depicted in Figure 15a. As already mentioned above only the reconfiguration sequences of the EECFBs within the CECA are of special interest for the execution order. The initialization and deinitialization sequences are necessary, too, but they do not influence the control application. The very simple schematic in Figure 15a provides a very good overview of the execution order and increases the usability of the evolution control engineering approach.

The modeling of distributed CECAs can be added in a simple manner based on the enhanced evolution control engineering method. When we assume that an EECFB and its associated EROI are related to only one device, a distributed CECA can be modeled by using communication FBs in order to spread the coordination and synchronization events and data connections via the communication network. In case of very tight coupled EECFBs this has to be handled appropriately.

4.4 Downtimeless system evolution with physical reconfiguration

The previous investigations for a modeling method for DSE have been focused on the control logic. But as stated in Section 3.4 we want to take dynamic reconfiguration of both software and hardware into consideration. This dynamic reconfiguration of hardware, in short *physical reconfiguration*, can be easily applied within the engineering cycle for DSE. But the ACS customer has to be involved into the execution of a physical reconfiguration since up to now it is not possible to provide basic reconfiguration services for hardware (this possibility may arise in highly flexible and self-adapting ACSs in the future).

Figure 16 depicts the enhanced engineering cycle for DSE of hardware and software within an ACS. The main elements remain identically to the description given in Section 4.1 (Figure 16 only includes the main elements). In the lower part of Figure 16 an example for the removal of a device is given from the system's perspective. This example will be used for explaining of the necessary tasks in order to provide also physical reconfiguration within the engineering cycle:

- ***Acquire existing application:*** The first step remains similar as it already includes the acquisition of the existing software as well as the hardware configuration. For the example given in Figure 16 only one device 'Dev1' exists which includes one control application 'Application'.
- ***Application modeling:*** The application modeling already includes one task which aims at the configuration of the hardware. Herein it is necessary to include new hard-

ware devices into the overall hardware configuration. Depending on the possibilities of the engineering tool and runtime environment it may be sufficient to add the physical device into the ACS and it will be recognized automatically by using plug-and-play mechanisms. In any case it is necessary to add the new hardware before the execution of the DSE.

- ***Evolution engineering:*** The task of modeling the ECA in order to achieve a smooth transition from the current system state to the new system state remains the same since there cannot be assumed any automatisms for changes of the hardware configuration. In the example depicted in Figure 16 the mapping of 'Application' is changed from 'Dev1' to the new device 'Dev2'.
- ***Execution of downtimeless system evolution:*** Also for the execution of DSE there are no changes necessary to the description given above. The only prerequisite is that any new hardware that has been specified needs to be available within the ACS. In the given example 'Application' moves to 'Dev2' and 'Dev1' remains unused.
- ***Removal of unused hardware:*** This task is necessary only for a physical reconfiguration. As due to the system evolution step some hardware may not be necessary within the ACS any more, these devices can now be removed.

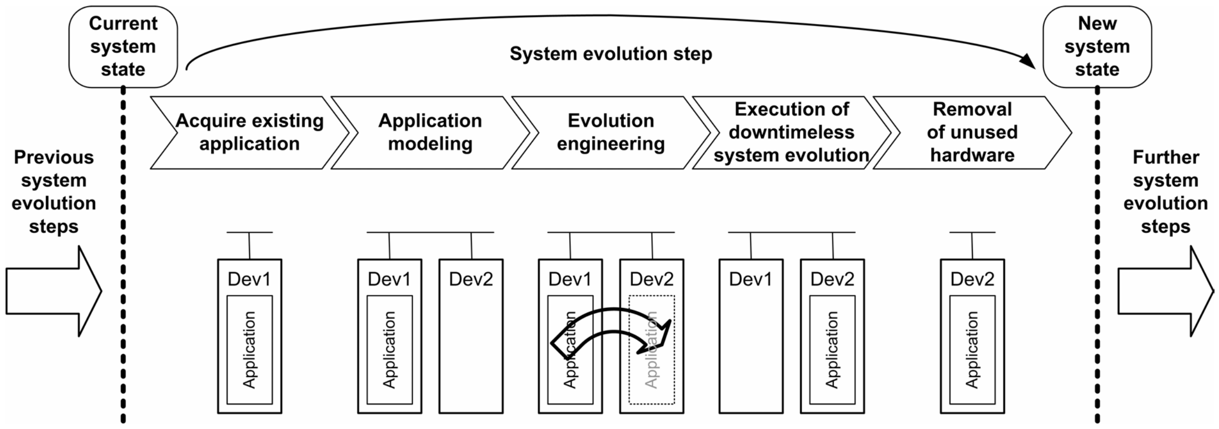

Current system state
System evolution step
New system state
Previous system evolution steps
Acquire existing application
Application modeling
Evolution engineering
Execution of downtimeless system evolution
Removal of unused hardware
Further system evolution steps
Dev1
Dev2
Application

Figure 16: Downtimeless system evolution with physical reconfiguration

The description of the engineering cycle for physical reconfiguration incorporates both possible scenarios, the addition and the removal of hardware. In both cases DSE needs manual support from the ACS customer in terms of adaptations to the hardware configuration of the ACS. But these manual actions can be integrated into the engineering cycle for DSE as unobstructed enhancement without changes to the above described methodology. Only in case of CECAs with physical reconfiguration within several EECFBs the execution of the DSE needs to be planned carefully in order to synchronize the manual actions with the CECA.

4.5 Summary

The modeling methodology for DSE represents the basic framework in order to enable ACS customers to describe changes within a system at run-time. The essential part within this engineering cycle is the evaluation of the DSE since any failures during the execution of a system evolution step may lead to a break-down of the overall ACS. Additionally the ACS customer is able to define appropriate failure handling mechanisms within the ECA. Physical reconfiguration is an unobstructed enhancement to the engineering cycle which focuses on changes to the control application mainly. In this case manual support by the ACS customer is necessary as there exist no means in order to automatically change hardware configurations.

In order to summarize the main ideas of this engineering methodology for downtimeless system evolution, we will consider the challenges for software evolution from Mens *et al.* (2005) presented in Section 3.4:

- ***Preserving and improving software quality:*** The clear structure for a single system evolution preserves high quality of the reconfiguration process itself and also of the overall system.
- ***Supporting model evolution:*** The methodology for modeling the DSE is based on a clear engineering cycle, therefore different versions and steps of the evolution process can be separated and planned in detail.
- ***Formal support for evolution:*** The evaluation of DSE is an integral part of the modeling approach. The details about the evaluation approach will be described in the following chapters.
- ***Evolution as a language construct:*** DSE is modeled with the elements of the IEC 61499 standard, enhanced by special FB types for dynamic reconfiguration (the basic reconfiguration services).
- ***Need for better versioning systems:*** The engineering cycle for DSE provides the means for the documentation of the different system states as well as the transition process in between. This can be used as basis for versioning systems.
- ***A theory of software evolution:*** The reference architecture for dynamic reconfiguration presented by Walsh *et al.* (2007b) has been presented as a basis with restrictions according to the special needs of DSE and the IEC 61499 standard.
- ***Post-deployment runtime environment:*** DSE is based on a runtime environment capable to change the control logic during operation. We will use the R^3E within this approach, which is an IEC 61499 compliant runtime environment with special adaptations to real-time execution and dynamic reconfiguration as presented in Zoitl (2007).

Chapter 5

New Concept for the Evaluation of Downtimeless System Evolution

The evaluation of DSE is depicted in the total engineering cycle as the fourth step within evolution control engineering (see Figure 10). But the evaluation is of outstanding importance for the application of DSE, because the main target next to the application of changes to the control application is the operation of the plant without disturbances. The concept for the evaluation needs to provide the necessary means for the proof of a system evolution step in such a way that it may be used also by ACS customers. The basis for this concept is represented by the structure of a system evolution step within the engineering methodology presented in the previous chapter.

The formulation of the concept for DSE will be split up into three items:

- First of all we will investigate the framework for the evaluation in an ACS, starting fromthe evaluation of control applications and as an additional task the evaluation of DSE.
- The formulation of the concept for the evaluation of DSE provides the main guideline within this work. The means will be different according to the necessary properties and different sequences within a system evolution step.
- A very important aspect for the evaluation process is the availability of a comprehensive description of the current system state. We will describe a possible scenario for the representation of this information that is based on the description of control devices.

5.1 Specification of the evaluation framework

The scope for the evaluation of control applications in general can vary in big extents. Bani Younis and Frey (2003) distinguish three different levels for the evaluation of control applications: some parts of the algorithm, the whole control application, or the whole system configuration. Based on the requirements that have been stated in Section 2.1, we need to take into account the overall control device and its configuration. This is stated especially by Requirement (3) “Underlying system configuration”.

In order to provide a structured analysis of the evaluation framework, we will start our considerations with ACSs operating only control applications and extend these considerations in a second step for DSE.

5.1.1 Evaluation of ACSs operating control applications

The classical situation of a control application that is used to control some kind of process or plant is depicted with respect to evaluation for instance in Hanisch (2004). The evaluation proves or falsifies if the system's behavior complies with the specifications of the desired or prohibited behavior. The control application as well as the process under control have to be considered in combination. Both have to be modeled in an appropriate formal description in order to provide the necessary system model for model checking. This way of system modeling is called closed-loop modeling. In the overall system, the control application acts on the measurement signals from the process under control and generates control signals that again influence the process. Therefore, a closed circuit of signals emerges that are exchanged between control application and process. Figure 17 depicts this classical situation of the evaluation framework for ACSs that operate control applications.

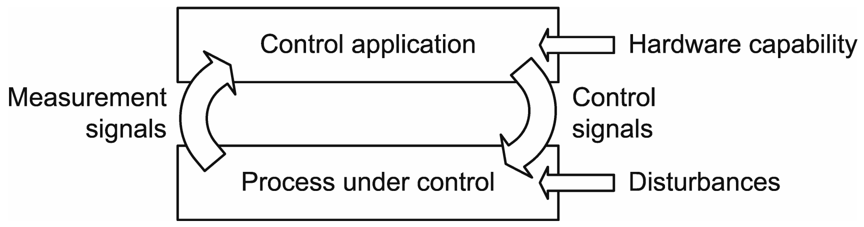


Figure 17: Classical situation for evaluation of control applications

The different elements involved in this closed circuit are characterized as follows:

- ***Process under control:*** In most cases no model of the process under control is necessary, when we consider the current practice of testing and simulation as main methods for the evaluation in ACSs. The ACS customer is not used to specify the process in advance and uses either the plant itself or a simulation model of some plant aspects for the development of the control application. Nevertheless, in specialized areas such as control theory high efforts are put into such a model in order to achieve highly efficient control strategies. But in general the design of a model of the process is an additional task that is necessary to permit also model checking for the evaluation. Hanisch (2004) considers this situation in more detail. As soon as any description of the process exists this can be used to generate the model in an appropriate input language for the model checking tool, as this is for instance described by Lobov *et al.* (2006a) for UML as the description of the plant and NCES for the input language for the model checking tool.
- ***Control application:*** The algorithms necessary to control the process are included in the control application, which is written in any kind of programming language as depicted in Section 3.2. Within this work we focus on the IEC 61499 standard. The FB networks can be used as input in order to generate the appropriate model of the control application in the input language of the model checker. Herein especially the Requirement (2), "Execution semantics", has to be treated carefully, as the implementation of the runtime environment used for the execution of the control application has significant influence on the behavior of the control application.
- ***Control signals:*** Control signals describe the interaction interface from the control application to the process under control. This interface is visible within the control application by means of SIFBs, that are the elements within IEC 61499 standard which are capable to integrate any kind of interaction with the environment.
- ***Measurement signals:*** These signals describe the counterpart of the overall interface to the process, the direction from the process to the control application. Again this interface is visible within the control application by means of SIFBs. As stated in Hanisch (2004) this interface usually does not provide access to all state variables within the process.

These four elements describe the models that need to be built on the information about the structure and internals of the ACS. Additionally, there are two aspects that have to be integrated into the evaluation process in order to evaluate the operational behavior of the ACS during the life cycle of the plant:

- ***Disturbances:*** An initial requirement of evaluation is to check whether the specification of a plant holds for all possible scenarios or not. The model of the process provides a variety of scenarios based on its modeled behavior. But additionally different disturbances may be modeled in order to describe for instance failures within the plant. These disturbances need to be modeled separately and are important for proving the behavior in unusual situations.
- ***Hardware capability:*** Another kind of disturbances belongs to the control application, in more detail to the underlying system configuration. In most cases a control device includes different software in order to provide the needed functionality of the ACS. A typical example is a web server, which does not influence the control application as it only reads the current status of the control device and offers this information within a web page. But it influences the execution behavior of the control application and may lead to violations of real-time constraints. Computational power of the control device needs to be considered, too. As the operation of control applications is constrained by time constraints, the speed of execution is an important source of disturbances within a given control device. Therefore these influences to the hardware capabilities need to be taken into consideration as additional disturbances within the evaluation process.

The above described elements of the framework for the classical evaluation situation have to be taken into consideration for the model of the system. Additionally, there are different categories of specifications that may occur, depending on the concrete system. Hanisch (2004, Section 4) mentions that at least three different groups of specifications exist:

- ***Plant specifications:*** "Plant specifications can often be formalized as forbidden state problems, but they might also specify forbidden sequences of states or state transitions."
- ***Process specifications:*** "Process specifications can be formalized as a set of partially ordered states or state transitions, sometimes even with time or hybrid dynamics."
- ***Product specifications:*** "Numerous product specifications cover an extremely wide range. Specifications of substances in process industries define chemical or physical properties of the products. (…) Product specifications in the manufacturing industry focus on geometrical or mechanical properties, color, surface properties etc."

5.1.2 Evaluation of ACSs incorporating downtimeless system evolution

When we consider ACSs that provide the possibility of DSE the framework for evaluation needs to be extended. All elements that have been mentioned above are valid also for the evaluation of DSE. But there are additional elements that need to be taken into account. Figure 18 provides a schematic of the overall framework for the evaluation of an ACS with DSE (additional elements are marked with gray color). Another closed circuit exists in between the control application and the ECA. The basic reconfiguration services used within the ECA are the appropriate means to influence the control application. On the other hand the current system state is the basis for the ECA in order to synchronize its actions with the control application. This second closed circuit is put on top of the above described closed-loop modeling of the process under control and the control application.

The additional elements within the framework for the evaluation of DSE are characterized as follows:

- ***Evolution control application:*** The ECA takes care of the execution of the DSE. Based on the EECFBs involved in the ECA different areas within the control application are related to the DSE. As the ECA can be modeled by the same means as the control application, namely as IEC 61499 application, the same prerequisites have to be taken into account for the ECA as for the control application (e.g., execution semantics). Further special requirements for the evaluation of the ECA are the incorporation of basic reconfiguration services which are represented as SIFBs.

- ***Basic reconfiguration services:*** The control flow from the ECA to the control application is represented by basic reconfiguration services. These incorporate management commands that are issued to the IEC 61499 device management in order to execute changes within the control application. Herein especially Requirement (4), "Modeling dynamic reconfiguration", has to be mentioned which demands the representation within the model of the ECA as well as within the model of the control application. The first aspect is simply defined by the interface of the SIFB and the sequence diagrams in order to describe the external interface behavior. The second task is much more complicated as formal models do not provide means for applying changes to the model during model checking. The approaches presented in Section 3.6.3 which investigate also dynamic reconfiguration provide possibilities to change the models based on given rules, but they do not incorporate changes to the model during evaluation. The modeling of basic reconfiguration services is one of the key tasks for the evaluation of DSE.
- ***Current system state:*** The ECA interacts with the control application by using event and data connections in order to get the necessary information about the current system state and coordinate its execution accordingly. The model of these event and data connections is the same as for the control application or the ECA. But additionally also basic reconfiguration services need to be used in order to achieve more detailed information on the current system state. As depicted in the example "closed-loop control circuit" in Section 4.2.3 for instance internal variables are necessary for certain transition management methods. Further simplifications of engineering such as a sniffer for events may be encapsulated as basic reconfiguration service and provides information of the current system state to the ECA. Accordingly, also basic reconfiguration services that provide information of the control application (instead of influencing its current state) need to be modeled for the evaluation of DSE.

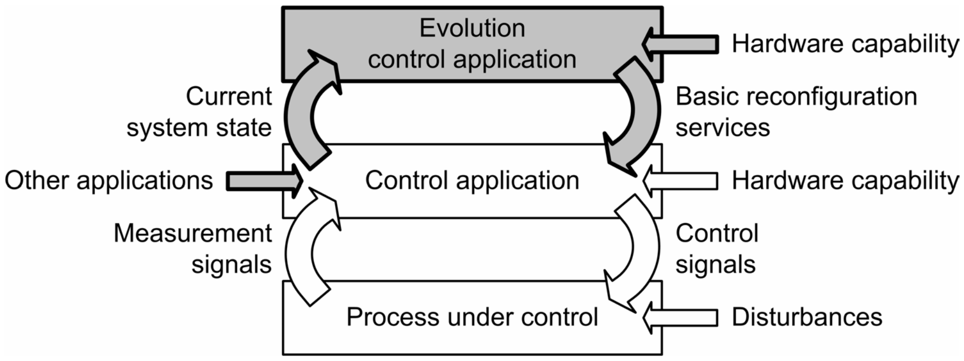


Figure 18: Framework for the evaluation of downtimeless system evolution

Also for the evaluation of DSE additional sources of disturbances exist next to those mentioned already in Section 5.1.1.

- ***Other applications:*** From the ECA's point of view only those parts of the control application that are included in the EROI are of special interest. In order to incorporate an appropriate behavior of the control application itself the areas of the EROIs need to be enlarged to the applications that incorporate these EROIs. But the overall control application, which usually consists of various IEC 61499 applications, is not completely part of the control application that needs to be modeled for the evaluation of DSE. From the point of view of DSE, a control application refers only to those

IEC 61499 applications that are affected by the EECFBs. All other IEC 61499 applications will be denoted as other applications. But as these other applications may provide some inputs also for the control application, the interaction between the control application and other applications can be taken into consideration as disturbances. Similar to disturbances to the process, other applications do not need to be modeled in detail. But their behavior and especially the interface behavior to the control application have to be incorporated into the evaluation process.

The influence of other applications is also important from the viewpoint of hardware capabilities mentioned for the control application. They have the same influence as the web server depicted above. Since they may need computational power for execution they also influence the execution of the DSE which e.g. may yield to violated real-time constraints.

- ***Hardware capability:*** On the level of ECA again hardware capabilities occur as disturbances to the evaluation process. But in this case they concern the actions that should be performed by the ECA, especially the basic reconfiguration services. Unlimited resources do not exist, e.g., memory, in order to execute any request from the ECA. These limitations of the hardware capability have to be considered apart from the general hardware capabilities described in Section 5.1.1.

The overall target of DSE is to change the current control application without causing disturbances to the process. The three different categories for specifications (plant, process, and product specification) need to be fulfilled for the system also during DSE (according to the reference model described in Section 3.4.1 these may belong to global and local consistency characteristics). Additionally, the ECA itself has to fulfill certain properties, which are summarized as fourth category of specifications:

- ***Evolution specification:*** *Evolution specifications* describe the properties of the ECA that additionally may be specified for the execution of DSE (e.g., preserving the consistent state of components that are exchanged).

5.2 Concept formulation

The evaluation of DSE has to prove whether the ECA violates any properties of the plant, process, product, or evolution specification. Therefore the execution of the ECA has to be taken into consideration. Figure 19 depicts the different phases for the execution of a single system evolution step. We will concentrate at the beginning on the basic evolution control engineering method, as within the enhanced methodology only the engineering process is simplified. The main characteristics of an ECA do not change and for the sake of concentration on the important aspects of the execution of an ECA we will neglect CECAs in a first step.

Five different phases can be distinguished during the execution of a single system evolution step, which will be discussed according to their impact on the execution of the control application:

- ***Download ECA:*** First of all it is necessary to download the evolution control application to the control device(s). This step is similar to the download of any application. From the control applications point of view it has no influence with the exception of one special case: The creation of connections between the control application and the ECA. When such a connection is an event connection, then events will be passed to the ECA and the ECA has to be operated already after this phase (before the system evolution step itself has been started).
- ***RINIT sequence:*** The initialization sequence includes actions for the preparation of the changes within the control application. As depicted in the example given in Sec-

tion 4.2.3 new FBs as well as their input connections are created and parameters of these FBs are written. As the input connections of the FBs may include also event connections the control application will trigger the execution of the new generated FBs which may influence the execution of the control application (in general by consumed execution time from the control device).

Both phases do not produce changes to the functionality of the control application. Influences happen based on side effects due to the interconnection of the ECA and the control application, but there are no active adaptations of the control application included concerning its behavior. These two phases provide the same characteristic and may be considered as preparation for the dynamic reconfiguration of the control application.

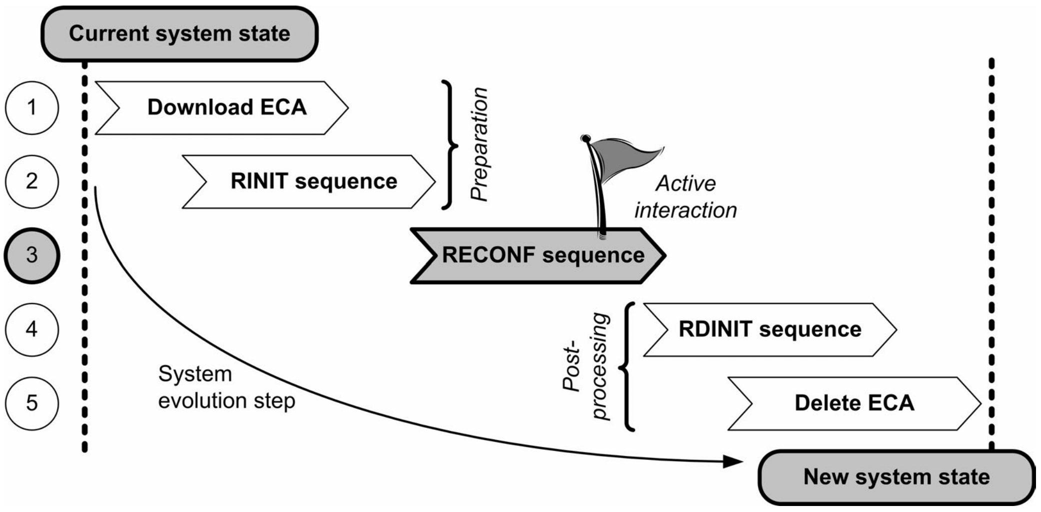


Figure 19: Execution phases of a system evolution step

- ***RECONF sequence:*** The reconfiguration sequence includes the active interaction of the ECA and the control application in order to change the control application. Any kind of adaptations to the control application may be applied based on the basic reconfiguration services and produce changes to the behavior of the control application.
- ***RDINIT sequence:*** The deinitilization sequence is responsible for cleaning up the control application, which aims at the deletion of FBs and connections from the old system state that are not used any longer in the new system state. The actions within the RDINIT sequence do not influence the behavior of the control application, but they provide changes in the execution behavior since the "old" FBs will be executed until they have been stopped or their input connections have been deleted. As depicted in the example given in Section 4.2.3 the parallel execution of the old and the new control application can be used in order to provide failure handling if the new control application produces errors.
- ***Delete ECA:*** The last phase in the execution of an ECA is the deletion of the ECA itself. The elements of the ECA themselves do not produce any disturbances to the behavior of the control application, but connections may exist between the ECA and the control application. As long as the ECA is available on the control device, the event connections will trigger the execution of FBs within the ECA and influence the execution behavior of the control application.

The two last phases within the execution of a system evolution step again handle very similar actions and can be considered as post-processing. The unnecessary parts within the control device (from the old control application as well as the ECA that has been executed) are deleted and the system is left in a clean new system state.

Based on this discussion the related approach for dynamic reconfiguration in ACSs presented in Zoitl (2007) can be classified. Zoitl (2007) describes the basic features real-time execution and reconfigurability of an IEC 61499 runtime environment (see Appendix B). The execution of dynamic reconfiguration is proposed by using an application that utilizes basic reconfiguration services. Furthermore Zoitl (2007, Section 3.5) identifies three different phases for the execution of a so-called reconfiguration application (similar to the ECA): the setup phase (for the preparation of the control application for reconfiguration), the execution phase (for the switch to the new application parts), and the shut-down phase (for cleaning up the remaining part of the original application). These sequences are compatible with the preparation of the system evolution step (download of ECA and RINIT sequence), the application of the system evolution step (RECONF sequence), and the post-processing of the system evolution step (RDINIT sequence and deletion of ECA). The main difference in Zoitl (2007) is that only the execution phase (in our terminology the RECONF sequence) should be represented within the reconfiguration application on the control device. The other two phases are executed by using the management application and the engineering tool.

Based on the engineering methodology for the modeling of DSE the approach presented by Zoitl (2007) can be included, too. The only difference is the design of an ECA that includes also the RINIT and RDINIT sequence within the control device. The additional possibility of modeling these two sequences as an application may not be necessary. In general the included actions will be executed almost sequentially. And by using the management application as a remote interface to the engineering tool exactly the same functionality is offered. The differences are on the one hand additional time consumption due to the communication between engineering tool and control device and on the other hand additional storage usage due to the representation of the RINIT and RDINIT sequence as IEC 61499 applications. When we consider also CECAs the situation is a little bit different, since the coordination of different system evolution steps needs additional means which are provided as IEC 61499 applications within this approach. Synchronization or influences between the different EECFBs need to be handled appropriately, which will be complicated if the engineering tool has to take care for the different RINIT and RDINIT sequences.

5.2.1 System integrity characteristics

The reference architecture for dynamic reconfiguration described in Section 3.4.1 includes a hierarchy of change types and their dependency as well as the different types of integrity management during the application of these changes. These integrity characteristics have to be considered for DSE in an appropriate way, as the concentration on ACSs and especially the IEC 61499 standard as programming language need tailoring of the general architecture. The change types that fit to the models of the IEC 61499 standard and which are supported by using basic reconfiguration services are protocol change, topology change, architectural change, and to some extent internal change (see Section 4.2.2).

We will consider the different characteristics and discuss their applicability for DSE. As changes to the internals of a software component (we have decided to consider an FB as a software component in Section 4.2.2) are only possible by changing the value of internal variables and ECC states, only three categories of system integrity remain based on the investigation of Walsh *et al.* (2007b):

- ***Global consistency:*** In terms of DSE global consistency aims at the preservation of the specifications of the control application and the process under control. These specifications are split up into plant, process, and product specifications in Section 5.1.1.

- ***Local consistency:*** Global integrity characteristics may be split up into local aspects that are mentioned within local consistency. Herein special issues for the different specifications of plant, process, or product are included.
- ***Active references:*** System integrity with respect to active references targets especially at SIFBs. A SIFB may encapsulate any kind of service which may include also dependencies to other SIFBs (for instance SIFBs for communication purposes). If the changes within the control application influence such an active reference to an underlying service, the behavior of the control application may produce failures. In detail this integrity characteristic has to be split up into two aspects: On the one hand such a dependency may be violated in the new system state and therefore has to be detected during the evaluation of the new application (see Section 4.1). On the other hand the dependency may be violated during the system evolution step temporarily (e.g. due to a disorder of basic reconfiguration services) which has to be proved by the evaluation of DSE.

Next to the given system characteristics there are further aspects that include especially evolution specifications. One of the above mentioned aspects, the active references consistency, already represents such an evolution specification for the dependencies that have to be kept in mind during a system evolution step. Further elements of the evolution specification can be derived also from the reference architecture in Section 4.2.2 as well as the from basic work Kramer and Magee (1985):

- ***State management:*** Although no basic reconfiguration service exists in order to exchange an FB instance it is possible to model such an exchange by a sequence of commands. The example given in Section 4.2.3 especially depicts this situation with an appropriate transition management for a controller. For this special case of FB exchange but also for the substitution of whole FB networks the requirement of state management has to be proved as property within the evolution specifications.
- ***Dependent operation:*** In contrast to the definition used in the reference architecture for dynamic reconfiguration, dependent operations may be recognized within the ECA. The data flow interrelation is based on the parameters of the basic reconfiguration services and hence on their effects to the control application. As already depicted for active references consistency above, a disorder within the execution of basic reconfiguration services of a system evolution step may not only produce failures within underlying services of SIFBs. For instance, a very simple dependency occurs if a connection should be established to a new FB instance which is created later during the execution of the ECA. Dependent operation consistency is an important property within the evolution specifications.
- ***Real-time constrained operation:*** A very important requirement within DSE is real-time constrained execution, which has its origin in the process under control and has to be fulfilled by the control application. As the ECA reconfigures the control application the changes to the control application are liable to appropriate real-time constraints. For the example of controller exchange mentioned in Section 4.2.3 the reconfiguration sequence starts as soon as one control cycle has been finished. But it will be executed successfully only when the reconfiguration sequence has been finished before the next control cycle is triggered (in detail as soon as the controller is triggered for execution). The time slot between two execution cycles of the closed-loop control circuit constrains the execution of the ECA and represents another property within the evolution specification.

  Real-time constrained operation is also included in the global and local consistency characteristics with respect to the ECA. Any execution phase of a system evolution step influences the execution of a control application since the same computational resources are used. Therefore, real-time constrained operation is part of these integrity characteristics, which have to be fulfilled although DSE is applied to the ACS. The real-time constraints should be modeled by the ACS customer during the engineering of the control application as well as the ECA. But also if the runtime environment is configured correctly in order to fulfill these constraints properly the influence of disturbances (see the discussion above) has to be evaluated.

- ***Requirements of resources:*** Already Kramer and Magee (1985) mentioned that requirements of resources are a desirable property for dynamic configuration (in this work we use dynamic reconfiguration instead of dynamic configuration). In detail they claim that it is necessary that the control devices provide enough free storage for the changes that should be applied. For DSE two different aspects have to be considered: the storage necessary for the ECA as well as the storage for the changes that are applied to the control application by the ECA. In both cases the system evolution step will only be satisfactory if enough free storage is available. If we consider also the computational power of the control device as a resource, the above mentioned real-time constrained operation characteristics also concerning the requirements of resources. Two similar applications with similarly applied DSE on different hardware platforms (with different computational power) may result in a successful execution of the system evolution step in the case of enough computational power and failure in the other case. Additional requirements of resources may emerge based on the type library of the control device. As a new FB instance can only be created if the proposed FB type is available in the type library of the control device, it is necessary to check the type library in regard with the actions within the ECA.

5.2.2 Evaluation means for a system evolution step

This work has identified evaluation of DSE based on model checking as basic methodology already in the introduction and especially in Section 1.1. But on the other hand an important aspect is the application field of ACSs, which especially has been stated in Requirement (6) to (8), namely “Extensive engineering support”, “Provision of formal models”, and “User-friendly definition of specifications”. The discussion above provides the different kinds of properties that are included especially in the evolution specifications. The different properties refer to very different questions concerning the execution of a system evolution step and may lead to even more simple evaluation methods than model checking in certain cases.

We will investigate each of the five execution phases with respect to the system integrity characteristics and their tasks during a system evolution step in order to identify the most appropriate evaluation means.

Download ECA

The download of the IEC 61499 application within the ECA is a time uncritical action with respect to the control application. The execution of the involved basic reconfiguration services needs not to be constrained by timing bounds. Based on an appropriate scheduling of the control application within the runtime environment of the control device no reason exists for a detailed analysis of this first phase within the execution of a system evolution step (we will take into consideration the R^3E in Chapters 6 and 7). But based on the connections between the control application and the ECA the execution behavior is influenced. This may be investigated by using model checking and appropriate specifications. On the other hand the influence on the control application is clearly arranged due to a very limited number of such connections and the clearly described influence within the ECA. Based on a comprehensive description of the current system state a rather simple valuation of the execution time necessary for the execution of the ECA in this phase is possible and the successful execution of the control application can be checked.

The different system integrity characteristics for the download of the ECA can be summarized as follows:

- ***Global consistency:*** The ECA does not influence the behavior of the control application. Only the effect based on the execution of new FBs due to connections to the ECA within the context of the control application has to be checked. This can be done by calculating the impact on the execution time of the control application.
- ***Local consistency:*** Similar to global consistency.
- ***Active references***: This property is not influenced by the download of the ECA.
- ***State management:*** This property is not influenced by the download of the ECA.
- ***Dependent operation:*** The download of any application has to follow appropriate rules in order to do not violate the dependent operation property. These rules have to be followed also for the download of the ECA and do not need an additional verification.
- ***Real-time constrained operation:*** No real-time constraints exist due to the nature of this execution phase for the download of the ECA. Therefore, the runtime environment may execute the necessary management commands for the download of the ECA during spare time. When we assume appropriate concepts within the runtime environment, there is no need to investigate on this property.
- ***Requirements of resources:*** As the resource computational power is already included within the evaluation of global and local consistency, only storage and type libraries as well as their requirements of resources during the download of the ECA have to be verified. As appropriate means the calculation of the interaction of the current system state and the memory management policy of the control device is sufficient in order to check this property.

The different system integrity characteristics for the download of the ECA can be checked for correctness without using model checking algorithms. Nevertheless, detailed knowledge about the current system state and the internal policies of the control device and the runtime environment can be used to provide an evaluation of the different evolution specifications.

RINIT sequence:

The initialization sequence provides the necessary preparation actions within the control application for the dynamic reconfiguration of the control application. Based on the approach presented in Chapter 4 this part of the ECA will be executed by the control device and appropriate computational resources are necessary. But the initialization sequence is also time uncritical and the influence to the control application is limited based on the scheduling policy of the IEC 61499 runtime environment. The establishment of the new FBs and connections for the new system state provide also an additional influence on the control application, as these FBs may be executed within the context of the current control application. The example given in Section 4.2.3 describes the final situation after executing the RINIT sequence as parallel execution of both the old and the new controller. But the new controller does not influence the behavior of the control application, which is a general prerequisite to the actions within the initialization sequence. Again only the execution time necessary for the new FBs has to be evaluated. Together with the information about the current system state the evaluation of the influences of actions within the RINIT sequence to the temporal behavior of the control application can be examined.

- ***Global consistency:*** The execution of the RINIT sequence does not influence the behavior of the control application. The additional FBs and connections which are added to the control application can be evaluated according to their necessary execution time.
- ***Local consistency:*** Similar to global consistency.

- ***Active references***: This property is not influenced by the RINIT sequence.
- ***State management:*** This property is not influenced by the RINIT sequence.
- ***Dependent operation:*** The correctness of the different basic reconfiguration services within the execution of the RINIT sequence can hardly be proved by model checking as no means exists for the dynamic changes to the model. But based on the representation of the current system state (especially including the changes applied during the system evolution step) the dependencies between the different operations can be checked rather easily.
- ***Real-time constrained operation:*** According to the scheduling policy of the runtime environment the execution of this sequence will be constrained in an appropriate manner. The prerequisites for these influences need to be evaluated according to the underlying scheduling theory.
- ***Requirements of resources:*** The resources memory and type library within the control device need to be taken into consideration based on the actions within the RINIT sequence and the current system state.

As the characteristics of the RINIT sequence and the download of the ECA are very similar (preparation of the system evolution step) also the used evaluation means are similar or even the same. Again it is not necessary to apply model checking for the verification of this execution phase. The evaluation process uses the information of the current system state and the actions within the initialization sequence for calculations in order to prove the different system integrity characteristics.

RECONF sequence:

The reconfiguration sequence represents the most important phase during the execution of a system evolution step. The control application is actively adapted to the new system state, which implies the time critical execution of the basic reconfiguration services and calculations included. This situation needs a very careful investigation on the influences between the control application and the ECA, which will be applied by the model checking technique. As already stated above no appropriate means exists for the dynamic adaptation of the system model for model checking. But at this point the preparation of the system evolution step has been finished and at least a static configuration of FBs can be considered. The changes to the control application are restricted to changes of connections and parameters (especially internal variables). The example in Section 4.2.3 for the exchange of the controller of a closed-loop control circuit includes the reading and writing of internal variables as well as the deletion and creation of connections. The scope of possible actions for dynamic reconfiguration is very limited for the consideration of the RECONF sequence and will be integrated into the model of the system. Additionally also properties exist that may be evaluated by using calculations within the information about the current system state.

- ***Global consistency:*** The properties within the plant, process and product specifications of the control application have to be considered based on the adaptations to the control application during the reconfiguration sequence. An appropriate model of all elements within the evaluation framework has to be established in order to apply model checking for proving these specifications.
- ***Local consistency:*** Similar to global consistency.
- ***Active references***: The interrelation of different control application parts due to underlying services encapsulated in SIFBs is considered within the active references property. Especially the temporal interruption of references needs to be considered during the execution of the reconfiguration sequence. The proof of this property within the evolution specification requires a detailed description of the mentioned underlying

services in order to provide the necessary information for the model checking procedure.

- ***State management:*** State management has to be modeled within the reconfiguration sequence of the ECA. The behavior of the control application (global and local consistency) is directly influenced by the used transition management method, but even if the transition management fails the disturbances to the process may be tolerated by the plant, process, and product specifications. As the target of DSE is the reduction of such disturbances, state management is added as a special property of the evolution specification and has to be checked by model checking.
- ***Dependent operation:*** For the reconfiguration sequence we have to design an appropriate system model that includes also the dynamic reconfiguration of the control application. This could be used also for checking the dependencies between the actions within the RECONF sequence, too. But on the other hand the incorporation of dynamic adaptations to the model will be based on a correct order of basic reconfiguration services as the appropriate means for modeling changes are not intrinsic functionalities of the modeling language. Similar to the RINIT sequence the calculation of the different basic reconfiguration services based on the current system state will be utilized for the reconfiguration sequence, too. This evaluation of the correct order of basic reconfiguration services provides a good basis for the generation of the system model used for model checking of the different other system integrity properties.
- ***Real-time constrained operation:*** As the reconfiguration sequence is deeply involved with the execution of the control application and the time critical aspect of this execution phase it is necessary to include this aspect in the model checking procedure. The temporal properties are as important as the functional properties within the evolution specification.
- ***Requirements of resources:*** Although the reconfiguration sequence concentrates on the adaptations of connections and parameters/variables also such kind of basic reconfiguration services may influence the memory of the control device. Similar calculations as already depicted above for the RINIT sequence and download of ECAs can be applied in order to evaluate the satisfaction of the requirements of resources of the RECONF sequence.

The reconfiguration sequence is the most critical part within the execution of a system evolution step and needs to be considered in all details by model checking techniques. Additionally also calculations based on the current system state are applied for certain properties, which will help to simplify especially the modeling of basic reconfiguration services.

RDINIT sequence:

The deinitialization sequence starts the post processing of the system evolution step and focuses especially on the deletion of the old parts within the control application. Based on the general idea of the engineering methodology for DSE this phase will only remove elements of the unused control application and should apply no influences in terms of additional execution time for new FBs to the control applications. In contrast the old application parts which are still part of the control application and consume execution time in the context of the control application will be removed and the total necessary execution burden for the control application will be decreased. As the deinitialization sequence does not influence the behavior of the control application itself and its execution is time uncritical, the influence to the control application can be neglected based on an appropriate scheduling policy of the runtime environment.

- ***Global consistency:*** No additional influence exists to the control application during the execution of the RDINIT sequence. Any connection between the ECA and the control application, which may exist from previous execution phases, has been already investigated during the corresponding execution phases.
- ***Local consistency:*** Similar to global consistency.
- ***Active references***: This property is not influenced by the RDINIT sequence.
- ***State management:*** This property is not influenced by the RDINIT sequence.
- ***Dependent operation:*** Similar to the creation of FBs and connections also the deletion of these elements may be executed in disorder. The consequences of such a disorder vary according to the concrete implementation of the IEC 61499 runtime environment and consequently should be avoided, too. The correct execution of the basic reconfiguration services can be evaluated based on the representation of the current system state during the sequence of commands.
- ***Real-time constrained operation:*** The scheduling policy of the runtime environment provides the basic framework for the evaluation of the real-time constrained operation of the RDINIT sequence. The RDINIT sequence has to be scheduled appropriately in order to do not disturb the execution of control applications.
- ***Requirements of resources:*** The RDINIT sequence will free allocated memory according to the deletion of unused FBs and connections. This property needs not to be checked in detail. A similar calculation as provided above can be used to update the current system state with respect to the requirements of resources.

The deinitialization sequence includes also no verification by model checking. Many properties do not need to be checked at all. The remaining properties can be evaluated based on the current system state.

Delete ECA:

The last phase within the execution of the ECA concerns to the deletion of the ECA itself by using the management application under control of the engineering tool. This sequence is of course time uncritical and will not influence the behavior of the control application. The execution of the different management commands has to be handled by the runtime environment similar to the download of any IEC 61499 application in order to delete the ECA without disturbances of the execution of control applications.

- ***Global consistency:*** This property is not influenced by the deletion of the ECA. The scheduling policy of the runtime environment has to handle any request by the management application without disturbances to the control application's execution.
- ***Local consistency:*** Similar to global consistency.
- ***Active references***: This property is not influenced by the deletion of the ECA.
- ***State management:*** This property is not influenced by the deletion of the ECA.
- ***Dependent operation:*** The deletion of any application has to follow appropriate rules in order to leave the control device in a clean system state. Based on the implementation of the runtime environment it may lead to an erroneous situation if for instance an FB is deleted although connections exist from or to this FB. The engineering tool has to implement appropriate rules, which do not need an additional check by some evaluation means.
- ***Real-time constrained operation:*** Similar to the download of the ECA also the deletion of the ECA does not have any real-time constraints. The runtime environment will execute the necessary management commands during spare time. An additional check for this property is not necessary.

- ***Requirements of resources:*** The deletion of the ECA does not require additional memory but relieves the amount of used memory. Accordingly this property does not need to be checked separately. A similar calculation as already mentioned during the previous execution phases may be used in order to provide the current system state after the deletion of the ECA.

The deletion of the ECA does not need any evaluation means to prove its correctness, when we assume an appropriate scheduling policy for the management application within the runtime environment and a structured sequence of management commands generated by the engineering tool.

Overview of evaluation means

The above discussion of appropriate evaluation means in order to prove the different system integrity characteristics for a system evolution step are summarized in Table 2. The different evaluation means have been simplified according to the following conventions:

- 'Verify' means prove of the property by using model checking.
- 'Check' means the evaluation of the property based on calculation especially by using the current system state.
- 'Engineering tool' refers to the necessity of appropriate rules within the engineering tool.
- Any prerequisites according to the scheduling policy of the IEC 61499 runtime environment have not been mentioned.

	Download ECA	RINIT	RECONF	RDINIT	Delete ECA
Global consistency	Check	Check	Verify	—	—
Local consistency	Check	Check	Verify	—	—
Active references	—	—	Verify	—	—
State management	—	—	Verify	—	—
Dependent operation	Engineering tool	Check	Check	Check	Engineering tool
Real-time constrained operation	—	—	Verify	—	—
Requirements of resources	Check	Check	Check	—	—

Table 2: Evaluation means for the proof of system integrity characteristics

This overview provides a clear classification of the different evaluation means necessary for the evaluation of the five execution phases of a system evolution step. Only for the reconfiguration sequence verification by model checking is applied. All other sequences can be handled by using appropriate calculations in order to evaluate the effect of the ECA on the control application. This also improves the usability of this method for ACS customers as the complex method of model checking is concentrated only on one sequence and also its scope is very limited.

The detailed consideration of the different calculations based on the current system state will be discussed in Chapter 6. The verification of the RECONF sequence by model checking is depicted in Chapter 7. The following section will investigate the necessary enhancements for CECAs as well as the representation of the current system state as it provides the necessary information in order to apply the different calculations and model checking.

5.2.3 Evaluation of CECAs

The enhanced evolution control engineering described in Section 4.3 is based on the encapsulation of the three sequences of an ECA (initialization, reconfiguration, and deinitialization) within EECFBs which may be modeled in order to coordinate and synchronize the execution of different system evolution steps within a CECA. Herein the reconfiguration sequence is of special interest, as it describes the coordination of changes applied to the control applications.

When we consider the evaluation means necessary for CECAs we can apply the previous system integrity characteristics to each single system evolution step within the CECA. Based on the five execution sequences of a system evolution step the situation can be described for CECAs as follows:

- ***Download CECA:*** The situation can be handled similar as depicted for an ECA. Only the overall CECA has to be taken into consideration and respectively each EECFB and its interactions with the control application has to be checked.
- ***RINIT sequences:*** Each initialization sequence within the EECFBs of a CECA can be treated separately in the same manner as for a single evolution step.
- ***RECONF sequences:*** The reconfiguration sequences of the different EECFBs have to be analyzed according to their interrelations as already depicted in Figure 15a in the overview schematic of a CECA. For the execution of CECAs this is the most important aspect and has to be treated appropriately for the evaluation of the reconfiguration sequence. If for instance the RECONF sequences of two EECFBs are started synchronously (as this is depicted in Figure 15 for 'EECFB2' and 'EECFB3') the evaluation process has to incorporate both sequences for the proof of the different integrity characteristics. If no interrelation exists between the different EECFBs (as for instance in the case of 'EECFB1' in Figure 15) the reconfiguration sequences of such EECFBs can be verified independently.
- ***RDINIT sequences:*** Each deinitialization sequence within the EECFBs of a CECA can be treated separately in the same manner as for a single evolution step.
- ***Delete CECA:*** The situation can be handled similar as depicted for an ECA. The overall CECA has to be taken into consideration instead of a single evolution step.

The main prerequisite for the evaluation of a CECA is that even if the different sequences may be considered independently at least the current system state incorporates the whole CECA (e.g. for the consideration of requirements of resources).

5.3 The current system state: KAPPA vector

The discussion about the evaluation of system integrity characteristics as properties of the evolution specifications above mentioned many times the link to the current system state in order to provide the necessary information for the different evaluation means. The different kinds of calculations and verification by model checking make high demands for the description of the current system state. Therefore this element becomes an important aspect within the concept for the evaluation of DSE.

In order to provide a comprehensive description of the current system state we will use the term *KAPPA vector* as synonym for the different kinds of information related to the current system state. In detail all information concerning the different applications, the system configuration as well as the interrelation between different control devices has to be included into the KAPPA vector. The calculations based on the KAPPA vector mentioned above for the evaluation of different system integrity properties will be summarized as *KAPPA calculus* (see Chapter 6).

The KAPPA vector characterizes the current system state. This means KAPPA is a structured list of parameters that describe the current system state. The classification of parameters of the

KAPPA vector can be provided by using different aspects. One aspect is the changeability of parameters according to the dependencies of applications and devices. This will be investigated in Section 5.3.1. Another aspect is the structure of the automation and control system and will be discussed in Section 5.3.2. Finally the temporal behavior of the KAPPA vector during the execution of a system evolution step will be depicted in Section 5.3.3.

5.3.1 Characterization of KAPPA vector elements

The parameters within the KAPPA vector (respectively the KAPPA vector elements) aim at the description of the hardware and the software within an ACS. These elements may be characterized according to the dependency of what may be changed during the process of DSE. This belongs on the one hand to the applications (as primary target of a system evolution step) and to the devices on the other hand.

- ***Applications:*** This category includes the software part within the ACS that may be changed during the DSE. It includes the control applications as well as the ECAs.
- ***Devices:*** This category includes the hardware in terms of control devices within the ACS as well as those software parts that will not be changed within a DSE. Herein especially the operating system and the runtime environment have to be mentioned as parts of the device. This is based on the fact that these elements of the software within a control device will only change in case of a physical reconfiguration (addition or removal of a control device).

According to these categories four combinations of parameters would be possible, whether the parameters are changeable or not concerning the application or the device. As the application will not be of interest in case of independent parameters from the device, only three different kinds of parameters can be distinguished.

Device dependent & application independent parameters

Within a heterogeneous system the fundamental parameters of a control device like available memory, processing capability, input/output interfaces or supported network communications have to be mentioned within this group. Further parameters within this group concern the features of the system configuration within the software part of the control device (e.g., operating system, runtime environment). Examples for these parameters may be the set of supported commands, the functionality of the runtime environment, processing time for atomic control operations, capability of the scheduling algorithm, etc. But also those applications that will not be changed (e.g., the web server mentioned in previous discussions) have to be incorporated.

Device dependent & application dependent parameters

Due to the relation of applications to control devices, any parameter dependent on the application is also a parameter dependent on the device. But within this group we especially focus on information about the currently available applications within the control device. This includes information about the currently available free memory space and processing power as well as the mapping of the control applications and ECAs and their actions (e.g., instantiation of function blocks). Within this group also the behavior of control applications has to be described, which is based on the one hand on the event connections within the control application and on the other hand on the behavior of the external triggers for execution (incorporated in SIFBs).

Device independent parameters

There may be also a third group, which is independent of both the device and the application. These parameters describe for instance the position of a device within the network. This information is necessary if network communications will be used within DSE. For instance, the number of switches within the communication of two devices has a major impact on the latency time of an Ethernet network.

5.3.2 FDCML as basis for a KAPPA vector representation

The concrete representations of the different parameters which are necessary for the evaluation of DSE are dependent on the implementation that provides the basis for the evaluation method. In order to provide an example for the representation of the KAPPA vector we will utilize the device description based on FDCML, which is depicted in more detail in Appendix A. As FDCML provides a framework for the description of any parameters for control devices it is capable to integrate the necessary parameters for the evaluation process. A brief overview on this kind of representation of a KAPPA vector is given also in Sünder et al. (2007c).

The general aim of the FDCML specification (FDCML.org, 2002) is to provide a meta language for the description of control devices. There is only a small amount of parameters (e.g., for device identification) that are defined within the specification. The majority of elements within the XML schema aim at a framework in order to define further parameters. Examples are for instance 'specificProperty' or 'additionalItem', as described in detail in Appendix A. Furthermore the topics 'DeviceFunction' and 'ApplicationProcess', which represent an important part of parameters especially concerning the group of parameters that are dependent on the device and the application are completely left open by the specification.

A first step for the adaptation of the FDCML specification for the use as representation of a KAPPA vector has to investigate the usage of an appropriate description of applications. Herein the data model of the IEC 61499 standard, especially the definition of the XML representation of its elements given in IEC 61499-2 (2005), provides a good starting point. Figure 20 depicts the general structure of the XML schema for FDCML (lower part) and the DTD of an IEC 61499 system (upper part). As the XML schema for FDCML is explained already in Section A.1, only the elements of the IEC 61499 system will be introduced roughly. The DTD represents the definitions of the system model which has been provided already in Section 3.2.2. A system consists of applications (depicted by the element 'Application'), devices (depicted by the element 'Device'), and the communication systems (depicted by the elements 'Segment' and 'Link'). The element 'Mapping' is responsible for the description of the interrelation between an FB instance in 'Application' and 'Device'. Additionally appropriate DTDs exist in order to describe libraries, especially 'LibraryElement' for FB types, adapter types, or subapplication types and 'DataType' for the declaration of user defined data types. In general the parameters included in these DTDs provide a mixture of structural information about the different devices and their connections via the communication networks and application information in terms of FB networks.

The bold arrows used in Figure 20 depict the strategy for incorporating the contents of the IEC 61499 models into the framework of the FDCML XML schema. FDCML provides a very detailed description of the structural elements of a control device, which is not part of the IEC 61499 standard and expands the relatively simple overview of a system configuration. On the other hand, the representation of applications is completely missing in FDCML. The following list provides an overview on the principle methodology of merging the two different representations on the basis of the FDCML XML schema.

- The most important enhancement of FDCML is the use of the element 'Device' as child element of 'ApplicationProcess'. As the FDCML representation is focused on a control device, the mapping as well as applications within the system will be represented by the appropriate application parts within the 'Device' elements. A control device may have several devices (there are different runtime environments possible within one control device), which may include several 'Resource' elements as well as control applications and ECAs in terms of 'FBNetwork'. The reason for the usage of 'FBNetwork' within 'Resource' and 'Device' comes from the definition of IEC 61499-1 (2005, Section 1.4.2) that a device which does not contain a resource should be functional equivalent to a resource.

- The type definitions included in 'LibraryElement' and 'DataType' describe the capabilities of the runtime environment within the control device. A runtime environment is a special kind of 'resourceEntity' following the definitions of FDCML. The 'resourceEntity' element has to be enhanced by these two kinds of elements in order to describe the type library within a runtime environment.

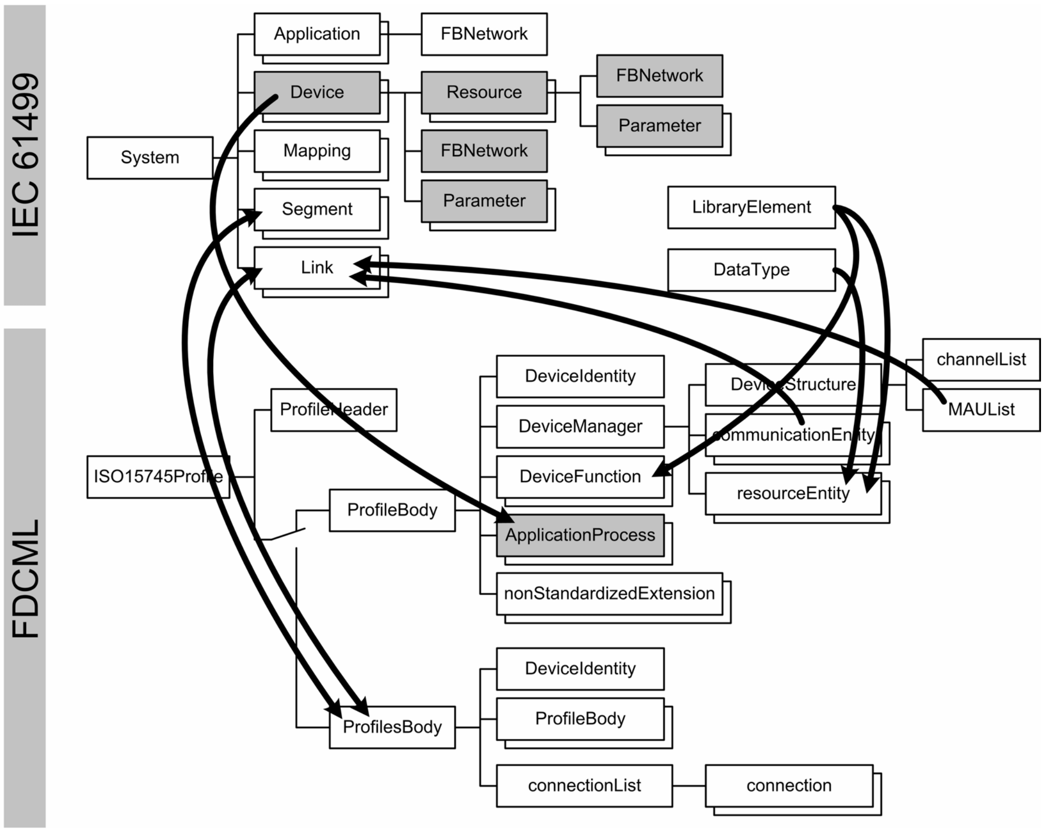


Figure 20: Incorporation of IEC 61499 into FDCML in order to represent the KAPPA vector

- Another aspect representing a completely open point within the FDCML is the element 'DeviceFunction'. Herein the "intrinsic function of a device in terms of its technology" (ISO 15745-1, 2003) should be included. The sequence diagrams within the declaration of SIFBs provide some aspects of 'DeviceFunction', as SIFBs are the interface to the intrinsic functions of the device within the runtime environment. Accordingly the DTD elements representing the declaration of SIFBs may be used also within 'DeviceFunction'.
- The representation of communication networks and the connections of devices via communication are described very roughly in IEC 61499. FDCML includes a more detailed description of the interfaces within a control device ('MAUList') as well as the communication aspects ('communicationEntity'). This information can be used to improve the brief description within IEC 61499. In terms of elements within FDCML the cooperation of different control devices is given within the element 'ProfilesBody'. In detail, a list of control devices ('ProfileBody') and their connections ('connectionList') are depicted based on the information about the internal structure of a control device.

The FDCML device description needs additional enhancements in order to provide a comprehensive representation of the KAPPA vector. One important aspect concerns to the element 'resourceEntity' within the element 'DeviceManager'. Up to now we have discussed the use of the description of library elements within the runtime environment. But there are different aspects that need to be incorporated within the 'resourceEntity' element, too:

- ✓ Computational unit and available memory storage
- Operating System: The description of the operating system may start with general parameters such as information about the scheduling of tasks (e.g., number of priorities) as well as further aspects such as the behavior of the operating system for administrative activities (e.g., amount of time and frequency for such activities). The temporal behavior is of special interest for the formal description and has to include time parameters for instance for a task switch.
- Runtime environment: Next to the library elements available within the runtime environment further parameters such as the memory consumption or the set of accepted management commands may be part of the description. With regard to the temporal behavior, the different timing parameters are of special interest for a detailed analysis of the execution behavior.
- Formal models: A very detailed description of a control device may also include the formal models of its elements such as operating system, runtime environment, or FB types. FDCML provides the possibility to include this kind of information by using external XML schemas, too.

5.3.3 KAPPA vector during execution of a system evolution step

The current system state which is represented by the KAPPA vector is a constant representation of parameters during the normal operation of the ACS. But especially for DSE the KAPPA vector is changing according to the applied changes to the ACS. This has to be pointed out as a very important aspect during the evaluation of a system evolution step. Figure 21 depicts the situation during the execution of a system evolution step.

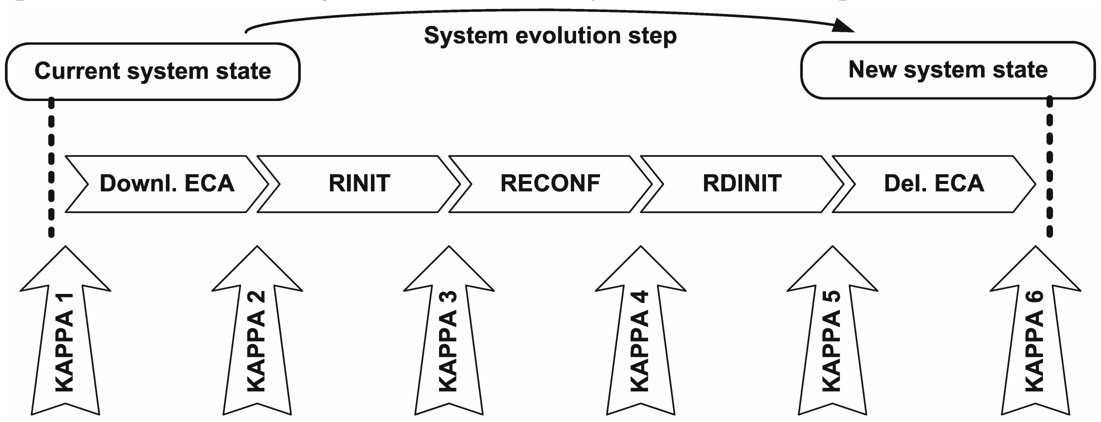


Figure 21: Execution of a system evolution step with regard to the KAPPA vector

We can distinguish the five different sequences that provide changes to the current system state: download of the ECA (denoted by 'Downl. ECA'), RINIT sequence, RECONF sequence, RDINIT sequence, and deletion of the ECA (denoted by 'Del. ECA'). In a very brief overview two different KAPPA vectors exist: 'KAPPA 1' for the current system state and 'KAPPA 6' for the new system state. These two KAPPA vectors represent the stable system configuration before and after the DSE. But after each of the different execution sequences within a system evolution step we can characterize the system state again by using KAPPA vectors. Figure 21 exactly describes this situation mentioning also 'KAPPA 2' up to 'KAPPA 5' in between the stable system states. And if we again have a more detailed look at the five execution sequences, we can split up the changes to the KAPPA vector into the smallest part of basic reconfiguration services which are applied to the system state. Therefore, the KAPPA vector provides only a snapshot of the system state. Especially for the calculations based on the KAPPA vector mentioned for the evaluation of system integrity properties it is important to provide the evaluations on the currently valid KAPPA vector. This situation includes high requirements to the engineering tool which incorporates DSE and especially its evaluation, since the overall process of the execution of a system evolution step has to be verified before its actual execution.

The situation can be adapted accordingly also to the combination of system evolution steps within a CECA. Special attention has to be paid to the situation of physical reconfiguration as described in Section 4.4. Herein the interaction of the ACS customer is mentioned in terms of changes to the hardware configuration also for DSE. This represents also a change to the KAPPA vector and has to be coordinated with the actions within the (C)ECA and accordingly also with its evaluation.

5.4 Summary

The concept for the evaluation of DSE has to take into consideration the different elements and especially disturbances within the evaluation framework as well as the properties mentioned as evolution specifications for the proof of a system evolution step.

The evaluation framework is characterized by two closed circuits. On the one hand the control application and the process under control are modeled as a closed circuit via the control and measurements signals. On the other hand, the ECA and the control application are modeled also in a closed circuit via the basic reconfiguration services and the current system state. Apart form the usual sources (disturbances to the process and hardware capabilities) also any other applications that request computational power as well as hardware capabilities for the execution of the ECA act as disturbances to the overall system.

The evolution specifications which have to be fulfilled by the ECA can be split up into different properties according to the system integrity characteristics with special tailoring to the field of ACSs and IEC 61499:

- ***Global and local consistency:*** Preserving the properties of the product, plant, or process during execution of DSE.
- ***Active references:*** Incorporation of dependencies based on underlying services which are utilized via SIFBs.
- ***State management:*** Inspecting the effects to the process under control and the efficiency of modeled transition management methods.
- ***Dependent operation:*** Checking the correct temporal order of basic reconfiguration services.
- ***Real-time constrained operation:*** Investigating the temporal behavior of operations is as important as the functional behavior for the correctness of DSE.
- ***Requirements of resources:*** Evaluation of basic properties such as sufficient resources in terms of memory and type libraries.

The analysis of the different properties of the evolution specification in terms of appropriate evaluation means results in the use of model checking only in the reconfiguration sequence of DSE. Many properties can be evaluated by calculations based on the current system state. Therefore, applicability for the ACS customer is increased as the complex method of model checking can be restricted to a small portion within the overall evaluation method.

The current system state, denoted as KAPPA vector, is of special interest for the evaluation method because it includes all necessary information for the evaluation based on calculations and model checking. In contrast to normal operation, where the KAPPA vector can be assumed as static, the current system state is changed due to the execution of each single basic reconfiguration service and therefore it is a highly versatile quantity (in detail a structured set of parameters) during the evaluation of DSE.

Chapter 6

Evaluation of properties by KAPPA-based calculations

This thesis proposes the use of verification by model checking in order to prove the correctness of DSE. Already in the introduction model checking has been identified as the best fitting means available for evaluation. The considerations about the different system integrity characteristics in the previous chapter give a detailed description of the necessity of model checking within the most critical phase of a system evolution step, the reconfiguration sequence. But the discussion also provides the important result that many properties can be simply evaluated by some calculations based on the current system state, the KAPPA vector. Especially the preparation of DSE, the download of ECA and the initialization sequence, as well as the post-processing, which includes the deinitialization sequence and the deletion of the ECA, can be sufficiently evaluated by KAPPA-based calculations.

The different properties of the evolution specification have to be treated by different types of calculations. The following three categories will be distinguished within this chapter:

- ***Influences to temporal control application properties:*** The global and local properties of control applications are influenced only in their temporal properties due to the additional execution of the system evolution step. Therefore it is not necessary to verify functional properties. But the disturbances to the execution of control applications have to be taken into consideration in order to prove whether the control logic will be executed in time or not.
- ***Check for dependent operation:*** The order of the basic reconfiguration services within all phases of a system evolution step is part of these calculations. Two different situations have to be distinguished. On the one hand the three phases of the execution of an ECA need to be considered for any disorder in basic reconfiguration services. On the other hand the download and the deletion of the ECA are executed by the engineering tool, whereas certain rules need to be followed in order to do not be in conflict with the dependent operation characteristic.
- ***Check for requirements of resources***: Two kinds of resources are of special interest within the process of DSE: the library elements of the runtime environment and the available memory.

6.1 Influences to temporal control application properties

The preparation of a system evolution step includes two phases: the download of the ECA and the initialization sequence. Both phases are characterized as not time critical because the control applications are not changed in their functional behavior. Nevertheless, the temporal behavior may be changed and therefore in this context the global and local properties of the control applications may be violated. The evaluation of the influence to these properties will be based on the KAPPA vector, which includes the characteristics for the execution of the control applications as well as the basic theory of scheduling within the runtime environment.

There are two different aspects that need to be checked:

- ***Control applications which are not part of the EROI:*** A control application which is not involved in the DSE is considered as disturbance to the execution of the system evolution step. But on the other hand, these applications may be influenced by the DSE, too. The reason is the same in both cases: each execution within the control device needs computational power, and if certain limits are exceeded the temporal behavior of applications will be violated.
- ***Control applications under control of the ECA:*** A control application which is influenced by the ECA will not be changed in its functional behavior, too. But during the download of the ECA also event and data connections between the control application and the ECA are created. These connections are necessary for the synchronization during the execution of the system evolution step. In addition to the above mentioned interrelation based on the computational power of the control device the control application will include further executed FBs based on the new connections. The effect to the temporal behavior of the control application has to be checked. The situation is even more complicated during the initialization sequence. Herein new FBs and connections are added to the control application itself (e.g., the new controller in the example given in Section 4.2.3). From the viewpoint of temporal behavior the influence to the control application is the same: additional FBs need to be executed in the context of the control application.

For a general consideration of an appropriate calculation for the estimation of the influence to the temporal behavior of a control application it may not be possible to evaluate this property in a satisfactory manner. The reason for this is the dependency of the control application behavior on the process under control, which may provide triggers for the execution at any time (if we include erroneous behavior, too). For the general case the use of verification by model checking (see Chapter 7) will be necessary in order to prove the global and local consistency of the control application during the download of the ECA and the initialization sequence. But under certain prerequisites, which depend also on the characteristics of the runtime environment, it is possible to use theoretical results of scheduling theories and approximations for an evaluation of the temporal behavior of the control applications.

We will investigate two different situations. First of all we will provide a short description of the evaluation for a cyclic execution of control applications. Secondly we will follow the investigation depicted in Zoitl (2007), which is the basis for the R^3E.

Consideration of cyclic execution

Cyclic execution of control applications is widely used in ACSs and also manifested in the principle architecture of IEC 61131-3 (2003). If we neglect the possibility to trigger an execution by an external event, the situation gets very simple concerning the evaluation of temporal behavior. The influences from outside are incorporated only within the cyclic execution. Based on an analysis of the current system state, it is easy to evaluate whether there will be an influence to the control applications or not.

In order to provide a more detailed example we can examine industrially used operating systems and their development tools, as for instance (Wind River, 2007). Herein the workbench for the real-time operating system VxWorks [62] is depicted, which includes so-called run-time analysis tools. These tools provide a visualization of the tasks of the operating system during operation of the system, which includes for instance the dynamic interaction of the target hardware, the operating system, and the different programs. Another aspect is especially important for an estimation of the influence to the temporal behavior of control applications: the so-called performance profiler. The system displays the execution time of each control application (minimal and maximal execution time). Based on such a detailed information about the current system state and the settings of priorities and cycle times for the control algorithms it is possible to estimate whether the preparation of a system evolution step influences the control applications or not.

The paradigm of cyclic execution can be utilized also for a performance analysis in ACSs which is based on IEC 61499. The IEC 61499 standard is purely event driven, but by the restriction of external events to cyclic occurrence the execution within the event-driven FB network will have cyclic behavior, too. This prerequisite has been used in Khalgui *et al.* (2004) for the development of a scheduling design that does not violate temporal properties of control applications. In more detail each input event of an FB is considered to occur periodic. The behavior of the resource and the FBs is given as state machine implemented in timed automata (see Section 3.6.3). Based on a characterization of the input event occurrences (period and jitter) and the execution time for algorithms the behavior of output events and the composition of FBs can be evaluated. Khalgui *et al.* (2004) use this information in order to construct a non-preemptive offline scheduling which avoids simultaneous event occurrences for a given FB. The evaluation of a given system state as well as the execution of the first two phases of a system evolution step can be done by using the schedulability conditions defined by Khalgui *et al.* (2004). The first schedulability condition examines if occurrences exist, and the second condition proves the assumption of periodic output events.

Consideration for pure event-based execution

The assumption of periodic input events of FBs based on the IEC 61499 standard is a very limiting prerequisite for the execution behavior. And it neglects one of the most important aspects of IEC 61499, the event-based execution. Within this thesis we especially focus on a concrete implementation of an IEC 61499 runtime environment, the R^{3}E. This runtime environment is based on the fundamental theory given in Zoitl (2007), which provides a concept for the real-time constrained execution of pure event-based control applications. Appendix B provides a more detailed description of the basic idea for the real-time execution of FB networks, which uses the event sources as the initial points of execution paths within the control application. But in contrast to Khalgui *et al.* (2004) there is no prerequisite of periodic occurrence for event sources.

We will give a short description of the scheduling theory developed in Zoitl (2007) in order to identify the important parameters which are necessary for the evaluation of real-time execution within the R^{3}E. Then we will depict the extraction of the necessary parameters based on the KAPPA vector. Finally, we will examine the situation of influences to the temporal control application behavior during the preparation of a system evolution step.

6.1.1 Scheduling theory of R^{3}E

Zoitl (2007) takes into consideration the combination of real-time execution and dynamic reconfiguration within an IEC 61499 runtime environment. This section gives a rough overview on the scheduling theory that is given in Zoitl (2007, Chapter 4), which defines the following requirements for the real-time execution model of IEC 61499:

- The number of tasks within the control device may change during operation.
- The execution time of tasks, which includes the execution of event chains, may change based on adaptations incorporated as basic reconfiguration services to the control application.
- Control applications and ECAs may be connected via data and event connections.

Occurrence of external events

The initial elements of execution are the event sources, which are triggered by external events. These provide the incentives for any execution within the control application. Accordingly real-time constraints are related to event sources. In order to provide parameters for the occurrence of external events, Zoitl (2007, Section 2.3.3) takes into consideration different process models which are known in literature:

- ***Periodic occurrence model:*** A very strict periodic model can be simply characterized by the cycle time T_P and does not take into consideration deviations in the occurrence of the external event.
- ***Periodic occurrence with jitter:*** Additionally the occurrence of a periodic event may vary by a small amount of time. The occurrence of the external event is mainly defined by the cycle time T_P, but a variation in the magnitude of two times J is incorporated, too.
- ***Irregular arrival patter:*** Herein a varying sequence of time intervals which is known in advance is characterized. T_S is the bounded sequence of time intervals. Furthermore this sequence will reoccur based on an overall period T_P.
- ***Bursty arrival pattern:*** This pattern characterizes a group of n events (burst) which may occur with a minimal inter-arrival time T_{min} of two consecutive events. A time interval T_P exists that describes the cycle time or a minimal inter-arrival time of the different bursts.
- ***Bounded model:*** The bounded model limits the occurrence of consecutive events by a lower bound, the minimal inter-arrival time T_{min}, and an upper bound, the maximal inter-arrival time T_{max}. The upper bound may be neglected.
- ***Bounded average rate model:*** This occurrence model is based on a statistical description of event occurrences by using an average minimal inter-arrival time T_{min} and a distribution function such as a Gaussian distribution with the standard deviation σ.
- ***Unbounded arrival pattern:*** The sequence of events is not known and cannot be used as external event for the real-time constrained execution.

These parameters can be used in two different manners. First of all they provide the basis for the calculation of scheduling criteria which provide a check whether real-time constraints can be met or not. On the other hand this information can be used also within the runtime environment in order to limit the invocation of executions based on external events. As the ACS customer knows in detail the assumed behavior of the process under control he can specify also the appropriate parameters for the occurrence of external events during application engineering. Based on Zoitl (2007, Section 4.5) this can be incorporated in the runtime environment as filters of unexpected external events, e.g., in case of erroneous behavior of the plant.

Execution time of event chains

Each execution of FBs within an event source and an event sink is called event chain. These event chains may be constrained by real-time parameters. If a real-time constraint is issued to an event chain, a separate task is established within the runtime environment. A resulting set of tasks can be derived from the current configuration of the control device. Unconstrained event chains are executed within so-called background tasks, which do not need to be considered for the evaluation of real-time constrained execution. The execution time of event chains is characterized by two parameters:

- ***Worst Case Execution Time (WCET):*** The longest time which is necessary for the execution of the event chain.

- ***Best Case Execution Time (BCET):*** The shortest time which is possible for the execution of the event chain.

In case of event chain execution the differences in the execution time come from the internals of the event chain, the FB network. An FB may issue an output event only if the control application is in a certain state, and consequently different execution times occur for different triggers of the event chain. Another possibility for changing execution times is the application of DSE to the control application, e.g., during the execution of the initialization sequence. We will consider the extraction of WCETs and BCETs for given control applications in the next section.

The set of active tasks

Zoitl (2007, Section 4.6.2) introduces the *Worst Case Active Task Set* (WCATS) as basis for schedulability rules in the following way. The WCATS describes "the set of active tasks which requests the highest demand of processor execution capacity of all possible active task sets". In case of pure cyclic execution this set can be identified rather easily based on the cycle times and their starting times, and appropriate evaluations about the real-time execution can be applied rather easily. For the derivation of the WCATS in case of mixed periodic and acyclic tasks Zoitl (2007) defines the following rules:

- Only tasks with real-time constraints need to be considered.
- Arrival times of all tasks are not synchronized or harmonic to each other.
- The bounded model, which provides an upper boundary on the occurrence frequency of a task, is assumed as general model for real-time constrained tasks.
- Periodic tasks are a special case within the bounded model where the minimal and maximal inter-arrival times are equal to the cycle time: $T_{min} = T_{max} = T_P$.
- For each task the relation $WCET_i \le D_i \le T_{\min,i}$ has to be fulfilled, where D_i is the deadline of the *i*-th task.

Following these rules the WCATS consists of all tasks that may be triggered within the control device and it is assumed that all tasks are activated at the same time. But as depicted in Appendix B event chains may be coupled to each other by special SIFBs. Based on the parameters of the different coupled event chains a more concrete WCATS can be derived. If for instance the sum of all deadlines D_i of the coupled event chains is smaller or equal to the minimal inter-arrival time of the initial external event, only one task of this event chain can be active at the same time. Accordingly the WCATS is not a single set of tasks but different WCATSs exist. In the example given above a separate WCATS has to be considered for each task within the coupled event chain.

Bounds for real-time constrained execution

In order to provide schedulability rules for a given WCATS Zoitl (2007) determines two different situations: static priority scheduling and dynamic priority scheduling. In both cases a general task set based on the bounded model is taken into consideration. If the different WCATSs of a given configuration of a control device fulfill the schedulability rules all real-time constraints within the control application will be met.

Static priority scheduling (Zoitl, 2007, Section 4.6.3): The basic prerequisite for the static priority scheduling is the use of a deadline monotonic priority assignment as it has been checked that this is optimal for static priority scheduling of aperiodic task sets. Deadline monotonic means that the task with the shortest deadline gets the highest priority. The basis for the schedulability boundary is the so-called synthetic utilization $U_{syn}(t)$. For each WCATS the synthetic utilization can be derived over the set of active tasks $S(t)$ according to

$$U_{syn}(t) = \sum_{i \in S(t)} \frac{WCET_i}{D_i}. \tag{4}$$

The synthetic utilization may change over time according to the current configuration of the control device. The schedulability boundary *U_B(t)* for a given task set consisting of *n* tasks is

$$\forall t: \qquad U_{syn}(t) \le U_B(t) = \begin{cases} \frac{1}{2} + \frac{1}{2n} & : n < 3 \\ \frac{1}{1 + \sqrt{\frac{1}{2}\left(1 - \frac{1}{n-1}\right)}} & : n \ge 3 \end{cases}. \tag{5}$$

If Equation 5 holds for all WCATSs, the control application will meet all real-time constraints. For large numbers of *n* the bound for the synthetic utilization $U_B(t)$ will reach the value *58,3%.*

Dynamic priority scheduling (Zoitl, 2007, Section 4.6.4): If we assume that the priority of a task may be changed during operation, dynamic priority scheduling will be considered. Herein the most important scheduling policy is earliest deadline first, which assigns the task with the shortest absolute deadline the highest priority. The scheduling boundary for dynamic priority scheduling is according to Zoitl (2007) given by

$$\forall t: \qquad U_{syn}(t) \le 1. \tag{6}$$

All tasks within the configuration of a control device will meet their real-time constraints, if Equation 6 holds for the WCATSs at $t = 0$.

6.1.2 Calculation of event chain execution time

For the evaluation of real-time execution for the control applications based on the schedulability rules given above it is necessary to define the WCET of the event chains within the control application. Based on Equation 4 the synthetic utilization of the WCATS can be calculated by using the WCET $WCET_i$ and the deadline D_i of the different event chains. We will describe a methodology for the calculation of the execution time of any event chain based on models of the IEC 61499 standard as well as the runtime environment R^3E. Both aspects are necessary because this evaluation has to fulfill the Requirements (2) “Execution semantics” and (3) “Underlying system configuration”, which describe the dependency of the behavior of control applications on implementation details of the control device.

Different approaches exist for the calculation of WCET especially in the field of real-time computer systems, because it is an important measure for guaranteeing whether a system fulfills its real-time constraints or not. Kopetz (1997, Section 4.5) describes the analytic calculation of the WCET for different types of tasks within an operating system and states that “at present, the systematic analysis of all the effects that determine the WCET of C-tasks[13] is still in its infancy”. The current state of practice is described as the combination of diverse techniques, which are based on measurements of the real implementation (all involved parts such as tasks, operating system, or internal services), restriction of architectural elements within the control device, generation of an effective set of test cases, and extensive testing of the complete implementation.

The evaluation of the execution time of an event chain can be based on the characteristics of the models of IEC 61499-1 (2005). Herein the execution of an FB network is defined by the

[13] A C-task is defined in Kopetz (1997, Section 4.5.3) as a preemptive complex task which has access to protected shared objects.

event connections in between the FB instances. The FB type gives a more detailed insight into the behavior of the event chain, as for instance the ECC can be used to analyze which events are emitted by a BFB in certain situations. Additionally the execution semantics of the runtime environment have to be incorporated because this is of special interest for the execution order of FBs within the event chain and therefore it may have an impact on the execution behavior, too. The exercise of WCET analysis is to determine which path of an event chain, i.e., which sequence of FBs, will take the longest time for execution[14]. Analogous the BCET of an event chain can be defined as the path of an event chain which will take the shortest time for execution. As a result the minimal and maximal execution times of an event chain represents the two bounds of the execution time of an event chain.

The following analysis is based on the work presented in (Sünder *et al.*, 2007a), which takes into consideration a previous version of a runtime environment developed at the ACIN. Continuative considerations with different versions of the R^3E have been provided by Brunnenkreef (2006) within the εCEDAC project (supervised by Thomas Strasser) and Mandl and Zhang (2008) under the supervision of the author, which provide also measurement values for the different parameters of a given control device configuration.

Calculation of execution time at application level

The WCET analysis of an event chain can be spilt up into two parts: Firstly the FB network is analyzed only taking into consideration the information available at the application level. Second the internals of FBs are analyzed. The first step is depicted in Figure 22 by using simple FBs with only one event output. The execution semantics within an event chain of the R^3E implementation can be simply defined as follows:

- If an output event is sent by an FB instance, each connected FB input is put into the first-in first-out queue within the event dispatcher.
- If there are several input events connected to an output event, each input event is put into the event dispatcher (the order is given by the creation of the event connections during the download of the control application).
- As soon as the execution of an FB instance has finished, the eldest input event is taken from the queue and issued to the corresponding FB instance.
- There is only one FB executed at the same time (no preemption of FBs). In case of CFBs the component FBs are considered as non-preemptive (except a component FB is again of CFB type).
- If an external event occurs, the external event handler puts an identification, which corresponds to the registered SIFB, into the event dispatcher. The invocation of the SIFB is treated in the same manner as the invocation of any other FB by the issue of an input event.

As a consequence of these execution semantics, the execution within an event chain can be characterized as sequential. The depicted execution flow given in Figure 22 starts with the introduction of the 'SIFB-ID' into the event dispatcher as a result of the occurrence of an external event. As soon as the 'SIFB-ID' is the eldest entry in the event dispatcher, 'SIFB' will be invoked for execution (bold arrow). The 'SIFB' instance emits an output event, which is connected to event input 'EI1' of 'FB1'. Accordingly this input event is put into the event dispatcher queue (dotted arrow). At this point in time there is no further entry visible within the event dispatcher, as the SIFB is just executed and no other event has been emitted (the figure includes a virtual situation within the queue). As soon as 'SIFB' has finished its

[14] See analogous definition in Kopetz (1997, Section 4.5.1): "The WCET analysis of a program which is written in a high-level language must determine which program path, i.e., which sequence of instructions, will be executed in the worst-case scenario. The longest path is called the critical path."

execution ‘FB1.EI1’ is taken from the event dispatcher and ‘FB1’ is invoked with ‘EI1’ as input event. This procedure is continued for the FB network in a similar manner. The execution stops as soon as there are no more input events within the event dispatcher.

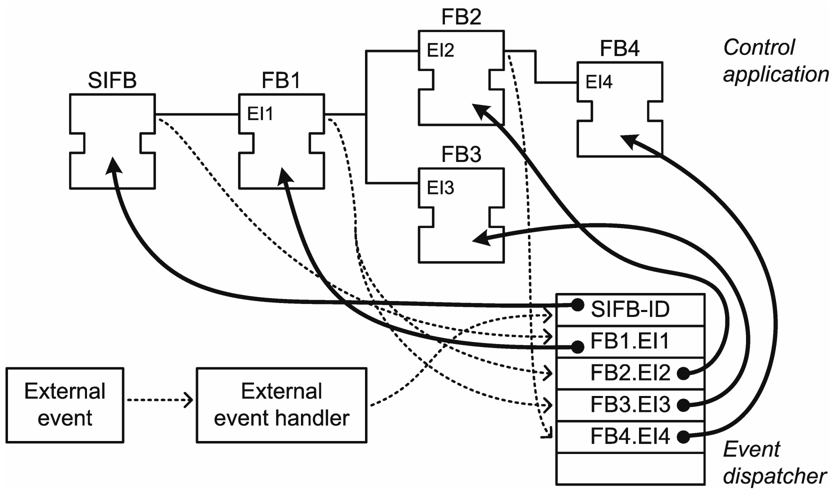


Figure 22: Execution behavior of control applications within R³E

According to this procedure a calculation of the execution time of the event chain can be based on a small set of parameters. Firstly the time needed to execute the external event handler T_{EEH} has to be considered. Herein an entry to the event dispatcher has to be added. The time needed for this insertion of an entry T_{entry} is considered separately, because it occurs in a similar way for each issued input event. Each FB within the event chain has a certain execution time $T_{FB,i}$. The invocation of an FB is characterized by the time $\mathrm{T}_{\mathrm{invoke}}$. As a consequence the execution time of the event chain T_{EC} with n FB instances (this number counts each FB instance as often as it is invoked as well as the SIFB which starts the execution) is

$$T_{EC} = T_{EEH} + n \cdot T_{invoke} + \sum_{i=0}^{n} T_{FB,i}(data) + n \cdot T_{entry} \ . \qquad (7)$$

The WCET of an event chain $WCET_{EC}$ is the maximal time necessary for the execution of the event chain. T_{EC} varies especially because of the internal behavior of the FB instances, which will be considered in a second step. There may be differences according to the issued input event, the current state of the FB instance, or the issued data, which is denoted as $T_{FB,i}(data)$ in Equation 7. The event outputs which are emitted by the FB instance are also depending on these parameters. So the FB instances influence the event chain execution time twofold: $T_{FB,i}$ may vary and the event flow depends on the FB instances.

Calculation of execution time of FB instances

The execution time of an FB instance depends on the FB type. For each type a unified calculation method can be provided which may be applied to a concrete FB type. We will start our consideration with the BFB type, which is characterized by the ECC structure and the algorithms. The CFB will be calculated based on the previous considerations. A SIFB needs to be considered based on the specified input behavior.

Basic FB: The procedure of executing a BFB is depicted schematically in Figure 23. If an event input is issued to the FB instance, first the associated data inputs have to be sampled. Then the ECC is evaluated and if a transition clears the active state of the FB changes and the associated actions will be executed. The execution of an action is split up into two parts: the algorithm (the execution time of an algorithm may depend on the current input and internal data) and the emitting of an output event. The emitting of an output event is again split up into two parts, the sampling of the associated output data and the event sending itself. As the FB instance does not have any information about the number of connected input events, the time necessary for event sending cannot be included in the calculation of the FB instance execution time. This part is already incorporated in Equation 7. The different actions within the active ECC state are executed one by one. If all actions have been executed, the ECC is evaluated again. At this time the issued input event has been cleared and only those transitions can be evaluated positively which consist of a Boolean condition only. If again a transition clears, the ECC will change its active state. The new active state is executed as already described above. If there is no more operation possible within the ECC, the execution of the FB is finished with the sampling of those data outputs which are not associated with any output event.[15]

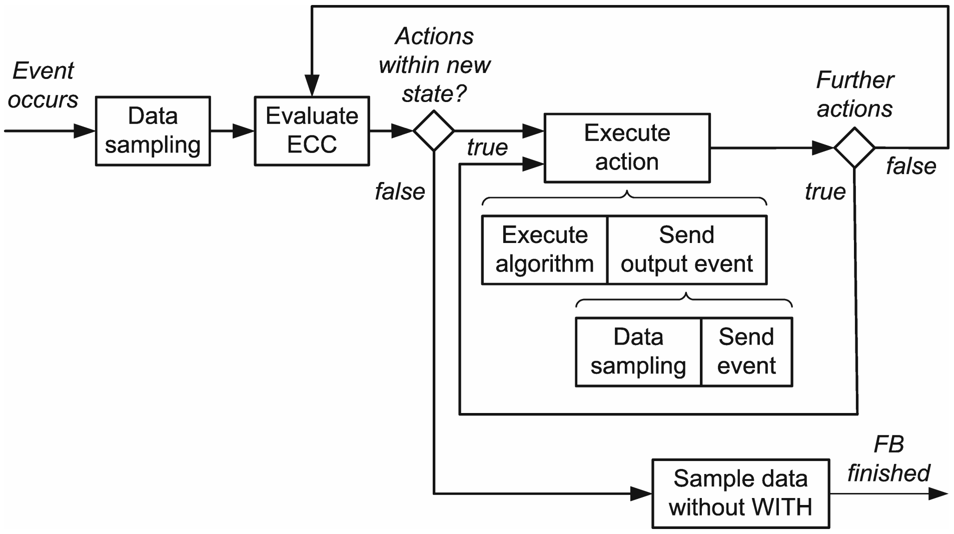


Figure 23: Execution behavior of BFBs within R^{3}E

The different parts of the execution time of a BFB type can be summarized as follows:

- ***Data sampling:*** There are three different situations where data has to be sampled: when the input event is issued, when an output event is sent and when the execution of the FB is finished. In all three cases the time necessary for this step may be calculated in a similar manner. An offset time $T_{DS,offset}$ exists which describes some effort independent of the data which is sampled. And for each data value (the number of data values is denoted generally by *dvalues*) which has to be sampled an appropriate time T_{sample} (dependent on the type of data *DT*) must be included. The time for data sampling T_{DS} can be described generally as

$$T_{DS} = T_{DS,offset} + \sum_{i=0}^{dvalues} T_{sample}(DT)\,. \qquad (8)$$

- ***ECC evaluation:*** The time necessary to evaluate the ECC $T_{ECC,eval}$ depends on the currently active ECC state and the sequence of transitions that need to be evaluated. It may take some time T_{AS} in order to find the currently active state and for the active state all outgoing transitions need to be evaluated as soon as a transition clears. Based on the definitions of a transition condition in IEC 61499-1 (2005) such a condition may be a single input event, a Boolean expression, or an AND relation of both. Accordingly the time necessary for a transition varies on the type of condition transition. As soon as a transition clears, no more transitions need to be evaluated. Equation 9 de-

[15] IEC 61499-1 (2005, Section 5.2.1.2) especially refers to this situation in note 7. As there is no information available when these output events are changed, the implementation of R^{3}E updates these output values each time the FB is executed.

scribes a general calculation of $T_{ECC,eval}$ which uses the variable $T_{Cond,i}$ in order to symbolize the different execution times for the different transition types. The addition ends if either one transition clears or no more outgoing transitions can be evaluated in the active ECC state (denoted by *cleared*).

$$T_{ECC,eval} = T_{AS} + \sum_{i=1}^{cleared} T_{Cond,i} \tag{9}$$

- ***Action execution:*** An action consists of an algorithm, or an output event, or both. For each of these items an appropriate execution time has to be taken into consideration. The time for executing an algorithm $T_{Alg,i}$ has to be evaluated for each algorithm seperately. As an algorithm may be any kind of source code, a WCET analysis for this part of the code should be provided. The analysis will be rather simple, because the containment for the algorithm is fixed by the FB instance and for example preemption is not allowed. As the sampling of output data is already given as $T_{DS,i}$ and the sending of an output event will be incorporated at the application level, the time necessary to execute an action $T_{action,i}$ is given by

$$T_{action,i} = T_{A\lg,i} + T_{DS,i}\,. \tag{10}$$

The overall execution time for a BFB instance T_{BFB} consists of the above mentioned elements according to the execution path depicted in Figure 23. The number of cleared transitions within the ECC will be denoted as *x*, and the number of actions within a state will be denoted as *y*. In addition to the above given parameters there may be a time for changing the active ECC state necessary, which is denoted as T_{CS}. The BFB execution time T_{BFB} depends on the issued input event, the internal data and the input data, depicted as $T_{BFB}(data)$.

$$T_{BFB}(data) = T_{DS,input} + \sum_{i=1}^{x}\left(T_{ECC,eval} + T_{CS} + \sum_{j=1}^{y} T_{action,j} \right) + T_{ECC,eval} + T_{DS,noWITH} \tag{11}$$

Composite FB: The execution of a CFB is similar to the execution of an FB network. The only difference is that an additional data sampling exists at the interface of the CFB. Based on Equation 7 and 8, the execution time for a CFB $T_{CFB}(data)$, which depends on the issued input event and the input data, is

$$T_{CFB}(data) = T_{DS,input} + \left(m \cdot T_{invoke} + \sum_{i=0}^{m} T_{FB,i} + (m-u)\cdot T_{entry} \right) + u \cdot T_{DS,output}\,. \tag{12}$$

The number of component FBs which are invoked during the execution of the CFB is denoted by *m*. The number of output events, which are emitted by the CFB is given by the variable *u*. Again it is not necessary to take into consideration the insertion of those output events which are emitted by the CFB. Therefore the term *(m-u)* is used for the execution of the component FB network (depicted by the outer brackets within Equation 12). The use of a sampling time for those data outputs which are not associated with an output event is not necessary, as IEC 61499-1 (2005, Section 5.3.1) states that those data outputs are represented by the component FB data outputs.

Service Interface FB: The execution of a SIFB is not defined by elements of the IEC 61499 standard. But sequence diagrams exist in order to describe the behavior of these FB type. An analysis of the execution time of an SIFB has to provide appropriate measurements for each of these different actions within the sequence diagram. A WCET analysis may become complicated as the internals of a SIFB are typically deeply involved in the internals of the runtime environment, the operating system, or the low level programming of the control device peripherals. A prerequisite for the incorporation of SIFBs into the execution time calculation of event chains is that all possible scenarios of SIFB invocation (this includes also a detailed description of the output events and the possible output data) are characterized by their functionality and timing behavior.

6.1.3 Evaluation of ECA influences on control applications

The evaluation of control applications regarding their real-time constraints consists of two tasks: the WCET analysis of the different event chains incorporated in the WCATS (herein only real-time constrained event chains are included) and the calculation of the synthetic utilization of the WCATS according to Equation 4. If the schedulability rules given in Equations 5 and 6 (depending on the scheduling policy applied in the runtime environment) are fulfilled, the control application will be executed without a violation of real-time constraints.

The influences of the first two phases within a system evolution step to the real-time behavior of the control application will be considered separately.

Download ECA

There are two things that need to be considered during the download of the ECA. On the one hand the download process itself may influence the execution behavior of the control applications. And on the other hand also connections between the control application and the ECA may be created, which lead to additional execution effort within the control application in the case of event connections.

- ***Download process:*** A main prerequisite for the use of DSE is a runtime environment which provides the ability to change the control logic during operation. The simplest case is the addition of further control applications which are not related to the existing ones. In this case no special engineering methodology is necessary, but the runtime environment needs to support the download of further applications without disturbances of currently executed applications. The R^3E provides this feature which has been proved by experimental measurements in Zoitl (2007). A short summary of these results is given in Appendix B.3. The download of an ECA will be applied without disturbances to the real-time constrained event chains within the control application.
- ***Interrelation of control applications and the ECA:*** If event connections exist between any part of real-time constrained event chains within the control application and the ECA, the WCET analysis of these event chains within the WCATS has to be repeated based on the changed situation. The enhancements to the execution paths within the event chains will be very simple, because the purpose of these connections is the synchronization of the ECA with the control application. In the initial state of the ECA there should be no actions within the ECA that are executed based on an event from the control application. The influence on the execution time of the control application will be limited to the invocation of in most cases one FB instance within the ECA which does not issue an output event. This can be stated as a design rule for the ECA. In addition there should be some spare execution time available because the addition of connections during the execution of a real-time constrained event chain provides some (according to Appendix B.3) limited additional effort.[16]

[16] Zoitl (2007, Section 5.3) has quantified the impact of a single management command (e.g., creation of an event connection to a started FB) on the real-time execution by an increase of execution time by one to ten percent. This is of course dependent on the size of the real-time constrained event chain, whereas Zoitl (2007) has taken into consideration a rather small test application. For a given system configuration appropriate test cases have to be provided in order to determine the necessary spare execution time. For instance Rasche and Polze (2005) describe a method for the calculation of processor resources that have to be reserved in order to

Initialization sequence

The initialization sequence within the ECA adds new elements to the control application without influencing the functional behavior. As depicted for instance in the example of a closed-loop control circuit (Section 4.2.3) the new controller as well as the input connections are added to the existing control application. Therefore, these parts are executed additionally within the context of the control application. In order to prove the influence on the real-time behavior of the control application, the worst case scenario has to be considered. As the main purpose of the initialization sequence is the addition of new FBs, connections, and parameters to the existing control application, the worst case is the final situation after the execution of the initialization sequence. Each new FB which is connected to a real-time constrained event chain has to be included in the WCET analysis of the event chain. The new WCATS has to fulfill the schedulability rules given in Equations 5 and 6. As already depicted above also in the initialization sequence a real-time constrained event chain is adapted during operation, which results in additional execution time within the event chain because of the influence of the basic reconfiguration service. A certain amount of spare execution time (see footnote 16) has to be available in order to operate the initialization sequence, too.

There is also another aspect which may violate real-time constraints of the control application and can be avoided by a further design rule for ECAs. During the execution of the system evolution step, events from the control application may be used to trigger the ECA execution. If these events stem from a real-time constrained event chain, a decoupling of these event paths is necessary in order to protect the control application's real-time constraints. The means for this decoupling have been described in Appendix B. Next to the initialization sequence this design rule should be applied for the reconfiguration sequence and the deinitialization sequence as well. If an additional execution of parts of the ECA exists based on such synchronization events a detailed WCET analysis of the control application has to be applied additionally.

Reconfiguration sequence

The reconfiguration sequence is the only time critical execution phase within the ECA. The proof of matching global and local consistency properties of the control application will be done by using verification by model checking. But based on the previous discussion the check for real-time behavior may be supported also by a WCET anaylsis. The WCATS will include the real-time constrained event chains within the ECA additionally. But it has to be stated that as depicted in Zoitl (2007, Section 5.3.3) the influence of basic reconfiguration services within a real-time constrained event chain, that influence another real-time constrained event chain, violates the consideration of independent tasks. The WCET analysis may be used to provide a rough estimation of the spare execution time within the control device, but it can not be used as sufficient check (see also the discussion of measurement results of R^3E in Appendix B.3).

6.2 Check for dependent operation

The dependent operation integrity characteristic has to be checked by some KAPPA-based calculations within all phases of the execution of a system evolution step. The download and the deletion of the ECA will be controlled by the engineering tool, which has to apply appropriate rules for the sequence of management commands in order to fulfill the dependent operation criterion. The other three phases are modeled by the ACS costumer in a free manner; therefore a check of consistency with respect to the sequential order of basic reconfiguration services is necessary.

apply dynamic reconfiguration without the violation of deadlines of real-time applications (it has to be stated that their model is based on cyclic execution of real-time applications).

As common element for both categories of dependent operation checks the dependencies of IEC 61499 management commands are taken into consideration. Based on these dependencies, the mechanisms for checking the initialization, reconfiguration, and deinitialization sequence as well as rules for the engineering tool for the download and deletion of applications are investigated. These rules may be applied also for algorithms which provide a skeleton of the FB network within the initialization, reconfiguration, and deinitialization sequence.

6.2.1 Dependencies of IEC 61499 management commands

IEC 61499 management commands are defined on the one hand in IEC 61499-1 (2005, Chapter 6.3) and on the other hand within the compliance profile of a runtime environment (if extensions or changes exist). The third possibility is that no standard compliant documentation exists, as it is the case for the enhancements of management commands within R^{3}E. Zoitl (2007) includes a detailed description of the supported commands which will provide the basis for this analysis. As the different basic reconfiguration services incorporate a management command we do have to take into consideration these different management commands and their dependencies for the check of dependent operation within the ECA.

CREATE

The CREATE management command is responsible for the creation of different kinds of elements.

- ***Create a resource or FB instance:*** A resource instance may be created within a device, if this device is available (which needs to be ensured by the ACS customer). An FB instance may be created within a valid resource instance or a device. In any case the type of the created element (resource or FB) needs to be available within the type library of the control device.
- ***Create a connection:*** A connection is defined by a source (event or data output of an FB) and a destination (event or data input of an FB). Source and destination need to be within the same resource (if no resource exists they need to be within the same device) and of the same type (different types are possible for event and data connections). In case of an adapter connection source and destination additionally need to have oppositional interfaces (a connection is only possible between a plug and a socket).
- ***Create a library element:*** A library element can be added to the control device's type library if the appropriate device is available and the format of the library element is provided in an appropriate manner.

DELETE

The DELETE management command provides the opposite functionality of the CREATE command. Correspondingly the following elements may be deleted:

- ***Delete a resource or FB instance:*** In order to delete a resource or FB instance the corresponding element has to be manageable. The possibility exists to use for example FB instances within the type definition of a resource type. Such an FB instance may not be deleted by a management command. Furthermore the operational state machine for managed FBs has to be considered, which defines that a DELETE command is only possible if the FB instance is in the states STOPPED and KILLED (see Figure 63). In case of a managed resource instance there should not be any active FB instance within the resource; all FB instances should be stopped before deleting the resource instance.
- ***Delete a connection:*** The only prerequisite for the deletion of a connection is that the connection is available within the control device.

- ***Delete a library element:*** A library element may be deleted within the control device's type library if it is available and no instances of this type are used within the control device.

WRITE

The WRITE management command provides the possibility to put a parameter to a data input or to an internal variable. In both cases the destination of the WRITE command has to be valid and the value of the parameter/variable has to be provided in the data type format of the destination. The opportunity of using the WRITE command also for internal variables is an enhancement of the R^3E.

READ

The READ management command provides the possibility to read any data input, data output, or internal variable (enhancement of R^3E). The prerequisites for the READ command are on the one hand that the source of the variable is valid and on the other hand that the expected data type format complies with the source format, e.g., a specific basic reconfiguration service for reading a variable of the type integer may be used.

START

According to the operational state machine for managed FBs (see Figure 63) the START command may be applied only if the FB instance is in the states IDLE or STOPPED. Within the IEC 61499 standard the START command is also mentioned to be used for applications, which is not supported by R^3E. This functionality has to be implemented within the engineering tool, which would have to start every FB instance within the application. The only restriction of such a sequence of START commands for FB instances is that the FB instance of type E_RESTART should be started at last, because the START command issues the initial event for the operation of the application. Otherwise the correct operation of the application may be violated because an input event may be issued to FB instances which are still in the IDLE state.

STOP

The STOP management command may be applied to a managed FB if it is in the state RUNNING (see Figure 63). Also the STOP command can be used for applications according to the IEC 61499 standard. This functionality is not directly available within R^3E and may again be implemented within the engineering tool. But as there may be instances of SIFBs within an application the STOP command should be applied to FB instances of the type E_RESTART at first, since a deinitialization of the application may be necessary. The main problem for the stopping of an application is that the engineering tool has to take care that the deinitialization is finished before all other FB instances are applied with the STOP command.

KILL

An FB instance may be applied with the KILL management command if it is in the state RUNNING (see Figure 63).

RESET

According to the operational state machine for managed FBs (see Figure 63) the RESET management command may be applied to FB instances if they are in the states STOPPED or KILLED.

QUERY

The QUERY management command does not have any consequences for the execution behavior of the control device and its applications. Therefore no prerequisites have to be stated for the application of the QUERY command. If the control device is available any information provided by the runtime environment may be asked for via the management application or by using an appropriate basic reconfiguration service.

Summary

The different dependencies for the IEC 61499 management commands are summarized roughly in Table 3. Herein the functionality for manipulating the type library of a control device as well as the starting and stopping of whole applications is neglected, as these managemant commands are not relevant within the execution of a system evolution step.

Command	Object	Dependency
CREATE	Resource	The device has to be valid; resource type has to be available.
	FB	The device or resource has to be valid; FB type has to be available.
	Connection	Source and destination have to be valid; type of source and destination need to fit to each other.
DELETE	Resource	The resource instance is deletable; any managed FB instance should be stopped within the resource.
	FB	The FB instance is deletable; it should be in the states IDLE or STOPPED.
	Connection	The connection should be available.
WRITE	Parameter	The target (data input) should be valid; the format has to comply with the data type of the data input.
	Internal variable	The target (internal variable) should be valid; the format has to comply with the data type of the internal variable.
READ	input or output variable	The target (data input or output) should be valid; the expected format has to comply with the data type of the data input or output.
	Internal variable	The target (internal variable) should be valid; the expected format has to comply with the data type of the internal variable.
START	FB	The FB instance should be in the states IDLE or STOPPED.
STOP	FB	The FB instance should be in the state RUNNING.
KILL	FB	The FB instance should be in the state RUNNING.
RESET	FB	The FB instance should be in the states STOPPED or KILLED.
QUERY	Anything	The device has to be available.

Table 3: Dependencies of IEC 61499 management commands

6.2.2 Correct order of basic reconfiguration services

In order to check the dependent operation integrity characteristic within the initialization, reconfiguration, and deinitialization sequence within the execution of a system evolution step different aspects of the above described dependencies need to be taken into consideration.

- ***Static dependency check:*** Different dependencies mentioned above can be checked without information about the use of the basic reconfiguration service within the ECA. These are for instance the data type of a variable which should be read/written or the availability of an FB type within the control device. These dependencies may be checked during the modeling of the ECA automatically by the engineering tool.
- ***Dynamic dependency check:*** All other dependencies are related to the position of the basic reconfiguration service within the execution of the ECA. Only if the basic reconfiguration services are applied in the correct order, a successful execution of the sys-

tem evolution step can take place. As a consequence the check for these dynamic dependencies has to be based on a dynamically adapted KAPPA vector within the evaluation framework. According to the execution sequence of basic reconfiguration services within the ECA the appropriate changes have to be applied to a virtual KAPPA vector in order to provide the basis for the check of the above given dependencies.

The final result of the execution of a system evolution step (the situation after finishing the deinitialization sequence) can be checked additionally by a comparison with the envisaged new system state. The check for the dependent operation integrity characteristic cannot provide an answer to the question if the system evolution will be executed without disturbances to the control application. But it checks if the used basic reconfiguration services are in principle able to reach the new system state. An analog comparison of the situation after the execution of the initialization sequence may be very helpful, too, as the evaluation of the reconfiguration sequence by model checking will be simplified (principle failures due to missing elements are neglected). Herein the ACS customer has to define the envisaged initial situation for the reconfiguration sequence during the engineering process.

Based on the dependencies of basic reconfiguration services and a description of the KAPPA vector at the initial and the final state of a sequence within a system evolution step the principal order of the basic reconfiguration services can be generated by an appropriate algorithm within the engineering tool. This would support the ACS customer to a high extent and simplify the usage of the engineering methodology for DSE. This algorithm will be similar to the download and deletion of applications, as it is depicted in the next section.

6.2.3 Creation and deletion of applications/application parts

There are several parts within the engineering process for DSE which can utilize automatic mechanisms for the creation or deletion of applications or application parts. On the one hand these are the download and the deletion of the ECA (the rules given within the following description can be applied for any application). Within the other three phases of a system evolution step, the automatic generation of the ECA for the initialization, reconfiguration, and deinitialization sequences (at least a basis for a detailed modeling by the ACS customer) may benefit of such an automatism. We will consider the principle mechanism for the creation and the deletion of applications or application parts. The initial and the final KAPPA vector (in detail the differences within the application model) are used as input for this algorithm. If elements are mentioned in the algorithm which are not necessary for the concrete difference in the KAPPA vector, the appropriate line may be neglected.

Creation of applications/application parts

- Create resource instances
- Write parameters of the resource instances
- Create FB instances
- Write parameters of FB instances
- Create data connections
- Option for application parts (if they are involved in an existing application which is already in operation):
    - Start FB instances which need an initialization event (these are in most cases of SIFB type)

  - Prepare necessary input parameters for the initialization as well as issue the input event for the initialization.[17]
  - Check if the initialization was successful.
  - Rewrite those parameters which have been changed for initialization.
- Create event connections
- Start FB instances: In order to avoid problems based on the event flow in between the FB instances the order of the issued start commands to FB instances should be in inverse sequence to the execution flow. For an independent application this rule leads to the above proposed situation that the last FB instance which receives a START command is the E_RESTART FB.

Deletion of applications/application parts

For the deletion of an application the above mentioned order can be inverted (without the mentioned option). Only two differences exist:

- The STOP commands issued to the FB instances are now ordered according to the execution flow. The first FB is of the type E_RESTART, which will issue a STOP event. The execution flow based on this STOP event should trigger the deinitialization of the application.
- Parameters do not need to be deleted separately.

If application parts or applications without an explicitly modeled deinitialization have to be deleted, a slightly different procedure has to be applied:

- Stop all FB instances which do not need a deinitialization (again the order of the execution flow may be followed).
- Delete event connections.
- Prepare necessary input parameters for the deinitialization as well as issue the input event for the deinitialization (see footnote 17).
- Check if hte deinitialization was successful.
- Stop all FB instances which are in the state RUNNING.
- Delete data connections.
- Delete FB instances.
- Delete resource instances.

6.3 Check for requirements of resources

The last category of properties which may be evaluated by KAPPA-based calculations concerns the requirements of resources. Such resources may be any functionalities within the control device that are necessary as basis for the use of DSE. For instance, the set of basic reconfiguration services, which may vary between different runtime environments, has to be checked in advance before modeling an ECA. For our considerations we will focus on the requirements of resources which may change dynamically during the execution of a system evolution step. These are the type library and the available memory.

[17] IEC 61499-1 (2005, Section 6.1) defines different standard inputs and outputs for SIFBs, which represent the necessity of initialization and deinitialization. As inputs the event input INIT and the data input QI (true in case of initialization, false in case of deinitialization) are forseen. The success of a (de)initialization is documented by the event output INITO and the data outputs QO and STATUS.

6.3.1 Type library check

The set of types within the type library of a control device will not be changed within the ECA itself, therefore it can be considered as static for one system evolution step. But as an ACS is typically engineered by different ACS customers it is necessary to check if the current situation within the type library satisfies the needs of the system evolution steps. If any violation is detected, the necessary element types may be added by the engineering tool before the execution of the system evolution step is triggered.

The following element types are supported within an IEC 61499 control device:

- Data types (a data type may have any other data type as prerequisite)
- Adapter types (an adapter type may have any data type as prerequisite)
- FB types (an FB type may have any data or adapter type as prerequisite, in case of CFBs also any FB type may be necessary)
- Resource types (a resource type may have any data, adapter, or FB type as prerequisite)
- Subapplication type (the R^{3}E does not support this element type, as a subapplication is handled only within the engineering tool)

6.3.2 Available memory check

The second type of requirements of resources is the available memory within the control device. For a very abstract consideration of this topic the rule seems to be very simple, as the amount of available memory has to be compared to the amount of necessary memory requested within the different phases of a system evolution step. But when taking a closer look to the practice of memory management within a computer system we will find highly sophisticated concepts. Douglass (1999, Section 2.6.1) describes this situation regarding predictability for operations that influence the memory of a real-time computer system. He subdivides the problem into different aspects:

- “Execution memory, where the executable code resides”
- “Data memory: 1. stack, 2. heap, 3. static”

Orthogonal to this view on the usage of memory, also the persistence of memory has to be considered, which may be distinguished in non-writeable persistent, writeable persistent, and volatile. Additionally modern operating systems implement different memory management policies such as paging or virtual memory as for instance described in Blunden (2003). A very problematic situation occurs in addition due to the dynamic changes in memory consumption, as it is the case especially for DSE. The memory segments will be fragmented as different areas within the memory will be freed for instance due to the deletion of FB instances within a system evolution step.

In order to provide a sufficient check whether the necessary memory is available within the control device three aspects have to be taken into consideration:

- The memory consumption of the different basic reconfiguration services, which of course depends on the concrete element that should be created or deleted.
- The memory management policy of the control device as well as the overall memory configuration (which kind of memory segments are available).
- The current state of memory usage within the control device, which is again part of the KAPPA vector.

The last aspect is the most critical one in order to apply the check for available memory, as the current situation of the overall memory has to be visible within the engineering tool. It is not sufficient to know the overall amount of available memory, since due to fragmentation there may be not enough atomic free space (a piece of free memory without fragmentation) for the creation of an FB instance although the overall free memory is much bigger. Currently only development tools for certain RTOSs deliver such a detailed insight into the memory usage of the system, as for instance described in (Wind River, 2007). Within the so-called run-time analysis tools also a tool exists which provides information about the memory usage in all details and therefore it may be used in the engineering tool for DSE in order to check the available memory for a system evolution step.

6.4 Summary

The different properties of the evolution specification are verified by two different means. On the one hand verification by model checking has to be provided for the reconfiguration sequence. But on the other hand the evaluation of properties based on KAPPA-based calculations is sufficient for all other phases within a system evolution step. It can be used to fulfill the requirements of verifying the download of the ECA, the initialization sequence, the deinitialization sequence, and the deletion of the ECA.

The three different types of calculations have been described as follows (see also Table 2):

- The global and local consistency of the control application can be reduced to the evaluation of the temporal behavior within the download of the ECA and the initialization sequence. Herein the mechanisms of the runtime environment as well as the operating system have to be considered. Based on schedulability rules and an appropriate method for the calculation of the necessary execution time of the contol application it can be checked whether the real-time constraints will be met or not.
- The consistency of the ECA regarding the dependent operation property leads to the consideration of the temporal order of execution of basic reconfiguration services or more general the IEC 61499 management commands. As these commands change the current system state (KAPPA vector) it is necessary to determine their dependabilities, as for instance a connection to a new FB instance may be created only if the FB instance has been created in advance. As a consequence of these dependabilities especially the download and deletion process can be described by an appropriate order of management commands. But also for the freely programmable ECA the order of basic reconfiguration services can be established automatically based on the information of the initial and final states of a given sequence in the system evolution step. The evaluation checks if the dependencies of a management command are fulfilled based on the current KAPPA vector, which is changed as soon as a management command is applied to the control device.
- The evalution of requirements of resources is the last aspect which has to be considered by KAPPA-based calculations. The complexity of this property depends on the concrete resource which is taken into consideration. A check for elements within the type library of a control device can be performed very easily. But the evaluation of the available memory within a control device becomes very complicated due to sophisticated memory management policies and the fragemtation of memory because of dynamic changes within the configuration of the control device.

Chapter 7

Evaluation of properties by model checking

Verification by model checking has been identified as the appropriate means for the evaluation of the core part of DSE. Based on the discussion in Section 5.2.2 only the reconfiguration sequence has to be taken into consideration by model checking, in detail the system integrity properties for

- Global and local consistency,
- Active references,
- State management, and
- Real-time constrained operation.

In addition KAPPA-based calculations for dependent operation and requirements of resources have to be applied in order to evaluate all properties within the evolution specification for the reconfiguration sequence (see Chapter 6)

The process of model checking can be split up into three parts (see Section 3.6). Firstly a formal model of the system has to be established in the modeling language of the model checking tool. Secondly the specification has to be defined, in most cases temporal logic is used for this purpose. The third and last part is the evaluation whether the system model satisfies the given specification or not. Herein a model checking algorithm is applied and automatically checks the given properties. We will investigate the first two parts of the model checking process within this chapter. On the one hand a detailed description of the different architectural elements within a control device are given, in order to fulfill the Requirements (1) "Temporal behavior", (2) "Execution semantics", (3) "Underlying system configuration", and (4) "Modeling dynamic reconfiguration". On the other hand the necessary properties of the evolution specification as well as their formulation are examined, with special respect for Requirement (8) "User-friendly definition of specifications".

The formal models for different aspects of the control device will be discussed on a concrete example, the demonstration control device used also in Chapter 8. As modeling language NCES (see Appendix C) will be used in order to provide concrete examples.

7.1 Architectural elements of the system model

The formal model of the system is the starting point for the model checking process. Based on the evaluation framework for DSE already depicted in Section 5.1.2 we have to consider at least three elements: the process under control, the control application, and the evolution control application. Additionally several sources of disturbances exist, which need to be taken into consideration for the evaluation of the specification properties. These are on the one hand explicitly modeled disturbances (e.g., a special condition within the model of the process under control) and on the other hand aspects that come from details within the implementation of the control device (e.g., hardware capabilities or other applications).

In order to incorporate the different aspects of the system model and to provide an appropriate model to fulfill the different Requirements (1) “Temporal behavior”, (2) “Execution semantics”, (3) “Underlying system configuration”, and (4) “Modeling dynamic reconfiguration” we have to consider a control device as the smallest part within the system model. It is not sufficient to model only some aspects such as the control application and the ECA without taking into consideration also all other elements within the control device. In addition to the control device the process under control (at least the portion which is relevant for the control application and disturbances to the process) and the communication network may be necessary. The communication network provides the possibility to extend the overall system model to several control devices.

A schematic of the architectural elements within the system model, when only one single control device is taken into consideration, is depicted in Figure 24. The gray shaded elements represent the intrinsic parts of the control device, which are independent from the application scenario. On the one hand the process under control and on the other hand the communication network are included as interfaces of the control device to the environment. Within the control device the control application and the ECA represent those parts which are of special interest for DSE: the model of the application, which will be changed during operation, and the model of the application which executes these changes. The elements additional tasks and other applications represent disturbances in terms of influences on the execution behavior (especially the temporal behavior) within the control device.

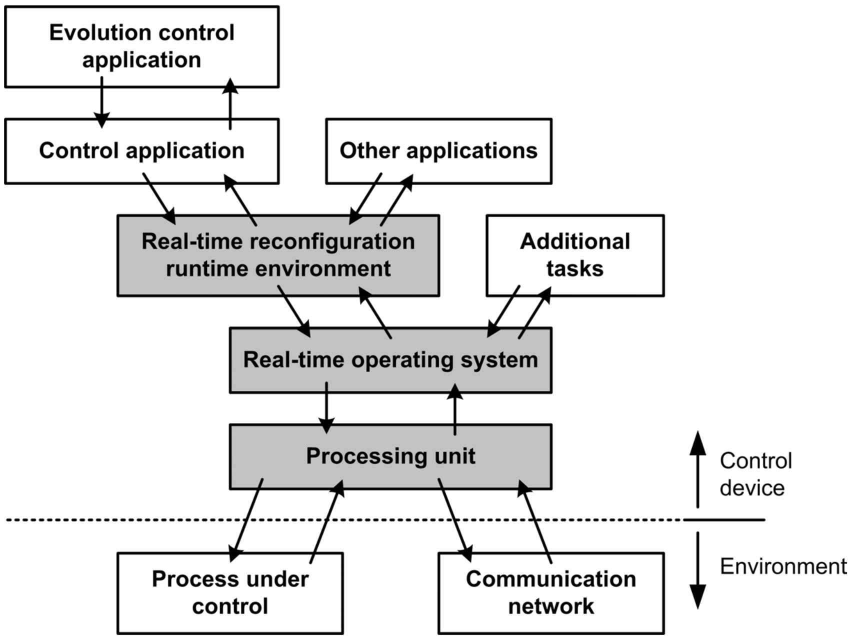


Figure 24: Architectural elements of the system model

The different elements of the system model architecture will be described in a general manner in the following paragraphs. An arbitrary modeling language may be used for these elements, according to the model checking tool that should be used for the evaluation of DSE. A more detailed description of several elements based on the modeling language NCES will be provided in the following sections.

Process under control: The model of the process under control (the plant) is the basis for a closed-loop modeling as described in Section 5.1.1. The model creation for this element is highly manual, because no predefined structures exist within a general ACS. Of course, libraries of model parts (pre-given or typical elements in the plant) may be available in an engineering environment, but the ACS customer has to establish the model of the process under control for each particular plant by hand. Examples for the modeling of processes are available in different publications: Vyatkin and Hanisch (2003) describe a model for a drilling station or Hanisch *et al.* (2006) present the model of a lifter. By using a transformation from UML to NCES Lobov *et al* (2006a) provide a framework that may utilize already existing descriptions of a plant in order to generate the model of the process under control. In addition to the nominal behavior of the plant it is also important to include failure situations to the models, as depicted for instance in Vyatkin and Hanisch (2003) by exceptions within the process. The more failure situations are incorporated in the model of the process the better and more realistic results will be achieved via verification by model checking.

Communication network: An ACS based on the ideas presented in Section 3.1 typically consists of several control devices and HMI devices which cooperate via a communication network. The influence of the communication network on the model of the system can be described in two different ways, depending on the target of the evaluation: (i) the communication network may be used as boundary of the system model and (ii) it may provide the interaction channel between models of different control devices. The first possibility is important in order to focus on the relevant parts for the DSE within the system model. If only one control device is involved in the system evolution step, all other control devices may be incorporated only by a model of the messages sent to the relevant control device. The second possibility is used for situations where several control devices are included within a DSE. Herein apart from a pure data exchange between the control devices also the temporal behavior of the communication network may be included into the system model, as this is also some kind of disturbance to the process of DSE (especially for temporal properties as there may be at least some latency due to communication).

Processing unit: This item represents all elements of the control device which provide the basis for the execution of source code. Generally speaking, this item includes the microprocessor and the memory of the device. Of course, a detailed formal model of the microprocessor may not be useful for the evaluation of a control device, because too much information is necessary and the details of the model would require an enormous effort for their design. But for the description of real-time behavior, this element is of special interest. The computational power of the processing unit will be abstracted in terms of execution time of different portions of the source code (see also the calculation of the WCET of an event chain in Section 6.1.2). If the different parts of the source code within a control device are characterized by their execution times, the real-time behavior will be included in the formal models because it is possible to sum up and measure the overall execution time along paths within the system model. But it has to be noted, that this simple model for the effect of the processing unit is limited by the structure of the microcontroller. It has to be evaluated if this model may be applied for a concrete example. For instance, in the case of multi-core processors or sophisticated algorithms for branch prediction, a more detailed consideration may be necessary.

The topic of predictable execution time is in general very important for the development of real-time computer systems. Bouyssounouse and Sifakis (2005, Section 7.3) state as relevant challenge and work direction also the WCET analysis: “The determination of precise bounds on the execution times of real-time software critically depends on the predictability of the processor architecture. They are the more precise, the more predictable the processor architecture is. Processor architectures started to being used today reach the limit of non-deterministic behaviour that makes computation of precise upper bounds possible. An interesting research direction is to identify principles for the design of processors that perform well both in the average and in the worst case.”

Real-time operating system: The basis for the software architecture of a computer system is given by the operating system. As control devices belong to the class of real-time computer systems, real-time operating systems are in use. An RTOS provides a deterministic behavior for the execution of different tasks (in general execution contexts) according to a scheduling policy. Furthermore the interaction between processes and resources within the control device are under control of the RTOS. A very important point for ACSs is also the usage of interrupts, which are the sources of external events based on a state change in the plant (e.g., a rising edge of a sensor signal) or also within the control device (e.g., the timer). If the mechanisms of the RTOS are incorporated in the model of the control device, the interaction between different parts of the control application, the external events, and other programs can be modeled in detail. The description of the RTOS has to include two aspects, the functional as well as the temporal behavior. The different actions within the RTOS, as for instance the switching time between different tasks in the case of preemption, have to be measured and the execution times need to be added to the formal model.

The topic of formal description of real-time applications and especially real-time operating systems and their applications has been discussed in several applications. Corbett (1996) aims at the formal description of Ada programs, with special attention to its concurrency and real-time constructs. For the scheduling of tasks the task dispatching policy has been modeled. The formal description is based on a constant slope hybrid automata, whereas a transition represents a code region and execution time is modeled with an appropriate delay before its (instantaneous) transition. Cofer and Rangarajan (2003) describe the verification of the DEOS real-time operating system by using SPIN model checker. In detail, the rate monotonic scheduler policy is implemented and analyzed in contrast to an event-triggered system environment. Waszniowski and Hanzálek (2003) depict their model of a real-time operating system with timed automata. They claim that the timed automata theory is not suitable for modeling preemption; therefore they focus on cooperative scheduling. The formal description includes for instance inter-process communication via semaphores or context switching time. In Krákora et al. (2004) the combination of the real-time operating system with communication via *Control Area Network* (CAN) is utilized for the verification of a distributed control application.

Real-time reconfiguration runtime environment[18]: The IEC 61499 runtime environment is executed as a set of tasks within the operating system. There are several aspects that need to be modeled within the runtime environment: (i) the event propagation within FB networks, (ii) the execution of the different types of FBs, and (iii) the interface to external event sources and the handling of these external events within the runtime environment. These aspects can be summarized as stated in Requirement (2) as "Execution semantics". In case of the R^{3}E the event propagation is implemented by using an event dispatcher, as depicted in Sections 6.1.2 and B.1. The external events are handled by the external event manager, which introduces a notification for the invocation of SIFBs into the event dispatcher, too. The different parts of the runtime environment need a certain execution time, which has to be incorporated to the models in order to model real-time behavior, too. The R^{3}E additionally implements a concept for real-time execution within the control application based on the event chain concept. According to the initial origin of event propagation (these are SIFBs that are capable to introduce events into the control application) the different event chains are executed as separate tasks within the RTOS. The use of basic reconfiguration services within SIFBs is described below within the ECA.

Additional tasks: Although IEC 61499 provides very general means for the implementation of control applications as well as additional functionality, several other programs will be available within the control device. For instance, a web server for simple access to diagnostic and supervisory data may be part of the control device, too. On the other hand additional tasks are also necessary for services within the control device that will be used by IEC 61499

[18] In general any runtime environment may be used within the control device. But as no equivalent runtime environment exists that provides the necessary basic reconfiguration services we directly mention R^{3}E.

applications, as for instance the interface to the communication networks. Such services are usually handled in separate tasks. The purpose of an evaluation process is to prove the behavior of the control application. But any additional task within the control device may disturb the execution of the control application (at least by consuming execution time). The effect on the execution behavior of the control application depends on the priority of these tasks as well as their runtime behavior: the frequency of invocation and the duration of operation. If no interrelations exist to the control application, a simple model for additional tasks may be based on the occurrence models of external events mentioned in Section 6.1.1. Otherwise a more detailed model of the functional and temporal behavior of these tasks as well as their interrelation with other tasks of the control device has to be included in the system model.

Control application: The IEC 61499 application which is effected by the DSE has to be modeled within the element control application. Next to the execution of the FB network by the runtime environment also interrelations exist with other applications (internal communication or by using the communication network) or the plant and additional tasks via SIFBs. The model of the control application consists of different parts: On the one hand the FB network has to be translated into the modeling language, and on the other hand all FB types and their formal models have to be available. The models of the different FB types may be provided in appropriate libraries if they are part of the initial setup of the runtime environment. Or they are generated by the ACS customer itself. In this case an appropriate support for the automatic transformation into the modeling language with only little manual work has to be available.

Other applications: This element includes IEC 61499 applications, too. But they are not affected by the DSE and need not to be considered in all details. Of course, a similar model as described for the control application may be established (also mainly automatically) including a detailed behavioral description. But it may be also sufficient to include only a very abstract behavioral description of these applications. As basis again the event occurrences of external events (Section 6.1.1) can be used. The execution time of other applications may be further determined by the WCET of event chains as described in Section 6.2.2. Based on these two parameters, the characteristic of invocations of an application and the possible execution times for these invocations (the disturbances to the execution of the control application and the system evolution step) can be described rather simple. Based on the level of interrelation between the control application and other applications an appropriate level of abstraction may be used for the system model.

Evolution control application: The ECA is an IEC 61499 application, therefore the same procedure as for control applications can be applied for the formal model of the ECA. But some of the FBs within the ECA belong to a special type, because they incorporate IEC 61499 management commands, the so-called basic reconfiguration services. These are special types of SIFBs and are part of the runtime environment. Next to the formal model of the FB itself also the effect of the management command to the control application has to be described by using the modeling language. Based on the evolution engineering approach only the reconfiguration sequence has to be taken into consideration, which is characterized as a time-critical sequence which executes the switch from the old system state to the new system state incorporating transition management. As depicted for instance in the closed-loop control circuit (see Section 4.2.3) it is not necessary to create new resources or FB instances within the reconfiguration sequence. The basic reconfiguration services may be restricted to actions such as the creation of connections as well as the reading and writing of parameters. This limited set of basic reconfiguration services needs to be represented in the system model in order to be able to check the different properties of the evolution specification.

7.2 Modeling real-time behavior

The real-time behavior of a control application is a very important aspect for the representation of the functionality of a control device and needs to be analysed in detail. The general situation of timing analysis for real-time computer systems already was described in Section 6.1.2 based on Kopetz (1997). For the incorporation of real-time behavior into the formal model of a system we have to mention the results from the WCET analysis, because the time values for the formal model have to be based on an appropriate analysis. Bouyssounouse and Sifakis (2005, Section 7.3) state that "some analyses are only possible, once the machine-code level is reached. Reliable and often precise upper bounds on the execution time of embedded programs can be obtained when all the information about the hardware platform is known."

The formal model of a system often represents an abstract behavior without the transformation of source code into a formal description. In recent years also model checking based on source code (see also Section 3.6.3) is available for some programming languages such as C or Java. But the scope of such a source code-based verification is limited to small portions of a program. Overall system architectures, as for instance described for a control device in Figure 24, with several programs and an operating system are still not possible up to now. The concept for real-time modeling in formal descriptions proposed for this thesis can be described as follows:

- Fragmentation of the overall system architecture into single units regarding the functional behavior of the system. For instance a certain method or object within a software program or a task within the operating system may be considered as a single unit.
- Extraction of the control flow between the different single units within the system architecture.
- Measurement of the timing behavior of the different parts of the system architecture.

In contrast to a classical WCET where different scenarios have to be explored in order to extract the longest execution path within a software program, the formal model of the system will include all these different possibilities of execution paths based on the control flow within the extracted single units. The different parts of the system architecture are interrelated according to their functional behavior and additional timing parameters for their temporal behavior. The formal model includes all possible paths that result in the different execution time of a software program within the given system architecture, and of course also the WCET and BCET as boundaries. But the ACS customer does not need to explore the different execution paths of the overall system architecture by himself. All possible combinations of execution paths and interrelations between the different parts of the system architecture will be established automatically by the model checking algorithm.

Bouyssounouse and Sifakis (2005, Section 7.4) describe the problem of current practice in WCET as follows: "Testing is often performed to measure real-time execution time and response time e.g. to check resource utilization or obtain an estimation for the worst case execution time. However, using this approach is very problematic because it is difficult to obtain safe and accurate bounds." By using the above described incorporation of real-time measurements into the system model and appropriate model checking it is possible to achieve more detailed checks for the temporal behavior of a real-time computer system.

A good example for the accessible benefits of this methodology can be considered based on the calculation of the execution time of event chains as presented in Section 6.1.2. The evaluation of the execution time is split up into two levels, the application level and the FB level, whereas the application level is highly influenced by the FB level due to the generation of output events based on the internal state of an FB. If we consider the different parts within the Equations 7 to 12 as single units of the formal model, which are afflicated with an execution time, the system model will include all possible execution paths of the event chain.

An appropriate analysis of the system behavior will also provide the parameters WCET and BCET of the event chain. But even more important is that both the temporal and functional behavior of the event chain are incorporated in the same system model and can be used for enhanced analysis, e.g., for DSE.

Timed model for single units of source code in NCES

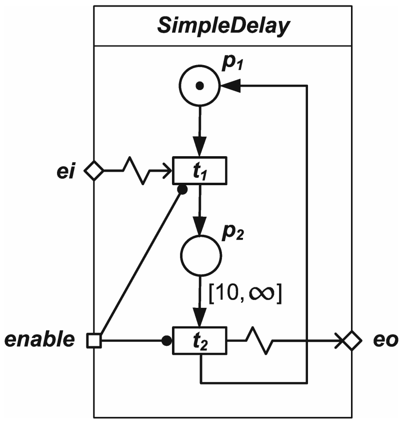
The representation of the combined temporal and functional behavior of a single part of source code depends on the modeling language and the possibilities based on the model checking algorithm. Within this thesis we will use the modeling language NCES and the model checker SESA, as described in Appendix C. In general the model of a single part of source code has to fulfill three conditions:

- The functional behavior has to be modeled according to the temporal order of the source code implementation.
- A time delay can be added to this sequence according to the execution time of this part of source code.
- The execution time modeled in the system model has to be interruptable.

The first condition requests the modeling of a sequential execution flow in the formal model similar to the execution of the source code. In most modeling languages the functional behavior can be represented in a more parallel manner (e.g. in Petri nets or timed automata), but due to the incorporation of timing behavior it is necessary to restrict these possibilities in order to achieve a sequential behavior in the formal model. It is essential to model the execution flow of the formal model in the same way as the execution flow of the source code. The second condition for the representation of time is dependent on the concepts of the modeling language. In case of NCES a timed flow arc is available, which is enabled by an internal clock of a place. But this simple concept will not be sufficient when taking into consideration also the third condition, an interruptable model of execution time. Of course this condition is only necessary as soon as preemption is used within the system architecture of the control device. In case of an RTOS this is typically the case for different tasks according to the scheduling algorithm. But it will occur in very simple control devices without operating systems, too. Sünder *et al.* (2007a) describe the formal model of a control device based on a 16-bit microcontroller without operating system. But also in this case preemption happens within the IEC 61499 runtime environment as soon as an interrupt occurs. The microcontroller switches to the interrupt service routine and disrupts the execution of the control application.

In order to model a disruption of the execution of one part of a NCES model it is sufficient to connect each transition with a condition input which is only true as long as the corresponding part of the model is executed. In case of preemption the condition input will be switched to false and the execution of the model is blocked. Figure 25a depicts this situation for a module with simple functional behavior. As soon as input event ‘ei’ occurs the marking flows from place ‘p_1’ to place ‘p_2’. Here a delay of *10* time units is modeled before the output event ‘eo’ is issued and the marking flows back to place ‘p_1’. The functional behavior of this ‘SimpleDelay’ module is a delay of the event flow by *10* time units, which may be representative for a certain part of the source code within the control device. The condition input ‘enable’ represents the model for preemption as described above. But what happens in case of preemption? If the condition input ‘enable’ is false while the marking is in place ‘p_2’ (e.g., because the execution of this part of source code may be preempted by an interrupt) the internal clock of place ‘p_2’ will still be increased, although another part of source code is executed, because there is a marking within place ‘p_2’. The temporal behavior will not be correct if we use simple timed arcs with preemption in NCES.

An appropriate solution for this problem is depicted in Figure 25b. Herein the same functional behavior (a delay of the event flow) is modeled, but the time which elapses in the model is represented as a number of markings. As soon as a marking is available in place 'p_2' again the internal clock starts counting. But after one time unit transition 't_3' clears and adds one marking to place 'p_2'. As soon as the number of markings has reached *11* (after 10 time units), the flow arc from place 'p_2' to transition 't_2' is enabled and the output event 'eo' is issued. If the condition input 'enable' is set to false while the time delay modeled by markings in place 'p_2' is active, also the increase of markings is stopped (the internal clock of place 'p_2' will still be increased, but without effects to the temporal behavior), because transition 't_3' is disabled via 'enable', too. By using such a model for time delays, the disruption of timed system models can be used without the violation of the temporal behavior.

a) 

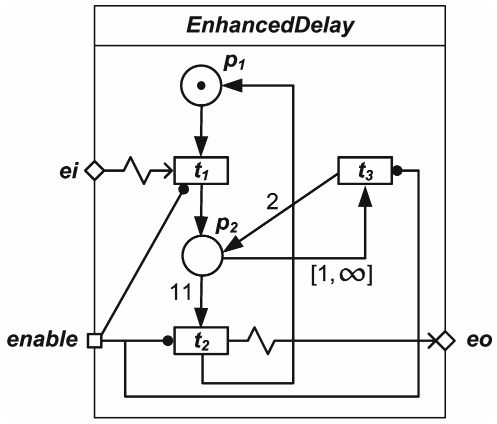
b) 

Figure 25: NCES representation of a delay (a) by a timed arc and (b) by markings

7.3 Dynamic reconfiguration support in formal models

As most important enhancement of the formal models of a control device the support for basic reconfiguration services within the system model has to be provided. As already discussed in Section 3.6.3 no formalism exists in order to incorporate a dynamically changing system model into the model checking algorithm. Based on a given specification and a given model the state space of the system will be explored for model checking, but within this process the model needs not to be changed. Different approaches in literature apply changes to models by certain rules (which may be related to a dynamic reconfiguration process) and then use the changed models for model checking. But for the verification of the reconfiguration sequence by model checking it is necessary to incorporate the execution of basic reconfiguration services during operation of the system into the model. The ECA itself can be modeled similarly to any other IEC 61499 application, but the influence to the control application has to be modeled, too.

The scope of DSE is very broad, and the inclusion of all possible basic reconfiguration services into the system model seems to be not realistic due to the limitations of the model checking approach. But as discussed in Section 5.2.2 the verification by model checking is only necessary for the reconfiguration sequence within a system evolution step. Table 4 gives an overview on the basic reconfiguration services which may be used within the different sequences of a system evolution step:

- ***Download of the ECA and initialization sequence (RINIT):*** Both sequences are used in order to prepare the DSE. The used basic reconfiguration services add new application parts without changing the functional behavior of the control application.
- ***Reconfiguration sequence (RECONF):*** The current system state is still operated, but all new elements of the new system state are already available. Based on the classification from Walsh *et al.* (2007b)—see Section 3.4.1 and especially Section 4.2.2—no

structural changes are necessary within the reconfiguration sequence. Only topological changes (creation or deletion of connections as well as writing of parameters) as well as internal changes (writing of internal variables) are possible apart from the execution control services for starting and stopping FB instances.

- ***Deinitialization sequence (RDINIT) and deletion of the ECA:*** These sequences belong to the post-processing of a system evolution step and are used to delete those parts which will not be used any more in the new system state. The functional behavior of the control application will not be changed.

Command	Object	Download ECA	RINIT	RECONF	RDINIT	Deletion ECA
CREATE	Resource / FB	Yes	Yes	—	—	—
	Connection	Yes	Yes	Yes	—	—
DELETE	Resource / FB	—	—	—	Yes	Yes
	Connection	—	—	Yes	Yes	Yes
WRITE	Parameter	Yes	Yes	Yes	—	—
	Internal variable	Perhaps	Perhaps	Yes	—	—
READ	Input/output variable	—	—	Yes	—	—
	Internal variable	—	—	Yes	—	—
START	FB	Yes	Yes	Yes	—	—
STOP	FB	—	—	Yes	Yes	Yes
KILL	FB	—	—	—	Yes	Yes
RESET	FB	—	—	—	—	—
QUERY	Anything	Yes	Yes	—	Yes	Yes

Table 4: Basic reconfiguration services within the different execution phases of a system evolution step

According to this limited set of basic reconfiguration services, which neglects any structural changes to the system model, an incorporation of these changes to the formal model can be applied by using the available means of the modeling language. Four different classes of changes to the system model have to be modeled:

- Manipulation of connections
- Execution control of FB instances
- Reading of input/output variables as well as internal variables
- Writing of input and internal variables

We will provide a detailed description of the modeling approach for these kinds of changes in the system model during execution based on the modeling language NCES.

7.3.1 Manipulation of connections

We have to distinguish two different aspects when a basic reconfiguration service manipulates a connection. On the one hand there are two types of connections, event and data connections, which will be modeled in different ways according to their functionality within the IEC 61499 standard. On the other hand there are two different management commands available for the manipulation of any connection: the CREATE and the DELETE management command. In the following discussion we will provide models for the two different types of connections (for data and events) that provide the possibility to create and delete the connection.

Event connections

An event connection is used to trigger the execution of FBs in the IEC 61499 standard. Based on the execution semantics of R^{3}E described in Section 6.1.2 the the issue of of an output event means that the connected input events will be put into the queue within the event dispatcher. This behavior can be mapped to the formalisms of NCES by using event arcs, which model the execution flow within the runtime environment. Based on the schematic of the execution behavior of BFBs in Figure 23 an output event will be put into the event dispatcher as part of an action. Afterwards, the execution within the BFB has to be continued. The corresponding NCES interface of an FB will consist of an output event in order to put the connected input event into the event dispatcher and an input event as notification that the execution flow within the BFB can be continued.

An appropriate NCES model for the event connection is depicted in Figure 26 incorporating the possibility to "create" and "delete" the event connection. The input event 'IN' receives an event if the BFB executes the sending of an output event. Based on the internal state of the event connection (represented by places 'p_1' and 'p_2'), two different paths are available in 'ManagedEventConnection'. If the event connection is enabled (the event connection has been created), the output event 'Trigger' will be issued, which can be used to put the corresponding input event into the event dispatcher. After a confirmation via the event input 'Confirm' the output event 'OUT' is triggered and the execution flow within the BFB will be continued. But if the event connection is disabled (the event connection has been deleted), nothing else will happen except that the output event 'OUT' is triggered. In terms of IEC 61499 this means that the event connection does not exist, because no entry within the event dispatcher is added.

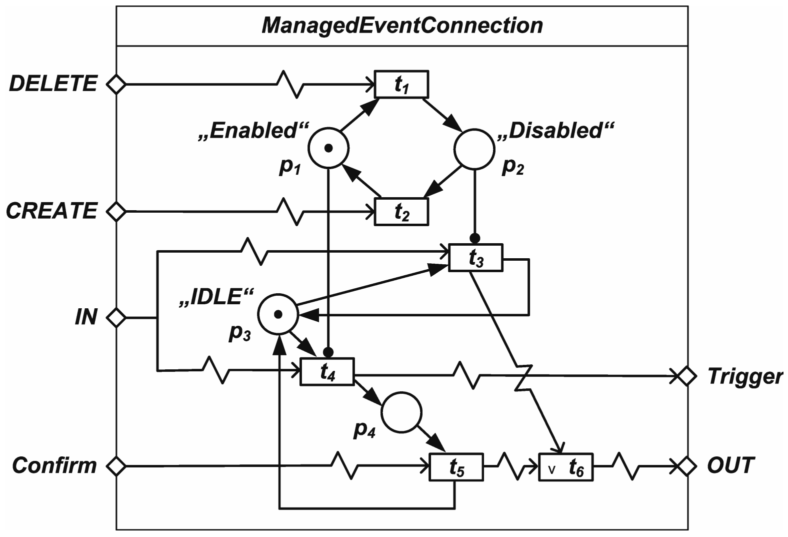


Figure 26: Formal NCES model of a managed event connection[19]

The creation and deletion of the event connection is triggered by the input events 'CREATE' and 'DELETE', which are issued by the basic reconfiguration services within the ECA. The model in Figure 26 describes an event connection that is initially created. By changing the initial marking from 'p_1' to 'p_2' an initially deleted event connection can be modeled.

Data connections

The behavior of data connections is different to event connections. Based on the implementation of R^{3}E a data connection includes a storage element. As soon as an output event occurs

[19] The condition input 'enable' is neglected for the sake of clarity. As this module will be used within the model of a task, each condition has to be connected to the 'enable' condition input, too.

which is related to the data output via the WITH construct, the storage element of the data connection is set according to the data output of the FB. If several data connections exist with the same source (several connections from the same data output to several data inputs) only one storage element will be used for all these data connections.

The formal model of a manageable data connection is based on a similar concept as for event connections (see Figure 27 for a data connection for Boolean variables). The internal state machine (places 'p_1' and 'p_2') defines the current state of the data connection, whether it exists or not. This state is influenced by the corresponding basic reconfiguration service within the ECA. The internal behavior is different to an event connection, as the internal storage of the data connection has to be set as soon as the execution flow enters the data connection via the 'IN' event input (according to the WITH construct this will happen during the sending of an output event). The internal storage for Boolean variables is represented by places 'p_5' and 'p_6'. As this operation will take a certain amount of time, a time delay of length 'x' is introduced at place 'p_4'. If the data connection does not exist (place 'p_2' is marked), no data sampling with time consumption will happen. In this case only the output event 'OUT' is issued, which means in terms of IEC 61499 that the data connection does not exist.

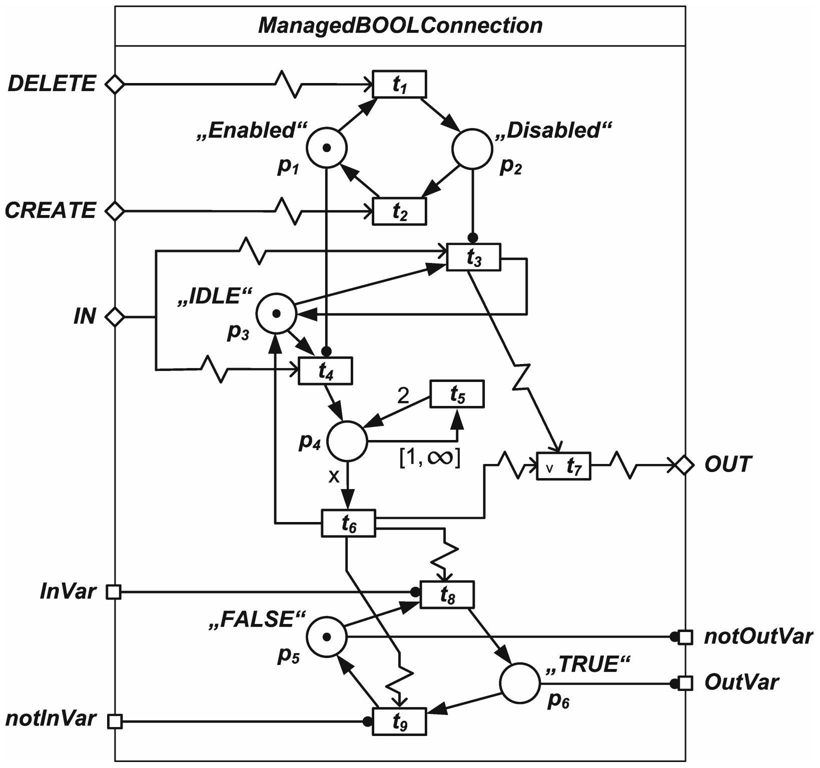


Figure 27: Formal NCES model of a managed data connection (Boolean type)[19]

The time delay modeled in Figure 27 depends on two conditions. On the one hand it will vary according to the data type which has to be stored. For each different data type the specific time value has to be measured. On the other hand this time will only occur if the data connection is the first one which has been established for a certain data output. As stated above, the implementation of the R^3E uses only one storage element, although several data connections exist with the same source. Accordingly only for one data connection the execution time has to be mentioned, for all other data connections with the same source 'x' can be set to zero.

The initial state of the data connection (available or not) can be modeled via the initial marking of places 'p_1'and 'p_2', as already mentioned for event connections.

7.3.2 Execution control of FB instances

The representation of the management commands START and STOP for FB instances can be summarized as execution control. The concept is similar to the one used for the manipulation of connections. Based on a state machine the execution flow within the FB instance may trigger the operation of the FB (in terms of NCES models) or it will be ignored and handed over to the next element in the execution flow. As basis for the internal state machine the operational state machine for managed FBs (see Figure 63) has to be used. As we will not consider the management commands KILL and RESET, a simplified state machine is modeled by the places ‘p_1’ to ‘p_3’ and transitions ‘t_1’ to ‘t_3’ in Figure 28. As simplified representation of the different triggers for the execution of an FB instance—this will be the request by the occurrence of an input event, see Section 7.4.3—we use only one input event ‘IN’. Based on the state of the managed FB instance, the execution of the internals of the FB will be triggered (denoted by the dotted rectangle). Or the execution flow will be passed over to the next element within the system architecture by the issue of the output event ‘OUT’. The state of the FB instance will be influenced via the input event ‘START’ and ‘STOP’, which represent the triggers sent from the corresponding basic reconfiguration services within the ECA. The current state of the FB instance is additionally available via the condition outputs ‘FBSidle’, ‘FBSrunning’, and ‘FBSstopped’.

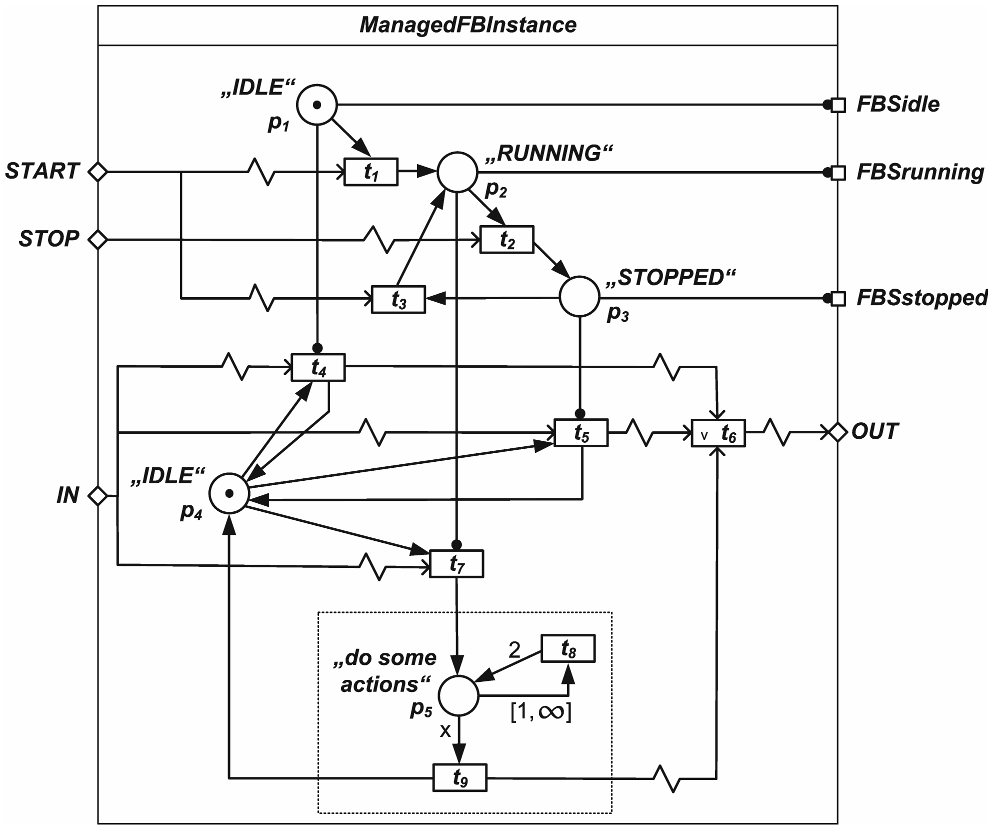


Figure 28: Formal NCES model of a managed FB instance[19]

This simple model also incorporates the correct behavior in the case of stopping the FB instance during its operation (the ECA may preempt the execution exactly during operation of the FB instance). According to IEC 61499-1 (2005, Table 9) the currently active execution of the BFB will be completed without the generation of output events. If the execution control is passed back to the FB instance after being stopped, the execution flow will continue the execution of the FB and the algorithms will be finished. For the the issue of of output events an appropriate condition for sending the output events based on place ‘p_2’ has to be incorporated in the model of the FB (not included in Figure 28).

Based on a similar concept also the CREATE and DELETE management commands for FBs may be incorporated in the formal NCES models by using an enhanced state machine of the FB instance. Only the KILL management command is problematic, as the operations of the FB instance have to be stopped immediately. If this happens, the execution flow within the formal NCES model of the control device will be stopped, too, which does not model the correct behavior of the implementation.

7.3.3 Reading of input/output variables as well as internal variables

The READ management command can be modeled without any additional effort in the NCES models. Any variable is represented by a set of places within the formal model, as for instance depicted for the storage element of a data connection in Figure 27 (places 'p_5' and 'p_6'). The corresponding basic reconfiguration service within the ECA has to be connected to these places via condition arcs. As soon as the basic reconfiguration service is executed it will gather the current value of the variable. The necessary execution time for reading the variable has to be incorporated within the basic reconfiguration service.

In principle any data type may be modeled by using places representing Boolean values. But the effort within the formal models grows to a high extend already for integer variables, as a 16-bit variable has to be represented by 32 places in the model. Consequently a read command (or any data connection, too) will consist of 32 condition arcs. The situation will be even worse for structured data types or character strings.

7.3.4 Writing of input and internal variables

The influence of the WRITE management command has to be modeled in a similar way as the assignment of a value to a variable, which is used within any kind of storage element. The only difference is that there may be several sources of values for a variable, but this may be necessary for the formal model of an algorithm, too. The NCES model depicted in Figure 29 may be used to model also other aspects than only the influence of the WRITE management command to a variable. In general the assignment of a value to a storage element from different sources is represented by this NCES model.

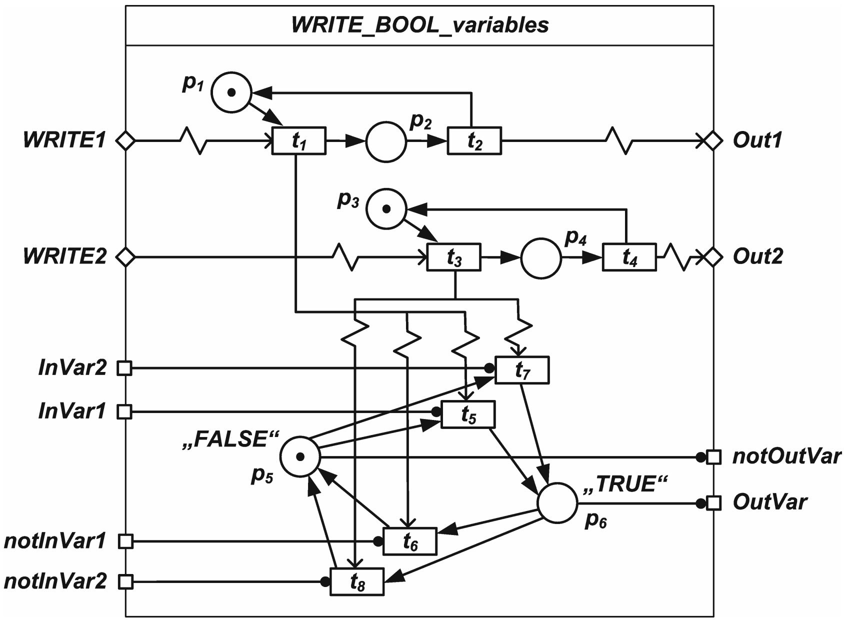


Figure 29: Formal NCES model of WRITE for two different values[19]

The concrete example of Figure 29 provides the possibility to write two different values to a storage element (again of Boolean type) by using different execution flows. These are depicted by the interface 'WRITE1' and 'Out1' as well as 'WRITE2' and 'Out2'. The different values of the WRITE management commands is provided via the condition inputs ‘InVar1’ and ‘notInVar1’ as well as ‘InVar2’ and ‘notInVar2’. If one of the two input events is triggered, the value depicted by the related condition inputs is assigned in the storage element. If the corresponding basic reconfiguration service is triggered, the new value for the variable has to be issued to the model given in Figure 29 by using condition arcs. The execution time for writing the value of the variable may be included also in the model in the same way as depicted for data connections (see Figure 27). But it may be incorporated also in the execution time of the basic reconfiguration service, as it is assumed in Figure 29.

7.4 Modeling architectural elements in NCES

Based on the representation of real-time behavior and the influence of basic reconfiguration services to the system model the overall formal model for a given control device can be designed. The following sections provide an overview on modeling approaches for a specific control device, which will be used for demonstration also in Chapter 8. As main architectural elements the *Embedded Configurable Operating System* (eCos) as described in Massa (2003) (see also Appendix C) and R^{3}E (see Appendix B) are used in the demonstration control device. The overall system model emerges by the composition of the different parts based on the given configuration, which depends apart from the main elements operating system and runtime environment especially on the control application and the ECA. According to the parameters of these applications the system model has to be configured, which will include also structural effects on the NCES model. Another important parameter is the execution time of the different single units within the system model, which have to be adapted according to the given system architecture.

Modeling language

Before we will go into detail about the modeling of the architectural elements, a short analysis of the proposed modeling language NCES should be provided. Many different modeling languages exist with specific features and modeling opportunities. The modeling language NCES is described in detail in Appendix C. The most important features for the evaluation of DSE are:

- ***Modular modeling:*** The overall system architecture can be established and configured in terms of modules in a hierarchical manner. Based on simple functionalities more complex elements can be created by composition. The configuration of a control device is a composition of different parts by itself.
- ***Execution flow via events:*** The incorporation of real-time behavior is based on the fragmentation of the overall source code and attaching execution time to these parts. The overall system behavior is represented according to the execution flow within the system model. The modeling elements events and event connections provide a powerful means for the modeling of the execution flow in NCES.
- ***Preemption:*** A real-time operating system is characterized by the scheduling algorithm in order to achieve real-time constraints for the different tasks during interaction with the environment mainly based on interrupts. A modeling language has to provide the possibility to model preemption as consequence of these characteristics. Especially in combination with the modeling of execution time preemption has to be supported (an appropriate approach utilizing NCES is described in Section 7.2).[20]

[20] A popular approach for the modeling of real-time systems is timed automata. According to Waszniowski and Hanzálek (2003) timed automata is not suited for modeling preemption. Stanica (2005) uses timed automata in order to model the behavior of IEC 61499 applications. His resource model is very simple with the only constraint that only one algorithm may be executed at the same time. The temporal behavior of algorithms is

But there are also some limitations based on the choice for NCES as modeling language. The main restriction is that no continuous time may be used. NCES provides only means for discrete models of time. According to the chosen smallest time unit a more or less accurate model of the system will be available. A very small time unit additionally increases the state explosion problem. A more detailed analysis about the comparison of discrete and continuous time model checking is given in Clarke *et al.* (1999, Section 17). This situation may not be problematic within a single control device, but based on a closed-loop modeling (the plant is a continuous time system by nature) as well as the cooperation of several control devices within the same control application (in general no synchronization of execution exist within different control devices) a discrete time model may restrict the expressiveness of the system model. A second drawback concerns the modeling of calculations with values aside from Boolean variables. Any calculation may be modeled by Boolean relations, but this is no appropriate means for the efficient modeling of calculation with non-Boolean variables, especially algorithms in BFBs.

We will provide a rough overview of the main modeling approaches for the elements real-time operating system, real-time reconfiguration runtime environment, control applications and evolution control applications as depicted in Figure 24. In addition a general behavior model for different elements such as additional tasks, other applications or the communication network will be presented in Section 7.5.

7.4.1 Real-time operating system (eCos)

The RTOS provides the basis for the execution of different tasks within the control device and the interaction between the tasks and the environment (based on interrupts). As a concrete example the eCos real-time operating system [9] will be considered, which is described in detail in Massa (2003) as well as in Appendix C. The models presented in this section are based on the master thesis of Gosetti (2007), which has been conducted under supervision of the author.

The main element of an RTOS is the scheduler, which provides the execution order of different tasks based on a specific algorithm. eCos includes two different scheduling algorithms, which both provide *32* priorities with task preemption. In case of the bitmap scheduler one task can be handled per priority level. As soon as a task with a higher priority (the highest priority is *0*) than the currently executed task wants to become active, a task switch is performed in order to execute the task with the highest priority. The second scheduling algorithm is called *Multilevel Queue* (MLQ) scheduler and supports several tasks per priority level. The active tasks within the same priority level are included in a queue, and based on a time parameter the execution is switched between these tasks in a round robin procedure. Again only the tasks which concern the highest active priority level will be executed.

RTOS configuration

In order to provide a component-based formal model of the operating system and its configuration the different tasks as well as the scheduler of the RTOS will be considered as separate NCES modules. Figure 30 depicts such an RTOS configuration with two priority levels, whereupon the MLQ scheduler is used because there are two tasks related to priority level *1*. In order to model the execution behavior of these tasks the following interface is utilized between the scheduler and a task (we will use the task perspective for explanation):

- ***Notification of task activation ('Wakeup'):*** If a task wants to be come active, the output event 'Wakeup' will be issued by the task. This may happen based on an external interrupt, e.g., the timer or the network interface.

---

characterized by their execution time. But there are no concepts included in order to provide a more detailed description of the temporal behavior of the overall system architecture, especially no preemption.

- ***Notification of task suspension ('Suspend'):*** If a task does not need to be executed any more, the output event 'Suspend' will be issued by the task. A typical example is the execution of event chains: as soon as there are no more events in the event dispatcher, the task corresponding to the event chain will suspend.
- ***Assigning execution control to a task ('enable' and 'stopped'):*** As already described in Section 7.2 preemption of the control flow within NCES models can be realized by using an enabling condition input. The scheduler will use the 'enable' condition input in order to assign the execution control to the task that should be executed. The condition input 'stopped' is inverse to 'enable'.

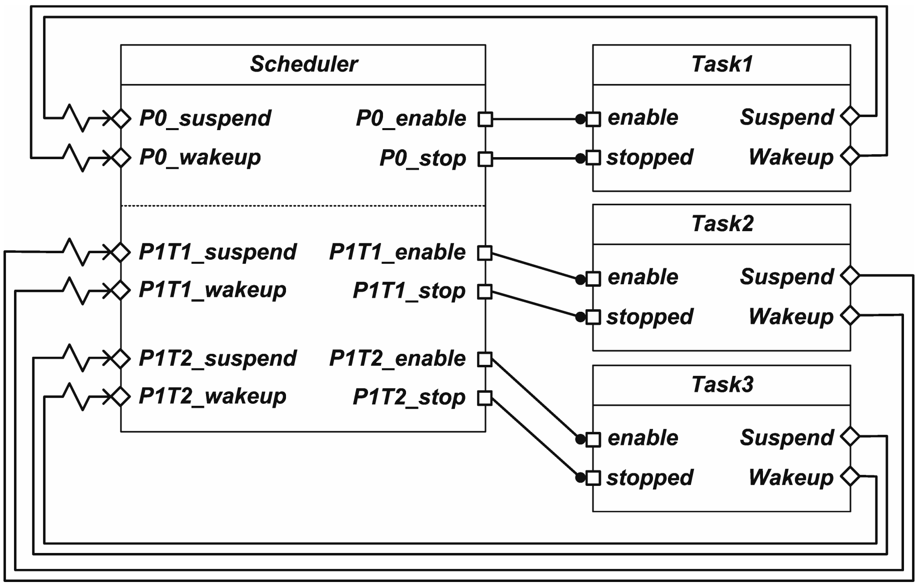


Figure 30: RTOS configuration with three tasks and two priority levels

The NCES model of the scheduler ('Scheduler') provides this interface for each task within the configuration of the control device. Based on the scheduling algorithm, the task with the highest priority is executed by setting the input condition 'enable' to true. There is only one task executed at the same time. If the scheduler has to perform an action (e.g., switching the task context), only the module 'Scheduler' will be active.

Model of the scheduler

The formal model of the scheduler is split up into several components again. In detail each priority level is handled by a distinct module, Figure 31 depicts the internal model of the element 'Scheduler' mentioned before in Figure 30. The principal idea for modeling the scheduling algorithm is that each priority level has information about the upper priority levels as well as the task related to its own priority level. The module 'IDLE' represents the situation that there is no active task within the system configuration. Initially 'IDLE' is active within 'Scheduler'. As soon as one priority level receives a 'Wakeup' event, the corresponding module becomes active within the scheduler. It may be interrupted by a higher priority, or the corresponding task suspends. In the second case the next lower priority level will become active. If there are no more tasks ready for execution, the execution flow will reach 'IDLE' again.

The behavior of the NCES module 'Scheduler' outlined above results in an interface between the different priority level modules which consists of the following elements:

- ***Execution flow ('activate', 'nextPx'):*** A priority level receives the control for the execution of its related tasks via the event input 'activate'. If there is no task ready for execution, the next lower priority will be activated via the event output 'nextPx'. If any higher priority level wants to execute its tasks, the corresponding event output 'nextPx' will be triggered. A priority level has event outputs for every higher priority level as well as the next lower priority level (simply denoted by 'x' in this description).

- ***Current state of priority levels ('Px_suspend', 'Px_wakeup'):*** Based on the internal modeling of the different priority levels a condition for the suspension of the priority level ('Px_suspend') and a condition about the availability of tasks for execution ('Px_wakeup') are sufficient to represent the current state of priority levels. In order to provide each priority the necessary information on the current state of the higher priority levels as well as the next lower priority level, appropriate condition inputs are used in the modules of the priority level.

In addition each priority level provides the interface to the tasks which are related to this priority level, as described above.

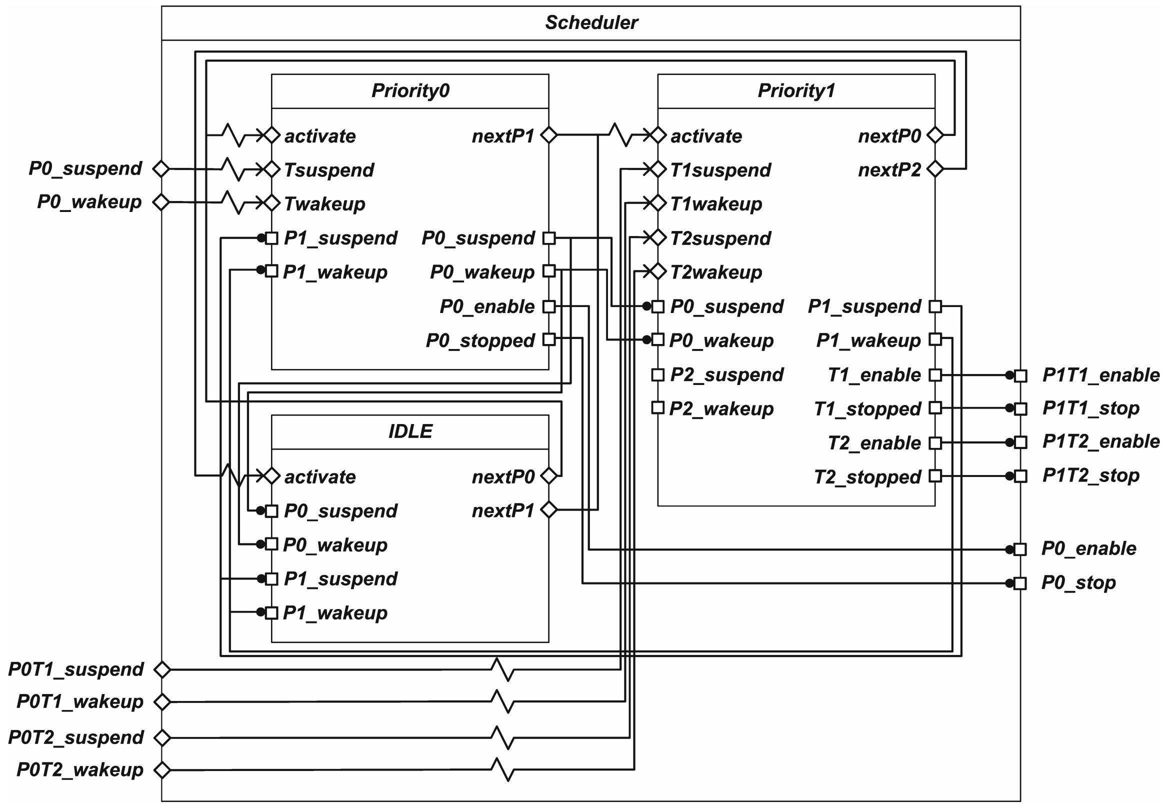


Figure 31: Internal model of the module 'Scheduler' mentioned in Figure 30

Interrupt handling

The eCos operating system uses so-called callback functions in order to handle interrupts. As soon as an interrupt occurs, the scheduler switches to the callback function, which includes the functionality related to this interrupt (application dependent source code). In order to model the interrupt handling the model of the scheduler can be used. A callback function is represented as a task on a priority level higher than *0*. If an interrupt occurs, the output event 'Wakeup' will be triggered.

Real-time behavior

The real-time behavior of the RTOS is mainly characterized by the scheduling of tasks. This is given by three different times (quantitative measurements are documented in Section 8.1.1):

- ***Task switching time:*** The time necessary to switch from one active task to another one. This time may vary according to the amount of data that has to be saved for switching the context.

- ***Task suspension and resumption:*** The transition between a priority level and the 'IDLE' state are characterized by separate time parameters.
- ***Task creation and deletion:*** The approach for the evaluation of DSE does not need to model these aspects within the reconfiguration sequence. There will be a fixed set of tasks within the operation system.

The different time parameters have to be incorporated in the model of the scheduler. For instance the information about active tasks in the next lower priority level is used in order to decide if the execution time of a task switch has to be included when a certain priority level has no more active tasks.

7.4.2 Real-time reconfiguration runtime environment

The execution behavior of the R^{3}E has been described in the context of the calculation of execution times for event chains in Section 6.1.2 as well as in Appendix B. In order to provide a model of the behavior of the R^{3}E the event propagation mechanism by the use of the event dispatcher as well as the separation of different event chains has to be taken into consideration. Based on the description above of the representation of different tasks within the RTOS (see Section 7.4.1) the different event chains have to be modeled within the different tasks. Therefore we will investigate only the formal model of a single event chain.

FB interface

The basis for the transformation of an IEC 61499 FB into a NCES module is the representation of the FB interface within NCES. Both concepts (IEC 61499 and NCES) provide events as means for the modeling of execution flow, but the NCES module has to take into consideration the implementation method, too. Figure 32 depicts the IEC 61499 FB as an example in both ways, a) as an IEC 61499 FB and b) as the corresponding NCES module.

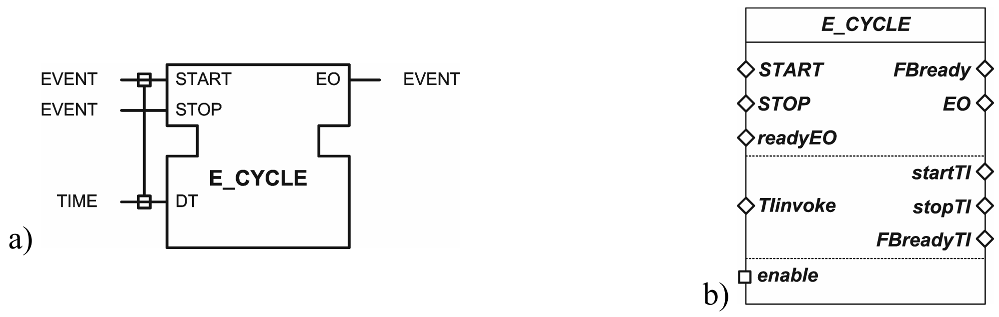


Figure 32: Representation of FB E_CYCLE in a) IEC 61499 and b) NCES

Each IEC 61499 input event is directly mapped to NCES because it represents the call of the FB and directly matches with the execution flow within R^{3}E. The end of such an call is depicted by the event output 'FBready', which signals that the execution control moves to the next element. An IEC 61499 output event is visible in the NCES module interface, too. But in addition to each output event an input event has to be incorporated, which signals that the sending of the output event has finished and execution flow comes back to the model of the FB (e.g., the output event 'EO' and the input event 'readyEO' relate to each other). Any data inputs and outputs may be represented by condition inputs and outputs. In the example given in Figure 32 the data input 'DT' is not visible in the NCES interface, because this value parameterizes the timing service of the runtime environment and will be used directly in the callback function of the timer. But instead an interface for the interaction between the timing service and the E_CYCLE FB is included in the NCES module. The timing service can be started ('startTI') or stopped ('stopTI'). The NCES module includes an additional event input 'TIinvoke' as trigger from the timer as well as a confirmation ('FBreadyTI') in order to move the execution flow back after an interruption from the timer (see next paragraphs for more details).

Event propagation within an event chain

Each real-time constrained event chain is executed in a separate task with a separate event dispatcher. All unconstrained event chains within a resource are executed in one task with a common event dispatcher. The concept and models for event propagation are the same in both cases. Figure 33a depicts a simple FB network which will be used to describe the transformation of an IEC 61499 control application into a formal model. The corresponding NCES model is given in Figure 33b, where each FB is represented by its NCES interface as mentioned above. The basic idea for the transformation is that each event input within the FB network is modeled by a number, e.g. the 'REQ' event input of 'simpleFB1' has number two ('EV2' in Figure 33a). By using this number the event dispatcher is able to distinguish between different events that are inserted ('inEVx' and confirmation via 'readyEVx') and fetched for calling of the corresponding NCES module ('outEVx' with feedback of execution control via 'endEVx'), where 'x' stands for the number of the input event. An event connection is modeled by a module similar to the one depicted already in Figure 26 and a data connection according to Figure 27 (both without the capability of dynamic reconfiguration).

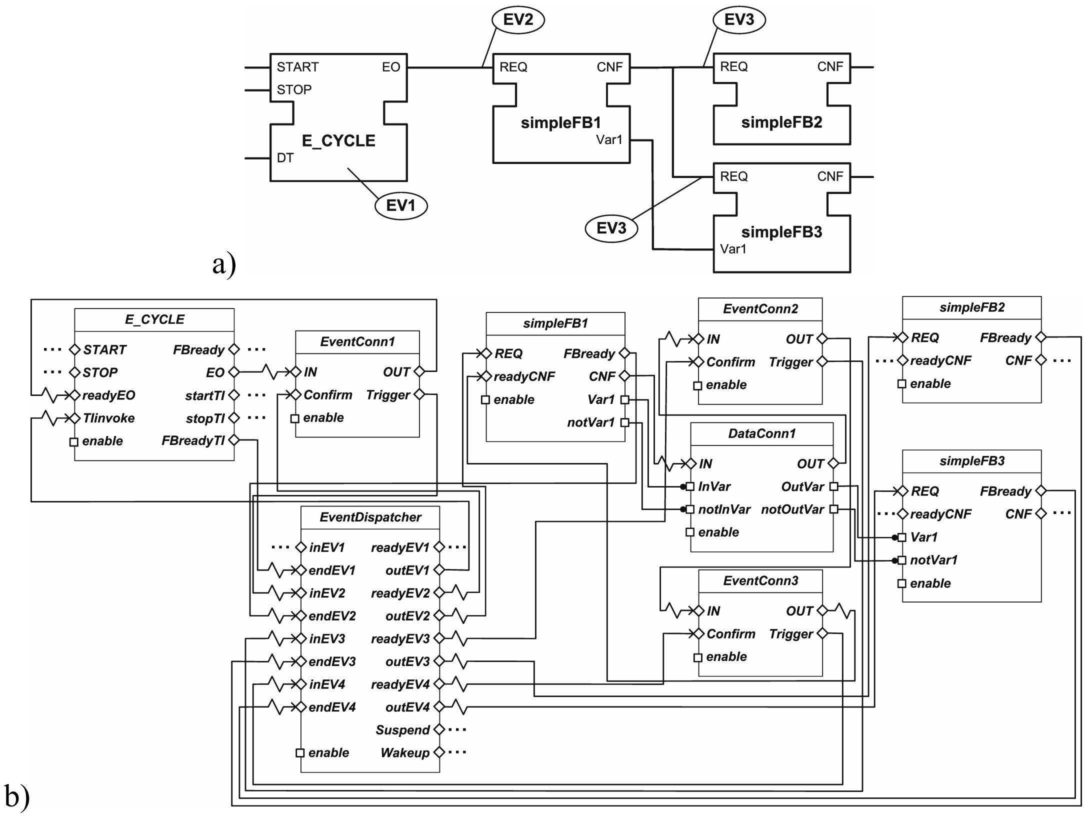


Figure 33: Representation of an FB network in a) IEC 61499 and b) NCES[19]

The introduction of the first event (which is the identification of the event source 'E_CYCLE', which again has a unique number) will be described in the next section. As soon as this event is put into the event dispatcher, the corresponding task will issue for execution time (output event 'Suspend') and then call 'E_CYCLE' via the 'TIinvoke' input event. 'E_CYCLE' sends the output event 'EO' which triggers the module 'EventConn1' representing the event connection between 'EO' and 'REQ'. The input event 'REQ' is put into the event dispatcher via the event input 'inEV2'. After finishing the execution of 'E_CYCLE', the next event will be fetched from the event dispatcher ('outEV2'). The overall FB network is executed based on this basic modeling approach. The FB network presented in Figure 26 depicts only a sector within the overall task model. The open interface to the rest of the task model is denoted by '…'.

Critical sections within R^3E

One main challenge in real-time computer systems is the assignment of priorities and tasks in order to avoid deadlocks. Different concepts exist for protection of such critical sections (see also Appendix C) within an RTOS. One of the most important critical sections within R^3E is the event dispatcher. On the one hand each event input within the task will be put into the event dispatcher. But on the other hand also the identification of an event source has to be put into the event dispatcher. If the execution of an event chain is interrupted by the timer exactly when an input event is put into the event dispatcher, the timer will not be able to use the event dispatcher, too. In order to avoid a deadlock in this situation, R^3E uses a mutual exclusion methodology with priority inversion.

The NCES model for the protection of this critical section is depicted in Figure 34. The modules 'RegIn' and 'RegOut' are used to capsule the critical region 'EventDispatcher'. The module 'Semaphore'[21] simply provides a storage element that is triggered as soon as the execution flow enters or leaves the 'EventDispatcher'. If an external event source (e.g., the timer) wants to add an identification of an event source to the event dispatcher the input event 'TRIG' occurs. Before the issue of this request to the event dispatcher via 'Insert', the availability of the critical resource 'EventDispatcher' is checked. If it is currently used, the *priority inversion protocol* will be applied: The event dispatcher will be executed ('enableEV' set to true) as long as 'EventDispatcher' is free again, then the request from the external event source will be handled.

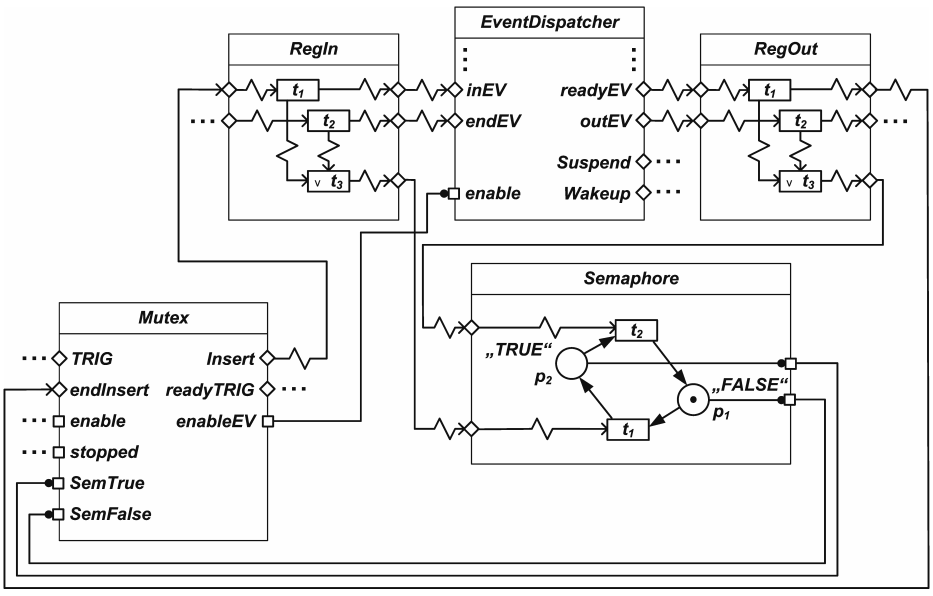


Figure 34: Representation of mutual exclusion for the event dispatcher

Figure 34 provides again only a sector of the overall model of the FB network. For instance, the confirmation of the insertion to the external event source is given by the event output 'readyTRIG'. In the model of the overall RTOS configuration in Figure 30 appropriate connections between the task representing the timer callback function and the event chain have to be added. Concrete measurements for the real-time behavior at the application level are presented in Section 8.1.1.

[21] A means for the synchronization of tasks within an RTOS called semaphores exists, too. The NCES module 'Semaphore' may also be used for this purpose, too. But within the application depicted in Figure 34 it is part of the mutual exclusion model.

7.4.3 IEC 61499 applications

The above given description of event chain modeling provides one part of IEC 61499 applications. The second part belongs to the internal formal models of FBs, which will be discussed in this section. Three different types of FBs exist within IEC 61499, whereupon we will focus on the BFB. The CFBs can be modeled based on the above given description of an event chain (the same event dispatcher as for the overall application will be used for a CFB), and the SIFBs are not described by means of IEC 61499. For SIFBs the implementation details need to be taken into consideration (see also Section 7.5).

Figure 35a depicts a very simple BFB example, which consists of an ECC with two states. If the event input ‘REQ’ is issued to the FB in its initial state ‘State1’, the ECC switches to ‘State2’, executes an algorithm ‘Alg.’ and sends the output event ‘CNF’. The ECC will only go back to ‘State1’ if the data input ‘DI’ is true.

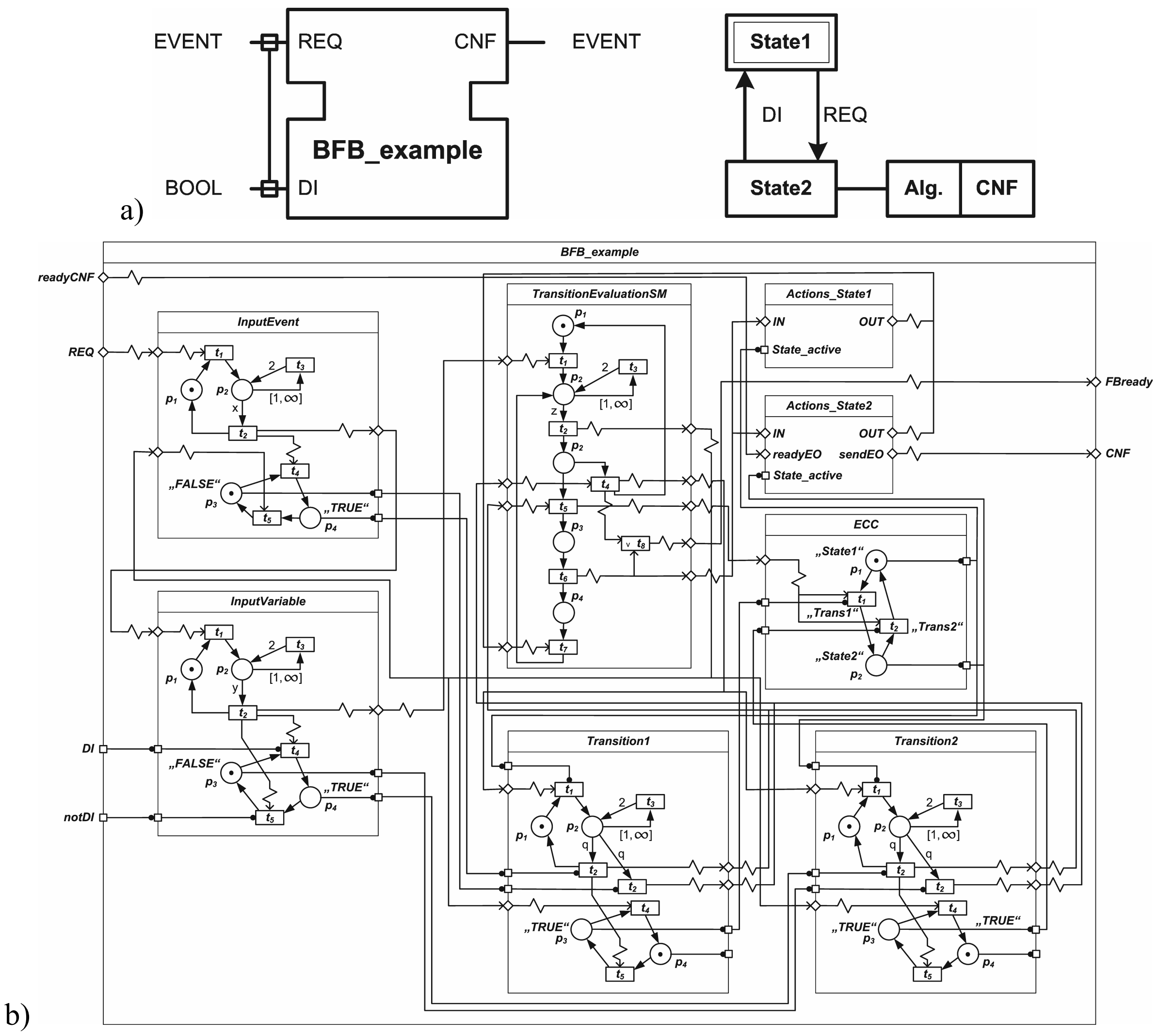


Figure 35: Representation of a BFB in a) IEC 61499 and b) NCES[19]

The corresponding NCES model is given in Figure 35b. The execution flow within the formal model is similar to the one given in Figure 23. If the BFB is called by an input event, at first the input event is registered to an internal storage (‘InputEvent’) and the corresponding data input is sampled (‘DataInput’). Both actions take some time, denoted by the time delays ‘x’ and ‘y’. The ECC is evaluated according to the procedure modeled in ‘TransitionEvaluationSM’. After a certain time delay ‘z’, representing a constant time for evaluation, the different transition conditions available within the currently active ECC state are evaluated.

The ECC is represented in the module 'ECC'. Herein, only 'Transition1' will be triggered as 'State1' is initially true. The evaluation result will be offered in two different ways. On the one hand a storage element is used to represent the evaluation result. On the other hand, an output event (the 'Transition1' module issues different output events according to the result of the evaluation) is used to model the execution flow. As each evaluation will take some time, a representative time delay 'q' is included in 'Transition1' and 'Transition2'. In case of a negative evaluation, the next transition will be triggered (not visible in Figure 35b). If the evaluation was positive and a transition clears, the state in module 'ECC' will be changed and the actions within the currently active state ('Actions_State2' in this case) will be triggered. This includes also the sending of output events ('event output 'CNF'), whereas the insertion of the connected input events is represented in the event chain model as depicted in Figure 33). Afterwards the transitions of the new active state are evaluated again. In case of a negative evaluation, the event output 'FBready' is triggered, as no more actions have to be performed within the BFB model. The sampling of data outputs, which do not concern any event output, has to be represented within the event chain model (directly connected to the event output 'FBready').

7.4.4 Evolution control application

An ECA is similar to any control application and can be modeled according to the concepts presented in Section 7.4.2. The only difference is the use of special FBs, the basic reconfigureation services. The formal model of a basic reconfiguration service is similar to the model of any FB, as depicted in Section 7.4.3. The only difference is that a special interface is added according to the IEC 61499 management command incorporated by the basic reconfiguration service. The interfaces for the relevant management commands within the reconfiguration sequence have been described already in Section 7.3:

- ***Manipulation of connections:*** CREATE or DELETE management commands are executed in the NCES model by a single event.
- ***Execution control of FB instances:*** The operational state machine has to be added to the NCES model according to Section 7.4.3. The START or STOP management commands are executed in the NCES model by a single event.
- ***Reading of input/output variables as well as internal variables:*** The necessary data is available within storage elements in the NCES model. The basic reconfiguration services can access this data via condition arcs.
- ***Writing of input and internal variables:*** The basic reconfiguration service has to provide the data via condition arcs. The WRITE management command is executed in the NCES model by a single event.

7.5 Interrelation with the system environment

The description given above of formal models represent the core part for the evaluation of DSE. The control application and the ECA are in the center of interest, but it is necessary to include also the IEC 61499 runtime environment, the real-time operating system, and the hardware (in terms of execution time for source code regions) to the formal description. But according to the evaluation framework for DSE presented in Section 5.1.2 and especially the architectural elements within a control device depicted in Figure 24 additional elements have to be incorporated to the formal models. We summarize these elements by the term system environment, because they are not directly involved in a system evolution step but do provide important effects to the overall system model:

- ***Communication network:*** If there are control devices which are not part of the control application (which will be adapted during operation) the behavior of the communication network can be used to integrate a brief description of the interaction with any other control device. The incentives from the communication network may be used within any task of the control device, especially in any other IEC 61499 application.

- ***Additional tasks:*** Any task within the control device influences the execution behavior of the control device at least due to the consumption of execution time according to the scheduling algorithm and its own execution characteristics.
- ***Other applications:*** Any IEC 61499 application within the control device has to be represented in the formal model, although it does not affect the functional behavior of the control application. According to the mechanisms of R^3E other applications are incorporated as tasks within the RTOS and therefore may be treated similar to additional tasks.

A detailed description of all elements within the system environment will possibly cause unacceptable effort in the engineering process. But for the evaluation of the evolution specification properties it is essential to include also these side effects within the DSE in order to provide significant results. As a possible solution, a very abstract behavior of these elements may be used as an approximation. In any case it has to be evaluated if this abstract behavior provides an over- or under-estimation of the real behavior of the element.

General behavior description

A general description of an element within the control device may be based on the occurrence models of external events presented in Section 6.1.1. For the formal model these occurrences may be used as triggers for some calculations within the different tasks of the RTOS or directly as triggers from the communication system. In addition to the pure occurrence of an event also the execution time of the element, e.g., a typical time necessary for a control application, has to be included into the formal model. Herein the possibility of conflicts within the formal model can be incorporated for the description of different execution times. A conflict in NCES occurs for instance if a place is connected to several conditions via flow arcs and there are more conditions enabled as markings are available within this place. In the reachability graph of the system each possible combination of transitions that may clear based on this situation will be incorporated. Accordingly different execution times of a given element are included in the formal model.

We will consider different occurrence models for external events represented as NCES model in order to give some examples. These modules may be used within the formal model of the elements to describe an abstract representation of the element's behavior. The first occurrence model is the periodic occurrence model. The occurrence of events is characterized only by the cycle time T_P without taking into consideration any disturbances of this cycle. Figure 36 depicts the NCES model of a periodic occurrence model for the cycle time $T_P = 5$.

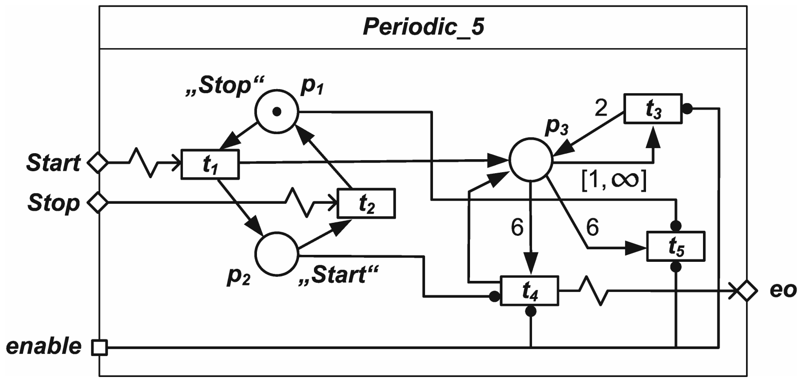


Figure 36: Representation of periodic event occurrences (5 time units)

The module may be started and stopped by the input events 'Start' and 'Stop'. As soon as the module is started, the internal clock in place 'p_3' modeled by markings starts counting and issues an event output 'eo' every *5* time units. Transition 't_5' is used to remove all markings from place 'p_3' in order to stop the emitting of output events.

A more complicated model is the periodic occurrence of events with jitter. Next to the precise cycle time T_P an event may occur already before or after the cycle time, bounded by the jitter J. Figure 37 represents a model with cycle time $T_P = 5$, whereupon a jitter J of one time unit is incorporated, too. The module may be stopped and started as described already above. Based on the time resolution T_{res} and the jitter time J, different paths are possible. The 'Start' event will add a marking to place 'p_3', which represents the point in time exactly at the beginning of the possible time window for emitting events. Accordingly an event may be issued immediately (path over transition 't3'), or every next time unit until the end of the time window (two times the jitter time J) is reached. In Figure 37 three different paths are possible, because the jitter time and the time resolution are equal and set to one ($J = T_{res} = 1$). In order to reach again the same point in time represented by place 'p_3' in all possible paths, an additional time delay after the issue of an output event 'eo' has to be added in each path. The reachability graph of 'Periodic_5_Jitter_1' includes all possible occurrences of output events 'eo' according to the parameters cycle time T_P, jitter time J, and time resolution T_{res}.

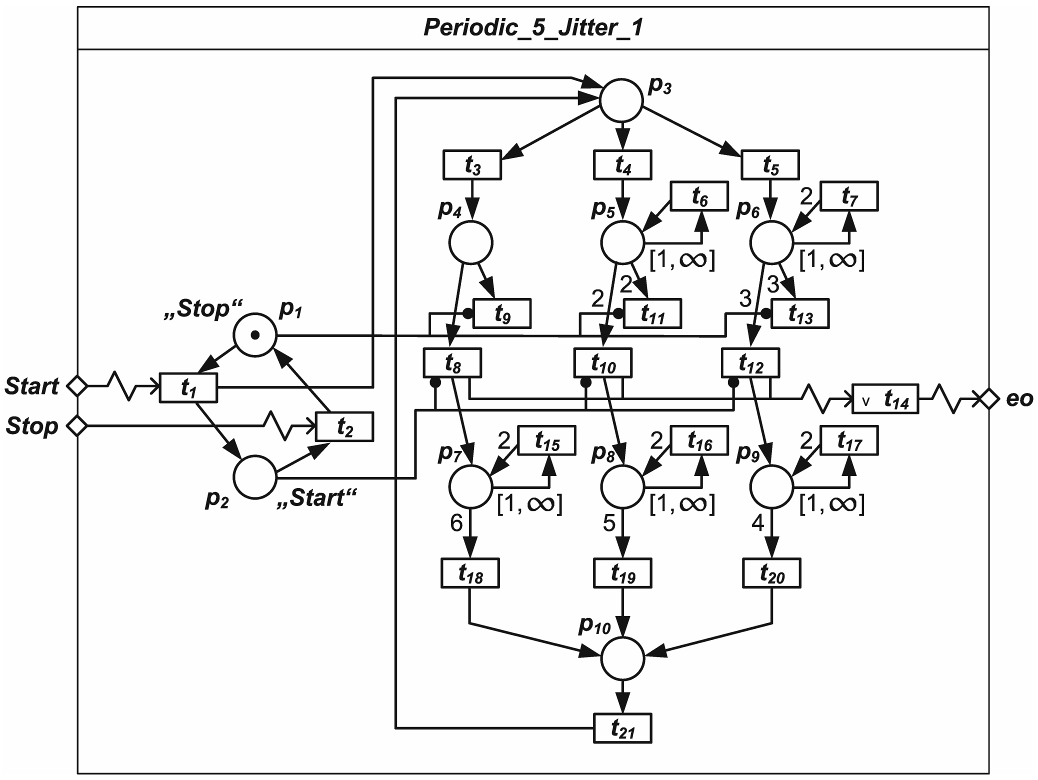


Figure 37: Representation of periodic event occurrences (5 time units) with jitter (1 time unit) [19]

The last kind of occurrence model for external events presented here is the bounded model. The bounded model is characterized by two bounds: the minimal and the maximal inter-arrival times (T_{min} and T_{max}). Any time delay between these two bounds is possible for the occurrence of two consecutive events. In the formal model of a bounded event occurrence additionally the time resolution T_{res} used in the model has to be taken into consideration. Figure 38 depicts a model with a minimal inter-arrival time of *3* time units ($T_{min} = 3$), a maximal inter-arrival time of *5* time units ($T_{max} = 5$), and a time resolution of one time unit ($T_{res} = 1$). Accordingly there are again three paths available within the module. Place 'p_3' represents the point in time when an output event 'eo' has just been emitted. Each path originated in place 'p_3' describes a certain inter-arrival time for events, whereas transition 't_3' represents the minimal und transition 't_5' the maximal inter-arrival time. The reachability graph of 'Bounded_3_5' will include any path based on the above mentioned parameters.

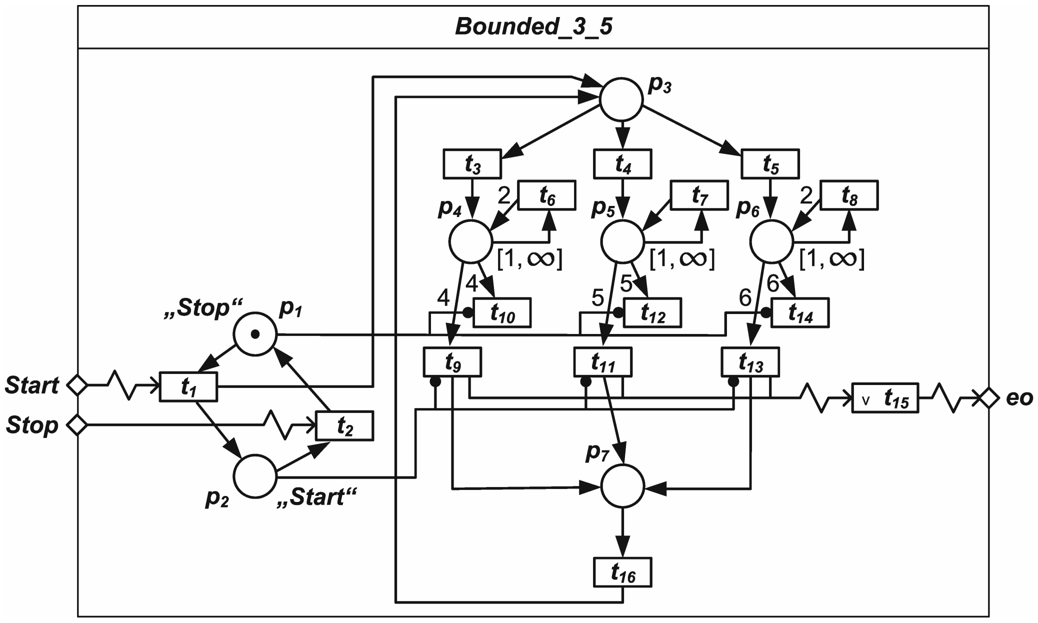


Figure 38: Representation of bounded event occurrences with minimal (3 time units) and maximal (5 time units) inter-arrival time[19]

7.6 Definition of evolution specifications

The description of the overall system architecture in terms of formal models is the first part that has to be defined for the verification by model checking. The second part concerns to the definition of the evolution specifications. According to the consideration of evaluation means for the different system integrity characteristics and evolution specifications in Section 5.2 several properties need to be fulfilled by a system evolution step. In order to define these properties two different tasks have to be handled:

- ***Specification of properties for DSE in temporal logic:*** The most important way in order to define specification for model checking is the use of temporal logic. We will discuss the use of specification patterns utilizing the representation of the elements within the system architecture presented above. The ACS customer does not need to be faced with specifications in temporal logic according to Requirement (8) "User-friendly definition of specifications".
- ***Definition of properties for DSE:*** The different system integrity characteristics that need to be checked by model checking for the reconfiguration sequence have to be analysed and appropriate general definitions need to be provided.

In addition we will investigate possible problems when restricting the scope for the verification by model checking to the reconfiguration sequence, which may be used to restrict the model of the plant and control application also only to a limited part of the overall life time of the plant.

7.6.1 Specifications in natural language

The use of temporal logic is very challenging for ACS customers as they often lack an appropriate background in the field of computer science. This situation has been highlighted for instance in Dwyer *et al.* (1998) for the use of temporal logic in general. They propose property specification patterns in order to simplify the usage of specifications in temporal logic. A property specification pattern provides a general description for a specific problem, which can be adapted to the specific property of the ACS customer.

Property specification patterns

A categorization of property specification patterns may be applied according to different characteristics. The following description follows Dwyer *et al.* (1998) and Dwyer *et al.* (1999), which have been continuously adapted and improved online in [53]. Appendix E provides a more detailed description of the different specification patterns. The main categorization is the so-called pattern hierarchy, which provides an order of patterns according to their semantics (see Figure 39). The main differentiation is based on whether a single item or an order of items is taken into consideration [53]:

Occurrence patterns: "Occurrence patterns talk about the occurrence of a given event/state during system execution."

- ***Absence property pattern:*** "To describe a portion of a system's execution that is free of certain events or states. Also known as never."
- ***Universality property pattern:*** "To describe a portion of a system's execution which contains only states that have a desired property. Also known as henceforth and always."
- ***Existence property pattern:*** To describe a portion of a system's execution that contains an instance of certain events or states. Also known as eventually."
- ***Bounded existence property pattern:*** To describe a portion of a system's execution that contains at most a specified number of instances of a designated state transition or event."

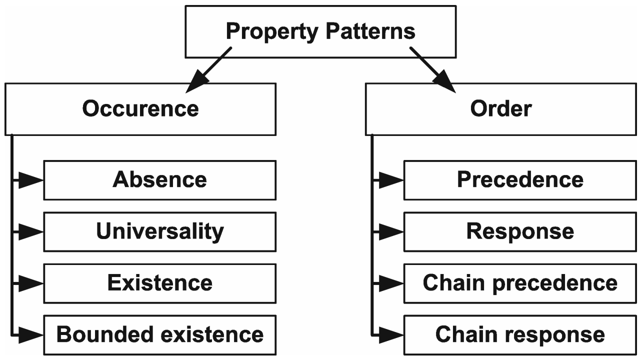


Figure 39: Pattern hierarchy, Dwyer *et al.* (1999)

Order patterns: "Order patterns talk about relative order in which multiple events/states occur during system execution."

- ***Precedence property pattern:*** "To describe relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for an occurrence of the second. We say that an occurrence of the second is enabled by an occurrence of the first."
- ***Response property pattern:*** "To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect. Also known as follows and leads-to."
- ***Chain precedence property pattern (one cause two effects version):*** "To describe a relationship between an event/state *p* and a sequence of events/states (*s*, *t*) in which the occurrence of *s* followed by *t* within the scope must be preceded by an occurrence of the sequence *p* within the same scope."
- ***Chain response property pattern (one stimulus two responses version):*** "To describe a relationship between a stimulus event (*p*) and a sequence of response events (*s*, *t*) in which the occurrence of the stimulus event must be followed by an occurrence of the sequence of response events within the scope."

Scope of patterns: In addition to these patterns the scope for each pattern may be constrained according to different aspects. Dwyer *et al.* (1998) distinguish five different kinds of scopes (see Appendix E for a more detailed description), as for instance the specification has to be valid globally, before a certain condition or after a certain condition. Furthermore two different variants for scopes that describe validity in between of different conditions are mentioned.

Application for the ACS customer: Each pattern with a specific scope is represented by an extensive description. The formula may be given in different kinds of temporal logic; a description of the aim of the pattern in natural language as well as a different example for its application can be included. The ACS customer is able to select the pattern which fits best to the specification he wants to define and only has to provide the data to parameterize the formula in temporal logic. According to the features provided by the engineering tool the ACS customer may be supported by dialogs and selection wizards, as this has been proposed for instance in Bitsch (2003) for the special purpose of safety requirement specifications.

Representing system architecture elements in specifications

In order to apply the above given property specification patterns for the evaluation of DSE, the elements of the formal models (places and transitions) have to be used for parameterization. In the best case the ACS customer does not need to mention any place or transition by himself, instead he should use the same means he has already used for modeling control applications: the elements of the IEC 61499 standard. Based on the models provided in Section 7.4 the following mappings can be used in order to identify places and transitions in the NCES models within the architectural elements of the control device:

- ***Activity of tasks:*** The scheduler provides a condition output in order to control the execution of tasks via the ‘enable’ condition input (see Figure 30). The activation of a task is represented by the place within the scheduler which is connected to this condition output. In order to check the activity of certain event chains, the activity of the related tasks has to be checked. As the callback function of an interrupt is also represented by a task, the same place within the scheduler may be used to describe the activation of interrupts.
- ***Usage of critical section within R^3E:*** As depicted for the example event dispatcher a critical section within the formal model is bounded by modules for the registration of the execution flow. The corresponding storage element (e.g., places ‘p_1’ and ‘p_2’ in module ‘Semaphore’ in Figure 34) can be used to address the usage of these critical regions in the specifications.
- ***Sending of output events:*** The model of a connection can be used to address the sending of an output event within the formal model. Place ‘p_4’ in Figure 26 (managed event connection) depicts exactly the situation when the connected input event is put into the event dispatcher. The concrete type of the FB emitting the event can be neglected in this case.
- ***Data outputs:*** A data output of an FB may be represented in the internal model of the FB, but it can be addressed also by using the model of the data connection. The internal storage of the data connection (e.g., places ‘p_5’ and ‘p_6’ in Figure 27) can simply be used to represent the data output of an FB without taking care of the internal representation of the FB.
- ***Triggering of BFBs:*** The triggering of a BFB by a certain input event is registered within a module (e.g., ‘InputEvent’ in Figure 35b) including a storage element for registration of the input event. The places of this storage element can be used to address the triggering of a BFB in a specification.

- ***ECC state:*** The states of a BFB according to its ECC are directly represented as places in the model of a BFB. The module 'ECC' in Figure 35b includes a place for each state of the ECC, therefore these places can be used to address the state of a BFB. A state transition may be identified by the transitions mentioned in the modul 'ECC', which directly represent the transitions of the ECC.
- ***Creation and deletion of connections:*** The models provided for data and event connections include a state machine that defines the state of a connection, whether it is available or no*t*. The places within these modules (see Figure 26 and Figure 27) can be used to address the current status of the connection. The transitions within the state machine represent the moment in time when the management command is executed.
- ***Operational state of managed FBs:*** The operational state machine for managed FBs is directly represented in the formal model of FBs (see Figure 28). The incorporation of the operational state in the specification is provided by these states, the execution of state changes is characterized by the appropriate transitions.

The interrelation of parameters within the specifications and the elements of the formal model is a matter of support in the engineering tool. Based on the rules given above the ACS customer will be able to define the variables within the property specification patterns without being in touch with the formal model itself.

7.6.2 Evolution specifications

The various properties that need to be checked in order to achieve system integrity during the execution of a system evolution step have been defined already in Section 5.2.1. According to the discussion of evaluation means in Section 5.2.2, the following properties of the evolution specification need to be checked by using verification by model checking:

Global consistency: Any specification that has been defined for the normal operation of the ACS has to hold also during the execution of a system evolution step. The specifications for normal operation have been split up in Section 5.1.1 into plant, process, and product specifications.

Local consistency: Similar to global consistency.

Active references: This properties aims at the check of interrelations between different parts of the control application based on services within the underlying system configuration. In an IEC 61499 based system these interrelations are based on the internals of SIFBs. If there are any disruptions of such an interrelation the properties for normal operation (plant, process, and product specification) will be violated. Based on the detailed model of these interrelations and their manipulation during the system evolution step it is sufficient to check the properties for normal operation.

State management: One main prerequisite for DSE is to disturb the operation of the plant as little as possible. This is especially achieved by appropriate transition management policies as described in Section 3.5 within the ECA. In order to check the integrity characteristic for state management, the effect to the plant under operation has to be taken into consideration. There are two different situations that need to be distinguished:

✓ The actions within the system evolution step commence a disturbance of a certain amount into the system, which will disappear in a short time frame apart from the execution of the DSE. Within the property specification patterns the absence pattern for a specific time frame (which can be specified by a beginning and an end condition) may be used (Appendix E, Equation 35). On the other hand it is possible to use timed CTL for the proposed model checker SESA, which enables the restriction of temporal parameters to a certain time frame. Accordingly the absence pattern for the consideration of all paths after a certain condition may be adapted to

$$\mathbf{AG}\ (\ Q \rightarrow \mathbf{AG}_{[0,\,a]}(\neg p)\ ) \qquad (13)$$

where Q marks the start of the reconfiguration sequence,
$[0, a]$ is the time frame for the observation of disturbances, and
p describes the occurrence of an exceeding disturbance.

Equation 13 uses the operator **G** with a time scope of a certain upper limit a, which does represent a sufficient time frame for the DSE of the control application. Based on restrictions of SESA the operator **G** is not supported with such a time scope. By using equivalences within the temporal operators in CTL Equation 13 can be formulated as

$$\mathbf{AG}\ (\ Q \rightarrow \neg\mathbf{EF}_{[0,\,a]}(p)\ )^{22}, \qquad (14)$$

which will be supported by the model checker SESA.

The open question for the application of Equation 14 is the selection of an appropriate property for condition p that describes the occurrence of an exceeding disturbance. This condition is highly application dependent. When we consider for instance the exchange of the controller within a closed-loop control circuit (see Section 4.2.3), we may use the difference between the reference value and the control value to determine condition p. If the ACS customer is aware of the concrete value of the control variable (e.g., because the trigger for the reconfiguration sequence is based on this value) a simple upper or lower limit for the control variable may be chosen as condition p. Of course, also any other description using temporal logic may be used in order to define condition p.

- The second possibility is that based on the DSE an incentive to the system is introduced which leads to a continuously growing disturbance. A concrete example for a closed-loop control example is that the controller becomes unstable. The transient will be very small in the first time after the DSE, but will lead to high disturbances to the overall system later on. Therefore a detection of this kind of fault will not be possible in a given time frame after the start of the reconfiguration sequence. An infinite time frame is not possible because for instance a high difference between reference value and control value may be part of the normal operation of the plant (when the reference value is changed stepwise). But as this kind of failure will lead to a permanent disruption of the normal plant behavior, it is possible to use the specifications for normal operation (plant, process, and product specification) in order to check for continuously growing disturbances commenced by the system evolution step.

Real-time constrained operation: The system integrity characteristic for real-time constrained operation aims at the execution of the ECA, which has to fulfill certain real-time constraints, too, because it influences a control application with real-time constraints. Next to the properties for the functional behavior of the ECA also the temporal behavior has to be taken into consideration (see also Requirement (1) "Temporal behavior"). The reconfiguration sequence will be executed only once and has to be finished in a certain amount of time according to the control application. Based on the property specification patterns several possibilities exist to check the real-time constrained operation of an ECA:

- The property of real-time execution may be related to the operation of the control application. The existence pattern for a specific time frame (which can be specified by a beginning and an end condition) may be used without taking care of the concrete execution time of the ECA in the following way:

$$\mathbf{AG}\ (\ Q \wedge \neg R \rightarrow \mathbf{A}\ [\ \neg R\ \mathbf{W}\ (\ p \wedge \neg R)\ ]\ )^{23}, \qquad (15)$$

[22] Based on the equivalence $\mathbf{AG}\,f = \neg\mathbf{EF}(\neg f)$, see for instance Clarke *et al.* (1999, Section 3.2).

where Q marks the start of the reconfiguration sequence,

R marks a condition within the control application, when the reconfiguration sequence has to be finished, and

p describes the end of operation of the reconfiguration sequence.

- The execution time of the reconfiguration sequence may be checked based on timed CTL formulas, too. As basis the universality pattern after a certain condition (Appendix E, Equation 39) may be enhanced by the application of a specific time frame. As already depicted for Equation 13 the limitations of the model checker SESA require a reformulation of this pattern to

$$\mathbf{AG}\ (\ Q \rightarrow \neg\mathbf{EF}_{[a,\,b]}(\ \neg p)\ )^{22}, \tag{16}$$

where Q marks the start of the reconfiguration sequence,

p describes the end of operation of the reconfiguration sequence, and

$[a, b]$ is the time frame when the operation of the ECA has to be finished (a may be set to zero).

- Based on the system behavior described in the system model it is possible to use verification by model checking not only to check some properties but additionally also performance characteristics may be evaluated in order to provide additional information for the ACS customer. Clarke *et al.* (1999, Section 16.4) describe two algorithms that are able to calculate the minimal and maximal delay between a request and the corresponding response. For the use of these algorithms the request could be marked by the start of the reconfiguration sequence, and the corresponding response would be the end of the reconfiguration sequence. The ACS customer receives the BCET and WCET execution time of the reconfiguration sequence, which may be helpful for the fine-tuning of the behavior of the system. The model checker SESA does not provide such algorithms.

7.6.3 Evaluation of small portions of system behavior

DSE is a single action that influences the behavior of the overall system, as already depicted in Section 1.1. Therefore the interesting part for verification by model checking is limited to the execution of the reconfiguration sequence, which is related to a very small part of the overall system behavior. Accordingly it may be possible to restrict the formal model of the system to this small part in contrast to modeling the overall system behavior, which is especially interesting for the model of the plant because it is a highly time consuming task (other parts may be generated automatically). But on the other hand there are also some risks that need to be evaluated for the restriction to a small portion of system behavior:

- The model of the plant may lack details that are important for the execution of DSE. The system behavior may not include effects on the ECA in certain circumstances.
- The variety of paths from the initial state of the plant to the starting point of the reconfiguration sequence may be very high. A limited system model can be enhanced by using different initial states in order to incorporate the high variety of the overall system model. But it may be difficult to define the different initial states of the limited system model.
- Several integrity characteristics within the evolution specifications are checked by using the specifications for normal operation (plant, process, and product specifica-

[23] A description of the weak until operator **W** is given in Appendix E.

tion). Therefore, it is necessary that the model of the system provides all information about these properties; otherwise the check for the evolution specifications will not be possible in all details.

7.7 Summary

The evaluation of properties for the execution of a system evolution step by means of model checking is restricted to the reconfiguration sequence. Only in this sequence the functional behavior of the control applications will be actively changed. Based on the methodology for model checking three tasks have to be taken into consideration: design of the system model, definition of the specification, and execution of the model checking algorithm.

The system model has to incorporate two essential parts for DSE: the real-time behavior within formal models and the dynamic reconfiguration of the model. The execution behavior of elements within a control device is not only characterized by a model of time within the modeling language. It is essential to provide models that are capable to model interruptible time delays within the formal models, because a real-time computer system is characterized by the preemption of tasks in the RTOS and external event sources. On the other hand, the dynamic reconfiguration of models has been taken into consideration. No means exist in modeling languages to directly include changes during the evaluation process, as the methodology of model checking is based on the assumption of a given model. But the existence of different models can again be modeled within the execution flow of the formal model, which has the same effect in the behavior of the overall system. The different basic reconfiguration services that may be used within the reconfiguration sequence of a system evolution step can be modeled in this way.

The definition of specifications is especially problematic for ACS customers. The usage of property specification patterns provides a good basis, which enables the determination of properties without directly using temporal logic. The necessary parameters for the different patterns can be customized based on the formal model, which again may be abstracted by means of the programming language for the control application. In addition, the definition of a specification in terms of a pattern provides independence from a special dialect of a temporal logic. The main prerequisite for DSE is that the normal operation of the plant will not be disturbed. Accordingly, several integrity characteristics can be checked by using the specification for the normal operation of the plant. In addition, the properties state management and real-time constrained operation may be taken into consideration by explicit specifications in order to achieve additional information about the quality of the system evolution step.

Chapter 8

Demonstration and Experiments

In order to provide a proof of the concept for the proposed methodology for the evaluation of DSE it is necessary to provide the overall description of a control device, the control application, and the modeled ECA. We will use a microcontroller platform, which is installed for different purposes in the Odo Struger Laboratory at the ACIN. The given example represents a typical situation for the exchange of some parts within a control application and has been executed on this platform. The evaluation of the example will be structured into KAPPA-based calculations, which have been integrated in an engineering tool, as well as verification by model checking, whereas the generation of appropriate models has been done by hand.

Next to this example, two further experiments dedicated to the verification by model checking are presented as a more detailed description of aspects for formal modeling within the evaluation framework proposed within this work. Due to limitations of the used model checking tool an integration of these aspects into the evaluation example was not possible. In detail, the handling of the priority inversion policy for the access to the event dispatcher and the integration of a plant model are taken into consideration.

8.1 Typical example on a specific test model

The demonstration is based on the engineering tool, which has been developed during the research project εCEDAC [8]. This environment provides on the one hand an appropriate framework for the engineering of IEC 61499 based control systems, and on the other hand has been extended to integrate the proposed modeling methodology for DSE. The evaluation of properties based on KAPPA-based calculations has been added to the engineering tool and therefore represents an integral part of the engineering tool. The evaluation of properties by model checking has been based on the tools Visual Verifier, Visual Editor and TNCES Editor (see also Appendix C.3).

8.1.1 Demonstration control device

The demonstration control device is based on an evaluation board for a microcontroller and would not be used in commercial applications in this configuration. But the missing elements only concern external parts such as I/O interfaces and therefore do not restrict the functional elements of the demonstration control device. According to the description about the architectural elements of the system model used for verification by model checking (see Figure 24 in Section 7.1) the concrete representation of the three gray shaded elements will be considered in the following explanations.

Processing unit

The evaluation board phyCORE-AT91M55800A (Phytec, 2003) from the company Phytec Messtechnik GmbH [43] utilizes an ATMEL AT91M55800A microcontroller, clock generation, memory modules as well as different interface peripherals (e.g., an Ethernet controller).

The performance of the processing unit will be described by the measurements of parameters for the RTOS and the R^{3}E in the following paragraphs. These values are usable only for this special configuration of the control device.

Real-time operating system

The RTOS eCos, which has been described briefly already in Section 7.4.1 represents the basis for any program execution within the control device builds the. The various configuration parameters for eCos have to be incorporated into the formal model. In addition, also the execution time for explicitly recognizable actions within the operating system has to be included according to the given processing unit. In case of the above mentioned evaluation board phyCORE-AT91M55800A the measurements of these parameters are summarized in Table 5 according to the work done by Ferhatbegovic (2007) under supervision of the author.

Time parameter	Value
Task switching time	82,0 µs
Task suspension	10,1 µs
Task resumption	13,3 µs
Task creation	85,5 µs
Task deletion	96,0 µs

Table 5: Real-time behavior of eCos on the demonstration control device, Ferhatbegovic (2007)

Real-time reconfiguration runtime environment

The control device utilizes the IEC 61499 compliant runtime environment described in Zoitl (2007) with some enhancements, which are related to additional functionality with respect to basic reconfiguration services. A detailed description of the execution behavior of R^{3}E has been provided in Section 6.1.2 (execution time calculation for event chains) as well as Sections 7.4.2 and 7.4.3 (with regard to the formal model of the elements of the runtime environment and the control application). For both purposes time parameters for the real-time behavior on a specific processing platform need to be provided in order to verify the different properties for DSE. Table 6 depicts a set of parameters which are necessary for the evaluation of control applications as well as FBs (as far as they are independent from a specific FB type), enhanced by the work done by Mandl and Zhang (2008) under supervision of the author.

Time parameter	Value
Insertion of an event T_{entry} (only 1 event, no data)	23,2 µs
Fetching of an event and triggering of the corresponding FB T_{invoke}	11,0 µs
Offset for data sampling $T_{DS,offset}$	67,4 µs
Data sampling of INT data type $T_{sample,INT}$	6,8 µ,s
Evaluation of active ECC state T_{AS}	0,8 µs
Evaluation of an ECC transition with	
“1” condition $T_{Cond,1}$	6,9 µs
Boolean condition $T_{Cond,bool}$	7,61 µs
event as condition $T_{Cond,event}$	7,02 µs
event and Boolean condition $T_{Cond,mix}$	7,60 µs
Constant time for algorithm execution $T_{Alg,const}$	6,4 µs

Table 6: Real-time behavior of R^{3}E within the demonstration control device, Mandl and Zhang (2008)

KAPPA vector of the demonstration control device

All information about the demonstration control device has to be provided in an appropriate form to the engineering tool, in order to integrate it into the evaluation process. Next to the above mentioned timing parameters for instance also memory consumption, FB types, or the configuration parameters of eCos need to be included. Appendix F depicts a description file representing the KAPPA vector of the demonstration control device (without any control applications) based on FDCML, as already described in Section 5.3.2.

The description of a control device can be used within the engineering tool to provide a more detailed view of the ACS. Figure 40 depicts the graphical representation of the IEC 61499 device 'myDevice' which is located on the demonstration control device within the εCEDAC engineering tool. Based on the KAPPA vector information the communication interface and the process interface of the device can be described according to the actual configuration. Furthermore two IEC 61499 resources exist within 'myDevice': The resource 'MGR' is part of each device according to the used compliance profile, 'Res_App1' is established during the engineering process.

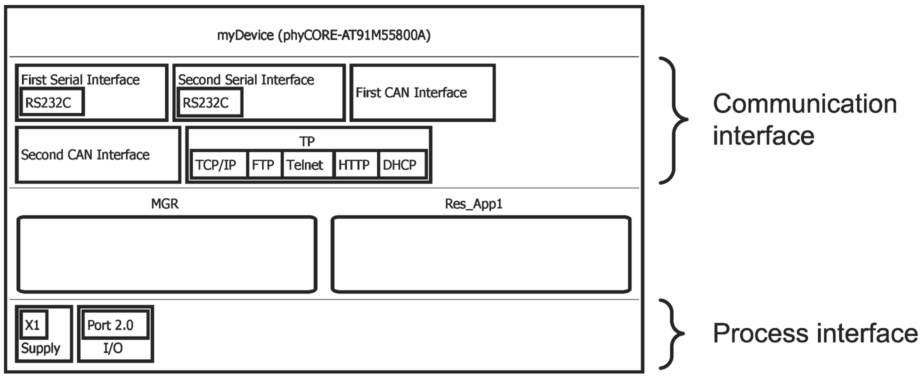


Figure 40: Demonstration control device within the εCEDAC engineering tool

8.1.2 Typical control example

In order to use a relevant practical example for the demonstration of DSE and its evaluation, the following aspects have been taken into consideration:

- Internal variables of FB instances have to be involved in the evolution process in order to check also state management characteristics.
- The control application should be independent of a plant as well as the network in order to have the demonstration focused on the evolution process.

The resulting control application for the practical example adds a certain value to an internal element. The periodic addition is triggered by the timer and ends after the internal variable has exceeded a certain limit. A similar situation can be found for a closed-loop control circuit whose controller has to be triggered (see the example given in Section 4.2.3), the exchange of a filter in the feedback loop of a control circuit, or the exchange of an encryption algorithm of a communication channel. Figure 41 depicts the initial state of the chosen typical control application, and Figure 42 provides in the upper part the user interface in order to interact with this control application. The control application uses the FB 'TAKT' in order to generate the necessary trigger for the calculations. The user controls 'TAKT' via the check box 'START', which sends a Boolean value to the control application. This value is decoded by the FB 'E_SWITCH', whose output events start or stop the FB 'TAKT'. During each clock cycle the FB 'E_CTU' is triggered, which increases a counter variable ('E_CTU.CV) at each call, starting from the value *0*. This data output is used as input for the addition within the FB 'ADD_INT_TO_INTERNAL' (FB 'CONV_UINT2INT' is necessary for a type conversion from the UINT to the INT data type). The output value 'ADD_INT_TO_INTERNAL.OUT' represents the result of the addition of this value and the internal variable. Both values, the data output 'CV' from 'E_CTU' and the data output 'OUT' from 'ADD_INT_TO_INTERNAL' are sent to the user interface at each cycle of the control application. FB 'CHECK_INT_GREATER' provides the evaluation if the current value of variable 'ADD_INT_TO_INTERNAL.OUT' exceeds the given limit, which is set to *100* in Figure 41. If the condition *ValA > ValB* is fulfilled, the Boolean output 'CHECK_INT_GREATER.Result' is set to true and the FB 'E_PERMIT' will send an output event 'EO' which stops 'TAKT'.

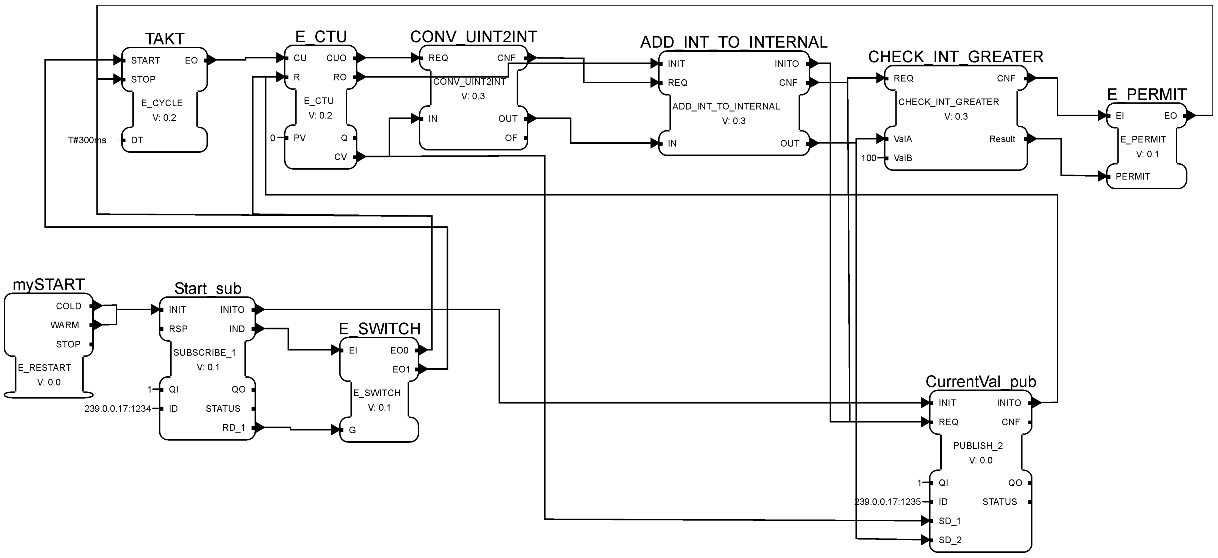


Figure 41: Typical control application example (initial state)

Two different system evolution steps have been modeled for this control application. The corresponding CECA is depicted in Figure 43, which includes two EECFBs:

- 'Change_Threshhold' will change the limit for the evaluation of the current value of the integrating variable. 'CHECK_INT_GREATER.ValB' is changed to a different value based on the execution flow of the control application. We will not go into detail for this system evolution step.
- 'Subtract' includes a bigger evolution within the control application. The task of DES is to exchange the summing up the 'E_CTU' output value by subtraction from the internal element. In order to stop the control application after exceeding a given limit, also a different kind of check has to be applied, if the internal element falls below a defined value. We will take into consideration this system evolution step and discuss the application of the evaluation methodology.

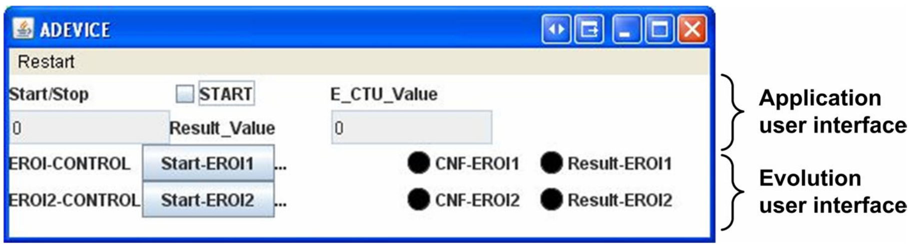


Figure 42: Application and evolution user interface for the typical control example

The two EECFBs are executed sequentially, whereas the successful execution of a sequence of the system evolution step is the precondition for starting the next sequence. The initial trigger for the execution of each system evolution step is provided by the corresponding user interface for the DSE as depicted in the lower part of Figure 42. Furthermore also a feedback from the execution of the EECFBs is provided to the user: 'CNF-EROI*x*' signals that event output RDINITO has been emitted, and ‘Result-EROI*x*’ represents the value of the data output ‘RDINIT_QO’ (*x* stands for *1* or *2*). Each EECFB may be triggered independently by the user by using the buttons ‘Start-EROI*x*’.

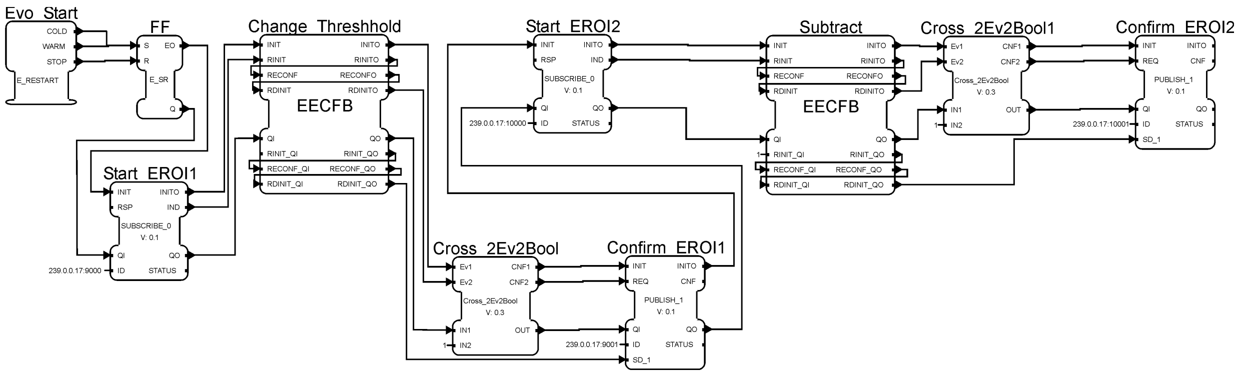


Figure 43: CECA including two system evolution steps for the typical control example

System evolution step Subtract

The ECA included within the EECFB ‘Subtract’ is given in Appendix G, Figure 72. We will describe the actions within the EECFB based on the control application’s model. Figure 44 depicts a virtual view of the control application. Next to the initial system state (solid lines) also the additional items of the final system state (dashed lines) are included. This situation never occurs during the system evolution step. The real situation within the control application after each of the three main sequences of the system evolution step is depicted in Appendix G: Figure 73 provides the situation after the execution of the RINIT sequence, Figure 74 after the RECONF sequence, and Figure 75 after the RDINIT sequence (which is already the final system state from the control application’s point of view). The CECA itself is located in a separate resource within the control device, therefore its download and deletion is not visible within the control application.

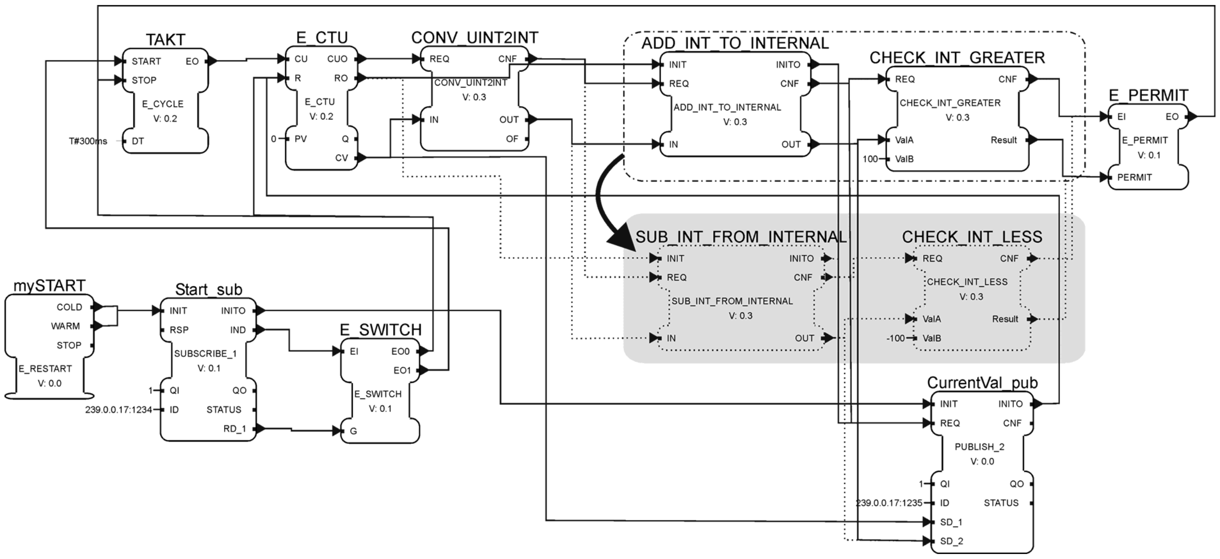


Figure 44: Typical control application: mixed representation of initial and new system state

RINIT sequence

The RINIT sequence is responsible for the preparation of the necessary elements within the control application for the new system state. The new functionality of subtracting the data output ‘E_CTU.CV’ from an internal variable will be provided within the FB ‘SUB_INT_FROM_INTERNAL’, which is established during this phase. Furthermore also the FB ‘CHECK_INT_LESS’ is necessary, because the new condition *ValA* < *ValB* has to be evaluated. Next to these two FB instances also the input connections for both FBs are created. These are

- 'CONV_UINT2INT.CNF' to 'SUB_INT_FROM_INTERNAL.REQ',
- 'CONV_UINT2INT.OUT' to 'SUB_INT_FROM_INTERNAL.IN',
- 'E_CTU.RO' to 'SUB_INT_FROM_INTERNAL.INIT', as well as
- 'SUB_INT_FROM_INTERNAL.CNF' to 'CHECK_INT_LESS.REQ', and
- 'SUB_INT_FROM_INTERNAL.OUT' to 'CHECK_INT_LESS.ValA'.

After starting the two new FB instances within the RINIT sequence the internal value within 'SUB_INT_FROM_INTERNAL' as well as the check for falling below the given limit are calculated at each cycle of the control application. But these new calculations do not have any impact to the behavior of the control application.

RECONF sequence

Within the RECONF sequence only the output connections of the marked application parts within Figure 44 have to be changed. The ECA splits up these actions into two parts: the connections within the logic part of the control applications and the connections to the user interface. We will describe only the first part of actions, as herein also the transition management has to take place:

- DELETE the connection from 'CHECK_INT_GREATER.CNF' to 'E_PERMIT.EI'
- DELETE the connection from 'CHECK_INT_GREATER.Result' to 'E_PERMIT.PERMIT'
- READ the internal variable of 'ADD_INT_TO_INTERNAL'
- WRITE the internal variable of 'SUB_INT_FROM_INTERNAL'
- CREATE the connection from 'CHECK_INT_LESS.Result' to 'E_PERMIT.PERMIT'
- CREATE the connection from 'CHECK_INT_LESS.CNF' to 'E_PERMIT.EI'

Based on the conditions within the plant there may be different real-time constraints for the execution of these two parts, as the logic within the control application may be more important than the correct update of the user interface. Within the ECA given in Appendix G, Figure 72, there are no real-time constraints modeled. The control application and the ECA are executed in different resources of the same device, but according to the point in time when the ACS customer triggers the system evolution step, it may be possible to violate specifications of the application and introduce disturbances due to the DSE.

RDINIT sequence

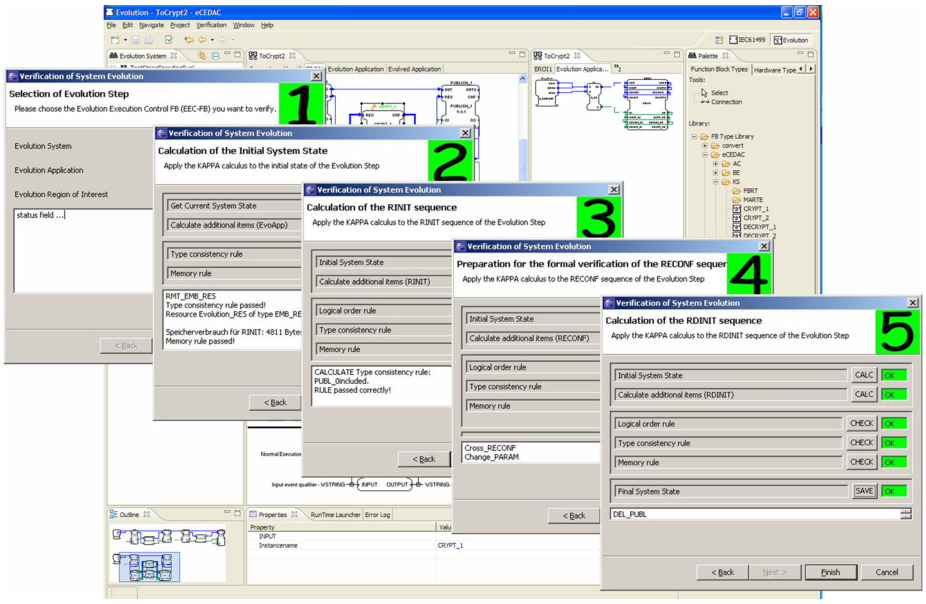
Within the RDINIT sequence the roles between the two marked application parts within Figure 44 have changed. The gray shaded part is now active, and the FBs 'ADD_INT_TO_INTERNAL' and 'CHECK_INT_GREATER' do not have any effect on the application behavior (the output connections of 'CHECK_INT_GREATER have been deleted, but the FBs are still calculated). Within this sequence both FBs have to be stopped, their input connections are deleted, and at the end both FBs are deleted.

8.1.3 KAPPA-based calculations

The evaluation of system integrity characteristics based on KAPPA-based calculations has been identified as appropriate evaluation means in Section 5.2.2 for global and local consistency, dependent operation, and requirements of resources. As basis for these evaluations a wizard has been implemented in the εCEDAC engineering tool, which will be described in general below. Furthermore the evaluation itself will be taken into consideration.

Evaluation wizard within the εCEDAC engineering tool

The εCEDAC engineering tool is an IEC 61499 compliant software which has been enhanced towards the modeling of DSE. For the description of control devices, which is out of the focus of the IEC 61499 standard, the format depicted in Sections 5.3.2 and 8.1.1 is used within the engineering tool to represent the current system state KAPPA. A given control device has to be represented by using an XML description file (see Appendix F for an example) and can be integrated in the type library of the engineering tool for further use within the engineering process. Figure 45 shows a screenshot of the εCEDAC engineering tool in the background. For the evaluation of evolution specifications based on KAPPA-based calculations an evaluation wizard has been integrated into the engineering tool. The different pages of the evaluation wizard are depicted in the foreground of Figure 45.



Verification of System Evolution

Selection of Evolution Step
Please choose the Evolution Execution Control FB (EEC-FB) you want to verify.
Evolution System
Evolution Application
Evolution Region of Interest
status field ...

Calculation of the Initial System State
Apply the KAPPA calculus to the initial state of the Evolution Step
Get Current System State
Calculate additional items (EvoApp)
Type consistency rule
Memory rule
RMT_EMB_RES
Type consistency rule passed!
Resource Evolution_RES of type EMB_RE
Speicherverbrauch für RINIT: 4811 Byte
Memory rule passed!

Calculation of the RINIT sequence
Apply the KAPPA calculus to the RINIT sequence of the Evolution Step
Initial System State
Calculate additional items (RINIT)
Logical order rule
Type consistency rule
Memory rule
CALCULATE Type consistency rule:
PUBL_0included.
RULE passed correctly!

Preparation for the formal verification of the RECONF sequen
Apply the KAPPA calculus to the RECONF sequence of the Evolution Step
Initial System State
Calculate additional items (RECONF)
Logical order rule
Type consistency rule
Memory rule
Cross_RECONF
Change_PARAM

Calculation of the RDINIT sequence
Apply the KAPPA calculus to the RDINIT sequence of the Evolution Step
Initial System State
Calculate additional items (RDINIT)
Logical order rule
Type consistency rule
Memory rule
Final System State
DEL_PUBL

Figure 45: Screenshot of the eCEDAC engineering tool and the evaluation wizard

The evaluation wizard consists of five pages, which are related to the different sequences of a system evolution step (except the deletion of the ECA which needs no evaluation):

- Page 1: Selection of the EECFB which should be verified within a CECA.
- Page 2: This step is related to the download of the ECA. As first step, the current system state is evaluated. Herein the currently active IEC 61499 applications are queried from the specified control devices. According to Figure 21 ‘KAPPA 1’ is now available within the evaluation wizard. The wizard page takes into consideration the overall CECA, which needs to be downloaded, and evaluates the requirements of resources. Furthermore, the KAPPA vector is enhanced with the CECA, which means that ‘KAPPA 2’ is created according to Figure 21, the initial situation for the RINIT sequence.
- Page 3: The RINIT sequence is handled within this wizard page. The FBs as well as their execution order related to the RINIT sequence are identified. Based on this information, the properties requirements of resources and dependent operation are evaluated. During the evaluation for dependent operation, the given system state is adapted according to the basic reconfiguration services included in the RINIT sequence and the initial system state for the RECONF sequence (‘KAPPA 3’ in Figure 21) is created.
- Page 4: Similar to the RINIT sequence this wizard page identifies the FBs and their execution order for the RECONF sequence, evaluates the properties requirements of

resources and dependent operation, and finally provides the initial system state for the RDINIT sequence ('KAPPA 4' in Figure 21).

- Page 5: The last wizard page is devoted to the RDINIT sequence. Again the similar actions as already described above for RINIT and RECONF sequence are executed within this wizard page. As final step the initial system state for the deletion of the ECA is created ('KAPPA 5' in Figure 21).

Each system state, which is generated by the evaluation wizard, is available within the engineering environment for documentation purposes. Since the last step, the deletion of the ECA, does not need any evaluation, the evaluation wizard does not take this step into account.

Check for global and local consistency

The given typical example for DSE does not mention the real-time constrained execution of the control application, therefore the evaluation for global and local consistency within the preparation phase of the system evolution step is not necessary. But in order to give an impression of the effort related to this property, we will describe the evaluation process briefly. The evaluation has to be split up into the two sequences within the preparation phase: download of the ECA and RINIT sequence.

For the download of the ECA, the runtime environment is expected to handle the download procedure without influences on the control application. But the interrelations between the ECA and the control application have to be taken into consideration, as there might be additional FBs triggered within the context of the control application. For the above given typical control example no additional interrelations between the control application and the CECA (especially the EECFB 'Subtract') exist.

The RINIT sequence has to be considered in more detail, as there are new FBs which are included into the control application. Figure 46 depicts the situation after the execution of the RINIT sequence, yet only the relevant part for the evaluation is presented. The new FBs 'SUB_INT_FROM_INTERNAL' and 'CHECK_INT_LESS' as well as their input connections (marked by bold lines in Figure 46) are added and have to be executed in the same context as the control application.

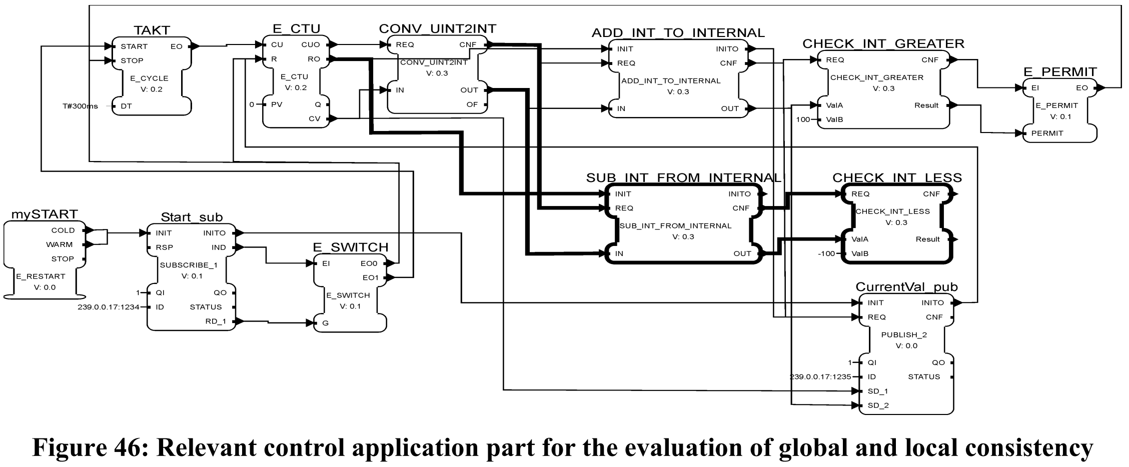


Figure 46: Relevant control application part for the evaluation of global and local consistency

There are two different event chains within the control application, which are affected by the new FBs. The first one is the calculation of the overall application, which is triggered by 'TAKT' now has to additionally execute both new FBs. Secondly a possible interruption of the operation by a stop command from the user interface has to be taken into consideration, where only the new FB 'SUB_INT_FROM_INTERNAL' will be executed.

- The additional effort within the event chain triggered by 'TAKT' starts with the insertion of the input event 'SUB_INT_FROM_INTERNAL.REQ'. Of course, this event will be fetched and the related FB is executed, which again sends an output event interrelated to ‘CHECK_INT_LESS.REQ’, and finally ends with the call of ‘CHECK_INT_LESS’. According to Equation 7 the additional time effort within this event chain is

$$\begin{aligned}T_{EC,add} &= T_{entry,1} + T_{invoke} + T_{SUB\_INT\_FROM\_INTERNAL}(REQ) + T_{entry,2} + \\ &\quad + T_{invoke} + T_{CHECK\_INT\_LESS}(REQ) \\ T_{EC,add} &= 4{,}9\mu s + 11{,}0\mu s + 167{,}0\mu s + 97{,}4\mu s + 11{,}0\mu s + 165{,}5\mu s \\ T_{EC,add} &= 456{,}8\mu s\end{aligned} \quad . \tag{17}$$

The first insertion of an event $T_{entry,1}$ is very fast because ‘CONV_UINT2INT’ already includes one connected input event. Only the additional effort for the second connected input event (see Mandl and Zhang (2008) for more details) needs to be taken into consideration. $T_{entry,2}$ includes in contrast the overall effort for sending one event as well as the latching of one data output. According to Table 6 this time can be calculated as $T_{entry,2} = T_{entry} + T_{DS,offset} + T_{sample,INT}$ (the data types BOOL and INT need the same time for sampling).

- For the second event chain, which is triggered by the user interface, the additional effort within the control application can be calculated as

$$\begin{aligned}T_{EC2,add} &= T_{entry} + T_{invoke} + T_{SUB\_INT\_FROM\_INTERNAL}(INIT) \\ T_{EC,add} &= 4{,}9\mu s + 11{,}0\mu s + 103{,}5\mu s \\ T_{EC,add} &= 119{,}4\mu s\end{aligned} \quad . \tag{18}$$

For the evaluation of global and local consistency of the control application the new time consumptions for all event chains within the control device have to be calculated and the synthetic utilization $U_{syn}(t)$ needs to be checked with the boundary given from the scheduling theory for the used scheduling algorithm within the control device (see Section 6.1.1).

Check for dependent operation

The system integrity characteristic dependent operation aims at the check for the applicability of basic reconfiguration services within the ECA. Therefore, the current KAPPA vector needs to be available for this evaluation for exactly that moment, when the corresponding FB has to be executed. This property has to be checked for the RINIT, the RECONF, and the RDINIT sequence, whereas the procedure for this check is the same for each sequence. As starting point within the evaluation wizard, the initial system state of each sequence is available.

The different tasks for the evaluation are:

- ***Identification of basic reconfiguration services:*** Each of the relevant sequences is free programmable by the user, therefore it is important to identify the execution order for the involved basic reconfiguration services (based on the experiences of the author a sequential execution is sufficient). The execution is based on the execution semantics of the runtime environment, which has to be mentioned by the identification algorithm. For each of the identified basic reconfiguration services, the following actions have to be performed.
- ***Check for basic reconfiguration service type:*** First of all the type of the evaluated basic reconfiguration service needs to be identified. The runtime environment within the control device needs to be able to execute this type of service.
- ***Check for applicability:*** The parameters of the basic reconfiguration service need to be valid within the current system state. If for instance a new connection should be established, the source and destination of this connection need to be available within the current system state. Additionally, further requirements need to be fulfilled: source

and destination have to be of the same type (e.g. INT) and in case of a data connection the destination needs not to be connected. Section 6.2.1 provides a detailed analysis of the different IEC 61499 management commands and their dependencies.

- ***Update of current system state:*** The last step for each basic reconfiguration service is the application to the current system state (only within the evaluation wizard), in order to generate the correct new system state for the evaluation of the next command.

The evaluation wizard implemented within the εCEDAC engineering tool documents the results of the evaluation of the dependent operation property in two different ways. On the one hand, the final system state for each sequence is stored as separate project within the engineering tool. The current state of the control application is depicted in Appendix G, Figure 73 to Figure 75. On the other hand each check of a basic reconfiguration service is documented in the status field of each wizard page. Herein the identified basic reconfiguration services as well as all checks provided for the sequence of commands are described. The output files for these checks are also incorporated in Appendix G, Table 7 to Table 9.

Check for requirements of resources

The requirements of resources belong to those properties, which may be changed during the execution of a system evolution step. In Section 6.3 especially the FB types and the memory consumption have been identified as important properties. The evaluation wizard includes a check for both requirements, whereas some limitations have to be mentioned:

- ***Type library check:*** Based on the information about the incorporated basic reconfiguration services within the ECA it is possible to identify the types which are claimed by the ECA. The second part of this check concerns the evaluation of the available elements within the type library. In the evaluation wizard the data included in the description file of the control device are used. It is also possible to get this data by using the QUERY management command directly from the control device.
- ***Available memory check:*** In order to apply the check for sufficient available memory within the control device it is necessary to get this data from the control device. The demonstration control device does not provide this possibility. Therefore, the evaluation wizard only sums up the necessary memory based on the basic reconfiguration services included in the ECA.

8.1.4 Verification by model checking

The system integrity characteristics global and local consistency, active references, state management, and real-time constrained operation need to be evaluated by using verification by model checking for the reconfiguration sequence according to the results presented in Section 5.2.2. The procedure for model checking starts with the design of the appropriate models, then the formulation of specifications, and at last the verification by executing the model checking algorithm. This will be the guideline for the application of the check for above mentioned properties within the reconfiguration sequence.

Design of the system model

The basic elements of the system model need to represent the configuration of the control device, as described in Section 7.1. The top-level view of the system model used for verification is given in Appendix G, Figure 76. For the typical control example the control application is located within one thread (‘Thread_APP’), as there are no real-time constraints mentioned in the application. Furthermore the control application makes use of the timer; accordingly the callback function has to be included in to the model (‘TimerTHREAD’). The ECA is located within a separate resource of the control device, which is modeled by another application thread in the system model (‘Thread_RECONF’). As there are no real-time constraints used in the ECA, too, the two threads including IEC 61499 control logic are located on the same priority level. The module 'Scheduler' is capable to handle five different priority levels, whereas on the lowest priority three different threads may be located.

Modeling of non-Boolean variables

The model of the control application is given in Appendix G, Figure 77. As prerequisite for the given functionality in the typcial control example it is necessary to provide means for the handling of integer variables in NCES. Two possibilities exist for this purpose: the value of the integer variable may be represented as a number of tokens within a place or a set of places is used to represent the integer value. The first possibility has been chosen in Pang and Vyatkin (2007). An unsigned integer value is represented by the number of tokens and models for all basic numeric operations as well as comparison are described, whereupon a variable and a fixed number are taken into consideration for these operations. The identification of a certain value within a place is based on condition arcs with weights according to the value which needs to be detected. This fact is problematic for the modeling of the typcial control example, as it is necessary to model connections with arbitrary values of integer variables. Therefore for each possible value a condition with the appropriate weight would be necessary. Furthermore also signed values of integers should be possible. Therefore this work uses a representation of non-Boolean variables in a binary format. An integer variable with a range of values according to a 16 bit binary format is represented by 32 places. For each bit two places are used in order to model the values true and false. Based on this information any operation for such a variable can be modeled in the same manner as this is done within any microcontroller or binary processing unit. For the typical control example addition and subtraction as well as comparison operations were modeled for this kind of representation. As a consequence it is not possible to represent an integer data connection by only one condition arc. For each data input/output in IEC 61499 32 condition inputs/outputs have to be modeled in the NCES representation. These sets of condition inputs/outputs increase the effort for modeling, especially without the support for automatic NCES model generation, and result in extensive and bulky NCES models. But the complexity does not increase, since any operation can be based on well known principles of digital data processing.

Model of the initial state of the control application before the RECONF sequence

The model of the control application for the verification of the RECONF sequence has to include the initial state for the RECONF sequence. This situation is presented in Figure 73 and includes apart from the original control application also the FBs 'SUB_INT_TO_INTERNAL' and 'CHECK_INT_LESS' as well as their input connections. The event and data flow is modeled according to the guidelines presented in Sections 7.4.2 and 7.4.3. Each input event is assigned to a specific number, and the event dispatcher needs to differentiate between each of these numbers. The model of the event dispatcher is capable to distinguish *30* different events, which is sufficient for the given control application. As mentioned above, the influence of the network was not targeted for this evaluation attempt. In order to simulate a certain network behavior, the internal timer functionality is used. Therefore three different interfaces to the callback function 'TimerTHREAD' are part of the model. One interface is necessary for the FB 'TAKT' which triggers the execution of the control application. The other two are used to provide a start and a stop command (which would be received from the user interface) in order to model a certain start and stop time for the verification process. The necessary NCES modules have been incorporated into the model of the FB 'Start_sub'[24]. For the evaluation of the DSE this does not have an effect, as the

[24] This is only a choice of the author for modeling such a behavior. It would be possible to simply use the means provided by NCES to simulate points in time when the FB 'Start_Sub' does invoke an output event. In a more comprehensive model of the control device, a thread dedicated to the network interface would be included, which would introduce external events and the SIFB-ID of the corresponding SIFB into the event dispatcher. By

verification will be focused on the time in between the starting and the stopping of the calculations within the control application. The second part of the interrelation with the user interface, the FB 'CurrentVal_pub', is only represented by its execution time. The data connection is modeled as storage element, but the condition arcs from this storage element to 'CurrentVal_pub' is neglected for the sake of a better clarity in the NCES representation. As this data will not be used in a model of the user interface, this simplification does not have any effect on the evaluation of the DSE.

Model of dynamic reconfiguration within the control application

The model presented in Appendix G, Figure 77, includes the models for dynamic reconfiguration for the first part of the RECONF sequence which is related to the control logic (see the description given above in Section 8.1.2). This part is crucial for the correct behavior of the application. The second part is dedicated to the correct representation of data in the user interface and will be neglected. The following commands need to be represented:

- ***Deletion of the event connection 'CHECK_INT_GREATER.CNF' to 'E_PERMIT.EI':*** The module representing this event connection is of the type managed event connection (see Figure 26) which is initially enabled.
- ***Deletion of the Boolean data connection 'CHECK_INT_GREATER.Result' to 'E_PERMIT.PERMIT':*** The module representing this data connection is of the type managed data connection (see Figure 27) which is initially enabled.
- ***Reading of the internal variable within 'ADD_INT_TO_INTERNAL'***: The interface of the NCES model of this FB is enhanced by condition outputs connected to the places representing the internal variable. At any time the current value of the internal variable can be used.
- ***Writing of the internal variable within 'SUB_INT_FROM_INTERNAL':*** The internals of the NCES model of the FB is enhanced in order to provide the possibility to set the value of the internal variable by an additional source. This source is the enhanced interface of the FB, which provides condition inputs for another integer variable. Furthermore an input event for setting the internal variable to this value is used.
- ***Creation of the Boolean data connection 'CHECK_INT_LESS.Result' to 'E_PERMIT.PERMIT':*** On the one hand the module representing this data connection is of the type managed data connection (see Figure 27) which is initially disabled. But another enhancement is necessary, because the old data connection and the new data connection are connected to the same data input. Within NCES it is not possible to connect several condition outputs to the same condition input (similar to IEC 61499), because the value of the condition input would not be specified unambiguously. Due to the procedure defined in the ECA (which has been already proved to be correct in the system integrity characteristic dependent operation) there is always only one data connection active at the same time. An additional module only has to copy the currently active value of the data connection to the condition outputs which are connected to 'E_PERMIT.PERMIT'.
- ***Creation of the event connection 'CHECK_INT_LESS.CNF' to 'E_PERMIT.EI':*** In contrast to the above mentioned data connection, it is sufficient to model this event connection by a module which belongs to the type of managed data connection (see Figure 26) which is initially disabled. Only those event connections which are enabled will insert the input event to the event dispatcher.

using the timer for simulating these external events, the procedure of calling the event dispatcher is the same as with a separate thread for the network interface.

Each of the models for dynamic reconfiguration (except the reading of the internal variable) is triggered by an event signal, which has to be provided by the model of the ECA. For the READ command, the time when the basic reconfiguration service latches the internal variable determines the value which will be read.

Model of the RECONF sequence within the ECA

The ECA is modeled within the module 'Thread_RECONF', which is depicted in Appendix G, Figure 78. As the verification by model checking is restricted to the reconfiguration sequence, only the relevant parts are included in the NCES model. Each FB representing a basic reconfiguration service is modeled as discussed in Section 7.4.3 for any IEC 61499 FB, but with the difference that an event output for triggering the corresponding dynamic reconfiguration is added (except for the READ command). The transition of the state from 'ADD_INT_TO_INTERNAL' to 'SUB_INT_FROM_INTERNAL' should be done within the ECA, but for the sake of clearness this is modeled directly by interconnecting the output conditions representing the internal variable of 'ADD_INT_TO_INTERNAL' with the condition inputs dedicated for writing the internal variable of 'SUB_INT_FROM INTERNAL'[25]. Only the point in time when the write command happens is provided by the execution of the ECA. As trigger for the RECONF sequence we will use again the timer of the control device. This trigger may vary in order to include different system behaviors. For a more detailed analysis it can be represented by a general event occurrence behavior as already described in Section 7.6.3.

Specifications and evaluation

The following sections will provide an overview on the different specifications which have to be considered for the check of the reconfiguration sequence. Several possibilities exist for this evaluation according to the used tool framework Visual Verifier (ViVe) [61] (see also Section C.3):

- ***Use of the internal model checker:*** This model checker only enables specifications based on first order predicates (no temporal logic). In addition it is capable to provide the set of states which fulfills a given property (e.g., all state of the reachability graph where a specific transition fires).
- ***Use of SESA:*** The model checker SESA may be used to check specifications in temporal logic according to CTL, which also includes the possibility to use time intervals for the temporal operators **X**, **F**, and **U**.
- ***Visual verification:*** Based on the timing diagrams for paths within the reachability graph it is possible to visually verify if certain conditions are fulfilled by simply displaying the relevant places and transitions.

It has to be noted that during the evaluation with the given tool framework a limiting behavior of the model checkers has been found. Due to the comprehensive modeling approach, the number of states and transitions within the models (*6672* states and *10563* transitions for the typical control example as described above) is rather high. On the other hand also the use of discrete time increases the number of states in the reachability graph. For a very accurate model of the system a time step of *0,1 µs* is a necessary choice for one NCES time unit, but on the other hand the total length that needs to be taken into consideration depends on the control application. For the typical control example at least three cycles of computation should be incorporated, which results at least in a length of *0,9 s* and *0,9* million states in the reachability graph for only one path within the reachability graph (no idle time of the

[25] It has to be mentioned that this simplification neglects the latching of the internal variable which is read by the ECA. If also real-time critical behavior should be evaluated, the value of the internal variable has to be latched by the basic reconfiguration service.

microcontroller assumed), and without those states which are based on the modeled functionality). This big amount of states in the reachability graph, where each state consumes a not inconsiderable amount of memory due to the big amount of states and transitions in the model, results in the requirement that the model checking tool needs a very big amount of memory. This is not the case for the current version of ViVe and SESA, therefore the experiments with the typical example have to be restricted to a manageable volume of states for the model checking tool. The following restrictions have been formulated for the typical control example:

- ***Non-consideration of exact real-time behavior:*** The principle temporal order of actions has to be incorporated in the system model, but the exact time delay of actions will be substituted by virtual values in order to reduce the number of states in the reachability graph.
- ***Single execution path:*** The use of non-determinism in the system model provides the possibility to include a high variety of system behavior into the model checking process, which is one of the main advantages of the evaluation by model checking. But this also increases the reachability graph to a high extent; therefore we will consider only one single path within the system model.

In order to provide the verification of the given typical control example, we will use the internal model checker of the tool framework together with the possibility for visual verification. The reachability graph includes *11116* states when using the option "Maximum set of spontaneous" as firing rule (see Section C.3 for details about this setting). By using the following checks the correctness of the DSE has been proven. Even if we have restricted our considerations to a very limited model, we will describe the evaluation process in a comprehensive way.

Global and local consistency

The check for global and local consistency aims at the verification of the specifications of the control application (without taking care of the DSE). According to Hanisch (2004) we have to distinguish plant, process, and product specifications. As our typical control example does not use a plant and accordingly does not produce anything that may be specified, we only have process specifications which have to be checked. Different examples for process specifications related to the typcial control example are:

- "If the user interface sends a start command, the FB 'TAKT' has to send at least one output event some time afterwards." This property may be formulated based on the response property pattern (see Section E.2.2, Equation 57) as

$$\mathbf{AG}\ (\ p1251 \rightarrow \mathbf{AF}\ (\ p1267\ ), \tag{19}$$

  where *p1251* is marked during the issue of 'E_SWITCH.EO1' and *p1267* is marked during the issue of 'TAKT.EO'.

- "If the user interface sends a stop command, the FB 'TAKT' has to be set to its idle state some time afterwards." This property may be formulated in a similar manner based on the response property pattern as

$$\mathbf{AG}\ (\ p1245 \rightarrow \mathbf{AF}\ (\ p1254\ ), \tag{20}$$

  where *p1245* is marked during the issue of 'E_SWITCH.EO0' and *p1254* marks the idle state of 'TAKT'.

- "Each start of FB 'TAKT' has to be followed either by a regular stop due to a successful evaluation of the stop criterion (FB 'E_PERMIT' sends an output event) or a stop command from the user interface." Again the response property pattern may be used to formulate this property as

$$\mathbf{AG}\,(\,p1256 \rightarrow \mathbf{AF}\,(\,p3455 \vee p1245\,)), \tag{21}$$

where *p1256* marks the active state of ‘TASK’,
p3455 is marked during the issue of ‘E_PERMIT.EO’, and
p1245 is marked during the issue of ‘E_SWITCH.EO0’.

The normal operation of the typical control example has been evaluated by model checking based on the model given in Appendix G, Figure 79, which includes the same models but no ECA, by visual verification. For the model including DSE this has been done by visual verification, too.

Active references

Within the typical control example only references to the underlying services timer functionality and communication interface are included. As the DSE is not related to these elements, no special consideration of this system integrity characteristic is necessary. If these elements would have been involved, a check for the properties of the plant, product, or process would be sufficient, which has been provided already above.

State management

Within the control application the internal state of FB ‘ADD_INT_TO_INTERNAL’ has to be transferred to FB ‘SUB_INT_FROM_INTERNAL’ without any additional calculations (as this may be necessary for changes related for instance to a closed-loop controller). In general the transition management policy may be evaluated according to the influences on the plant. But as the typcial control example does not use a model of the plant, the evaluation of the system integrity characteristic state management has to be focused on the control application itself. The following criteria may be used:

- “After the execution of the ECA the internal variables of the two FBs ‘ADD_INT_TO_INTERNAL’ and ‘SUB_INT_FROM_INTERNAL’ need to have the same value”. The point in time when this criterion has to be fulfilled should be exactly after the execution of the corresponding basic reconfiguration service, in detail the finishing of the WRITE command within the ECA. A possible formulation would be

$$\mathbf{AG}\,(\,p6436 \rightarrow \mathbf{AX}\,((\,p2421 = p3593\,) \wedge \ldots \wedge (\,p2452 = p3624\,)), \tag{22}$$

whereas *p6436* is marked during the issue of ‘SET_FBINTVAR_INTER NAL.CNF’, *p2421* to *p2452* represent the internal variable within ‘ADD_INT_TO_INTERNAL’, and
p3593 to *p3624* represent the internal variable within ‘SUB_INT_FROM_INTENRAL’.

The property has been proved by visual verification.

Real-time constrained operation

The execution of the control application as well as the ECA in time is essential for correctness. In the given typical control example we have not included concrete real-time parameters in order to limit the number of states in the reachability graph. But the temporal order of commands was introduced by virtual values. An exact evaluation of the system integrity characteristic real-time constrained operation thus does not lead to the desired results. Nevertheless, a list of possible specifications regarding this property should be presented:

- “The execution of the calculations within the control application has to be finished before a new trigger occurs.” This property would be part of the global and local consistency properties if a real-time constrained execution would have been modeled with the event source ‘TAKT.EO’. This property may be formulated based on the absence pattern (see Section E.1.1, Equation 35) as

$$\mathbf{AG}\,(p1267 \wedge \neg\, p3376 \rightarrow \mathbf{A}\,[\,(\,\neg\, p1257 \vee \mathbf{AG}\,(\,\neg\, p3376\,))\,\mathbf{W}\,p3376\,]\,), \qquad (23)$$

where *p1267*, which is marked during the issue of 'TAKT.EO', represents the starting point of the considered time frame,

p3376, which marks the triggering of 'E_PERMIT' with the input event 'EI', represents the end point of the considered time frame, and

p1257 marks the triggering of 'TAKT' by the timer.

This property is not very sharp because the end of the execution differs according to the current value of the internal variable. If the result of the evaluation within 'CHECK_INT_GREATER' or 'CHECK_INT_LESS' is true, FB 'E_PERMIT' will send an output event in order to stop 'TAKT'. But if this is not the case, the execution will stop at 'E_PERMIT'.

- "The execution of the event chain triggered by 'TAKT.EO' has to be finished in a certain amount of time". Herein the same problem as described for the previous property occurs, a possible formulation would be (see also Equation 14)

$$\mathbf{AG}\,(\,p1267 \rightarrow \neg\mathbf{EF}_{[0,\,\alpha]}(\,p3376\,)), \qquad (24)$$

where *p1267* is marked during the issue of 'TAKT.EO',

p3376 marks the triggering of 'E_PERMIT', and

α represents the end of the time frame, e.g. 250000 as equivalent to *250 ms* (*0,1 μs = 1* NCES time step).

- "The execution of the RECONF sequence within the ECA has to happen within a given time frame." This property may be formulated similar to Equation 24 as

$$\mathbf{AG}\,(\,p5962 \rightarrow \neg\mathbf{EF}_{[0,\,\alpha]}(\,p6565\,)), \qquad (25)$$

where *p5962* marks the insertion of the event 'DEL_CONN_CNFEI.REQ' to the event dispatcher, which is the starting point for the execution of the RECONF sequence,

p6565 marks the end of execution of the FB 'CREATE_CONN_CNFEI', the last FB of the modeled RECONF sequence, and

α represents the end of the time frame, e.g. 100000 as equivalent to *100 ms* (*0,1 μs = 1* NCES time step).

Further improvements of the modeling precision

The above given description of the verification process for the reconfiguration sequence of the typical control example has been limited to only one single execution path within the reachability graph. Figure 47 depicts the execution of the different threads within the control device in the first three time diagrams. Every millisecond the callback function for the timer ('TimerTHREAD') is triggered and evaluates the actually registered timer FBs. If the counter value for the FB 'TAKT' has elapsed, the SIFB-ID of 'TAKT' is put into the event dispatcher of the thread corresponding to the control application ('Thread_APP'), which takes place every *300 ms* as configured in Figure 41. The trigger for the reconfiguration sequence may be provided within any time, as there is no special event dedicated as starting point. Within the above described model, the reconfiguration sequence is executed between two execution triggers for the control application.

In order to improve the quality of the verification process, several aspects may be considered in more detail. For instance, the trigger for the reconfiguration sequence may vary according to its prerequisites: It may happen at any time during the execution of the control application, because it is triggered via the user interface. Furthermore any other threads which may belong to the control device (Figure 47 mentions for instance additional programs in the forth timing diagram) may vary in their occurrence (see the general behavior patterns in Section 7.5) as well as the time consumption of the related actions. All these situations and their combinations can be incorporated into the system model for the verification, which will be used to check whether the reconfiguration sequence will be executed successfully or not. This situation matches with the introductory comments described in Section 1.1, whereas there is a high variety of situations which has to be taken into consideration for deciding whether a DSE may be successful or not. The above described modeling approach provides the basis to incorporate this variety into the system model and therefore represents a basis for a well-grounded decision.

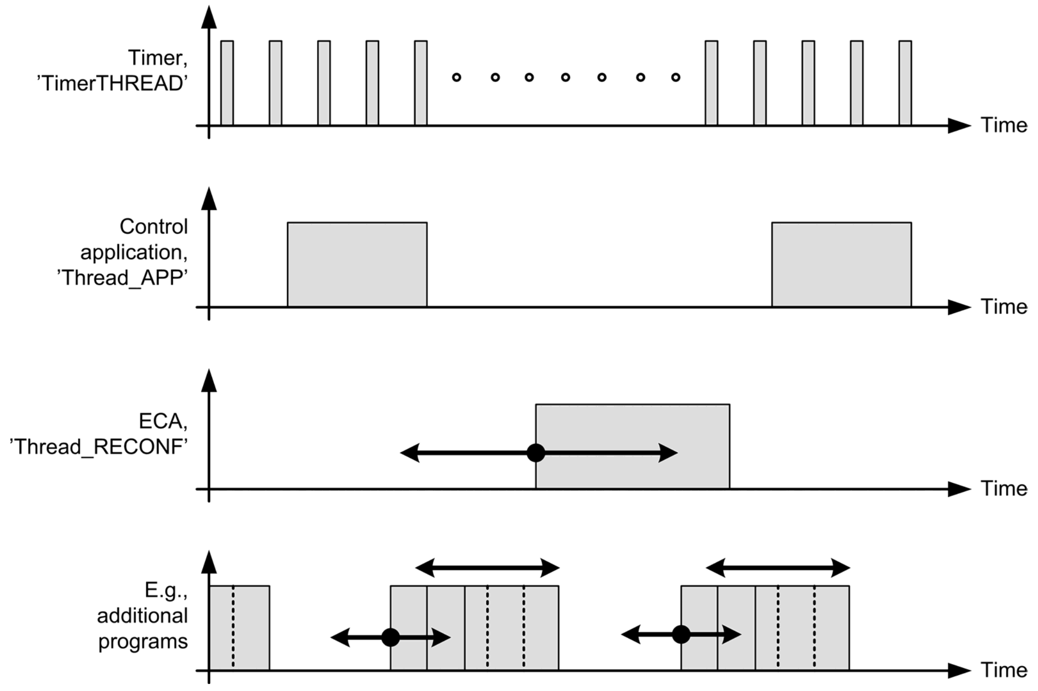


Figure 47: Overview on the execution of threads in the typical control example

8.2 Experiments with selected architectural elements

In order to give a more detailed description of the chosen modeling approach for the system architecture of a control device, we will present two examples of special situations that are considered without DSE. As mentioned within the last paragraph, the system model incorporates a high variety of possible system behavior. We will focus in the following two sections on the event dispatcher as the critical section as well as the model of a linear axis as the plant.

8.2.1 Event dispatcher as critical section

A very important concept within the implementation of R^3E is the event dispatcher and the insertion of events from different sources (FBs within the thread as well as external event sources). Therefore, the area of the event dispatcher has to be handled as a critical section within R^3E (see also Appendix B or Section 7.4.2). In order to verify the system behavior in the case of access to this critical section a simple experiment taking this aspect into consideration will be presented here. This example has been presented also in Sünder *et al.* (2008).

The control application for this experiment is depicted in Figure 48b. There are four event sources included in this small FB network: the 'E_RESTART' FB which provides the initial event for the application; the timed FBs 'E_CYCLE' and 'E_DELAY', and the 'SUBSCRIBE' FB. As there are no real-time constraints mentioned in the application, the whole application is mapped onto one single thread.

The computational architecture of the control device is provided in a simplified schematic in Figure 48a. The IEC 61499 control application is located in ‘THREAD3’ on the lowest priority of the operating system. There are two external events, which can be recognized by the control device: the timer interrupt and the network interrupt. Both are implemented as callback functions and are located on the highest priorities of the scheduler. Additionally, we assume two further threads that may be active on the control device. They are located in between of the external events and the IEC 61499 application in ‘THREAD1’ and ‘THREAD2’. The behavior of these two threads is modeled in a very abstract way by a typical execution time and activation behavior. We will consider especially the event dispatcher within ‘THREAD3’, which is interrelated with both external event source timer and network according to the control application.

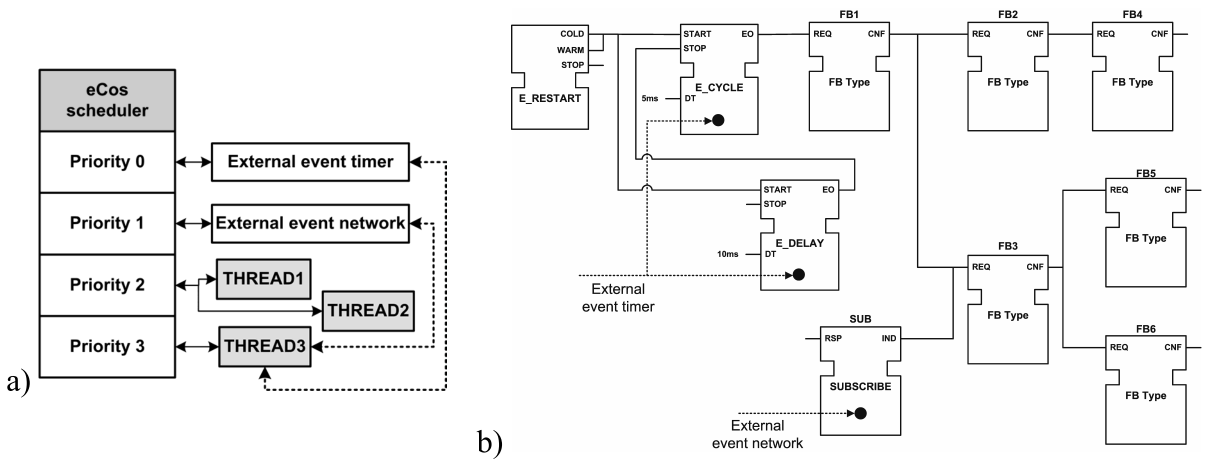


Figure 48: Example configuration a) of the control device and b) the control application in 'THREAD3'

Figure 49 depicts a shortened form of a path in the generated reachability graph. We have again mentioned only virtual values for the computation time of actions within the control device, but the temporal order of these actions is correct[26]. The path starts in the state when ‘FB1’ is executed because the timer interface has triggered the FB ‘E_CYCLE’. At this time also the external event network becomes active and interrupts the execution of ‘THREAD3’. Accordingly, the SIFB-ID for the ‘SUBSCRIBER’ FB is put into the event dispatcher. Exactly at the same time the event dispatcher is already in use (event ‘REQ’ of ‘FB2’ is put into the queue), ‘TREAD3’ is executed in the context of the network interrupt as long as the event dispatcher is free again. After the execution of ‘FB1’ has been finished, ‘FB2’, ‘SUB’ and then ‘FB3’ are executed. This is a notable result, since ‘FB2’ and ‘FB3’ are triggered by the same event – one can see that our NCES model correctly implements the sequential execution model of R^{3}E. During the execution of ‘FB3’, the timer interrupt has to be executed, but as no timed FB is ready to be triggered, this does not influence the execution of ‘THREAD3’. The event dispatcher includes now the input events ‘REQ’ from ‘FB4’, ‘FB3’, ‘FB5’ and ‘FB6’. During the execution of ‘FB6’, the timer interrupt occurs again and disrupts the execution of ‘THREAD3’.

[26] The time values in Figure 49 have to be considered as simplified values from different measurements with the demonstration control device.

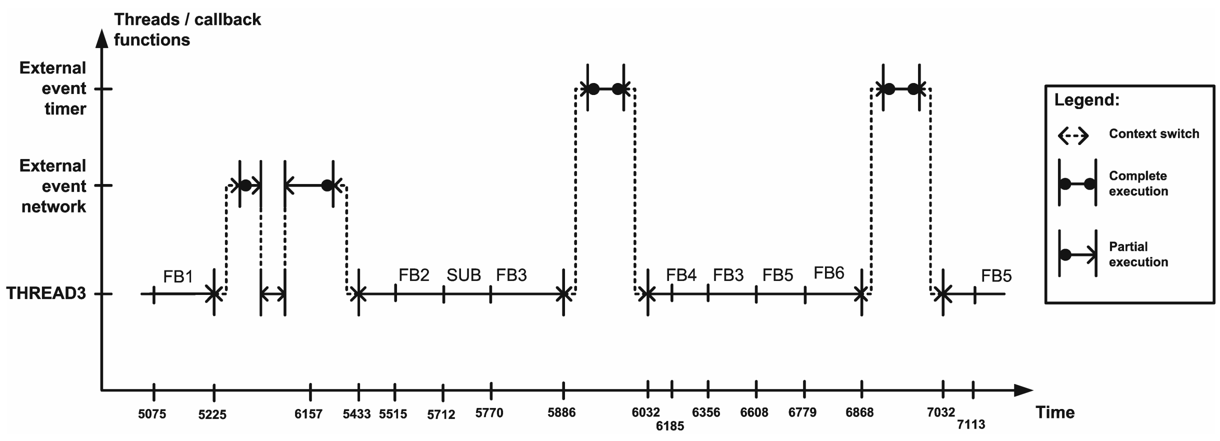


Figure 49: Excerpt of a path within the reachability graph of the control application within 'THREAD3'

8.2.2 Modeling the plant behavior

A very important part of the system model is the behavior of the plant which needs to be incorporated into the models of the control device. As an example for a plant model we will use a linear axis, which has been described in contrast to the application of DSE in Hanni (2007). In detail, the exchange of the closed-loop position controller was demonstrated in this thesis, which has been supervised by the author. Accordingly, the formal model of the plant has to describe the temporal behaviour of the movement of the axis. Herein appropriate descriptions, which usually already exist for the design of the closed-loop control circuit, need to be modeled by means of NCES. The linear axis used for this automation object can be described by the transfer function

$$G(s) = \frac{1}{1 + s \cdot 4{,}66 \cdot 10^{-4} + s^2 \cdot 5{,}2 \cdot 10^{-8}}, \tag{26}$$

which uses the current reference value as input and the force applied to the linear axis as output. As we are interested especially at the position control of the linear axis, we are able to simplify the overall model architecture by neglecting the details of the velocity control application. The model of the plant is enhanced by the velocity closed-loop control and has the velocity reference value as input and the current position of the axis as output. The introduction of this behavior into the NCES model can be achieved by a transformation into discrete time, as it is usual for sampled-data control systems.

Appendix F, Figure 80, depicts the NCES model used for testing the behavior of the plant model together with the closed-loop position control[27]. The model of the control device itself is neglected, and based on a clock which is used for calculating the current value of the plant model also the position control is calculated. The representation of the non-Boolean values for velocity and position of the linear axis has been chosen according to the description given above, as set of places. The NCES model of the position closed-loop circuit provides the same step response which has been simulated by using an appropriate model in the tool Matlab simulink. This result is depicted in Figure 50.

[27] The use of a simple model of the control device, including only the timer callback function as well as one thread for the control application (position closed-loop control) already exceeds the memory limits of the tool framework.

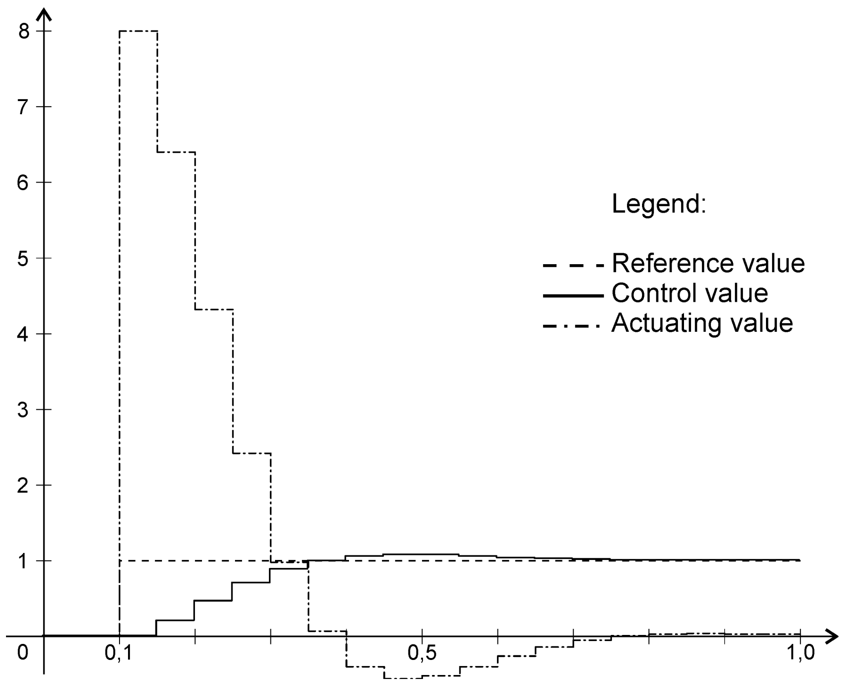


Figure 50: Result of the NCES model of a position closed-loop control

8.3 Summary

The proposed methodology for the modeling and evaluation of DSE has been demonstrated by using a typical control example within this chapter. Therefore, several preconditions have been described as starting point for the demonstration:

- The control application, which includes a simple integration of a given value. This value changes every cycle of the execution. This is a usual situation for any closed-loop control circuit.
- An evolution scenario, which includes the exchange of the addition of the given value by a subtraction, whereas also transition management has to be taken into consideration for the preservation of the integration variable. The same procedure has to be applied for the exchange of a controller or a filter in the feedback loop of a closed-loop control circuit.
- The representation of the demonstration control device by its KAPPA vector, which has been provided as a FDCML-based description file.
- The formal models for the different parts of the control device as well as the control application, whereas special attention has been paid to the representation of integer variables within NCES.

The evaluation of all system integrity characteristics has been described for the given typical control example. The KAPPA-based calculations have been incorporated into an evaluation wizard within the εCEDAC engineering tool. The verification by model checking has been based on a tool framework without the automatic generation of models. Herein as important requirement for the model checking tool the high amout of necessary memory has been pointed out. The given tool framework was not capable to handle both, the detailed models of the control device and the representation of real-time behavior with a fine grain time scale. Therefore, the evaluation results with respect to real-time behavior have only principal character. Based on the system model a high variety of scenarios for the execution of DSE can be taken into consideration (especially the various combinations of these scenarios) which will provide a significant basis for the evaluation process.

The described typical example of a DSE has been implemented and executed on the given demonstration control device in the Odo Struger Laboratory at ACIN. The successful evaluation of the DSE has been proved by its execution on the physical hardware, which has been applied successfully, too. Further different failure scenarios have been occurred during the engineering of the demonstration example, which have been detected during the evaluation process. For instance, type mismatch of FB instance names within the ECA have been detected during the check for dependent operation. Although the set of test scenarios has been very small, the demonstration example shows that the successful evaluation corresponds with the successful execution of a DSE. Further tests need to be applied in order to prove this new methodology in more detail.

In order to give also an impression of details within the system model of the control device, further experiments regarding the access to the critical section event dispatcher and the plant model of a linear axis are taken into consideration.

Chapter 9

Discussion of Industrial Application

This thesis proposes a new methodology for modeling DSE within an ACS and provides the evaluation process in order to check that the system will not produce disturbances to the process under control or even break down. A crucial point for DSE is the application in industrial practice, whereupon especially the ACS customer has to be taken into consideration (see also the introductory comments in Chapter 1).

We will split up our considerations into two parts, which are motivated by the following questions:

- How can the additional effort necessary for DSE be introduced without enormous endeavors?
- What kind of engineering can be established based on the possibility to use DSE in ACSs?

The first question is highly related to the kind of knowledge, which the engineers have within the different parties of an ACS. If an ACS customer has to create the formal description of e.g. the operating system, which is part of the control devices within the ACS, this methodology will not be used in industrial practice, because the additional effort as well as the principal feasibility is ignored. Therefore, we will discuss the interrelation of DSE and its evaluation in regard with the general structure and roles of vendors in ACSs in Section 9.1.

The second question aims at a more general consideration of the possibilities provided by DSE. We will consider a methodology well-known in computer science, which takes continuous changes and early operating software as basis for the engineering process in Section 9.2.

9.1 Value-added chain for total evaluation

DSE provides the capability to keep a plant in operation even if changes have to be applied. There is no need for often highly expensive ramp down and up procedures within the plant, because the plant will stay operational all the time. But these benefits have to be paid with additional effort in the engineering process. On the one hand the ACS needs to have the capability to model and execute these changes during operation. And of course it is on the other hand of eminent importance to check if a system evolution step will not produce errors, which may create even more costs than the ramp down and up procedure.

In order to discuss the responsibilities for the establishment of the basis for the evaluation process, the exhaustive description of the different parts of the control device and its integration into the engineering cycle, we will recall the general structure and roles of vendors in ACSs, which has been already introduced in Section 3.1 based on Vyatkin *et al.* (2005). The basic idea of this so-called value-added chain is that each of the companies provides its specific expertise to the ACS. Of course, the added value represents the basis for the business model of these companies, but on the other hand also a highly efficient engineering cycle is possible based on this structure. The company, which is most related to a topic, provides the solution for problems related to this topic. For instance, the component vendors provide the software for the control of their components, which are typically actuators and sensors. The machine vendor has its special expertise in the overall machine and the concepts for its operation, and does not need to take care about the control for each single sensor or actuator. This model holds also for the overall industrial enterprise, which coordinates different systems provided by system integrators (see also Figure 3).

If we put this structure as basis for the question: "How can the additional effort necessary for DSE be introduced without enormous endeavors?", the answer is simple. Those companies within an ACS, which are closely related to the necessary information for the evaluation of DSE, will incorporate this effort into their business activities. Figure 51 depicts this *value-added chain for total evaluation*, whereupon the effort for the different roles in an ACS is described briefly. The ACS customers, which are part of the roles component vendor, machine vendor, system integrator, and industrial enterprise, have to provide only the information which is related to their special expertise:

- ***Description of the software functionality:*** Each of the companies within the different levels adds functionality to the overall ACS. This functionality is represented in software and hardware behavior. The software functionality has to be described according to the rules given by the architecture of the used control device. According to Figure 24 especially the control and other applications, additional tasks, and their parameterization with real-time behavior have to be provided in terms of formal models and enhancements to the KAPPA vector of the related control devices. These companies create the software functionality; correspondingly they also have the responsibility to provide the necessary information for the evaluation process about these parts.

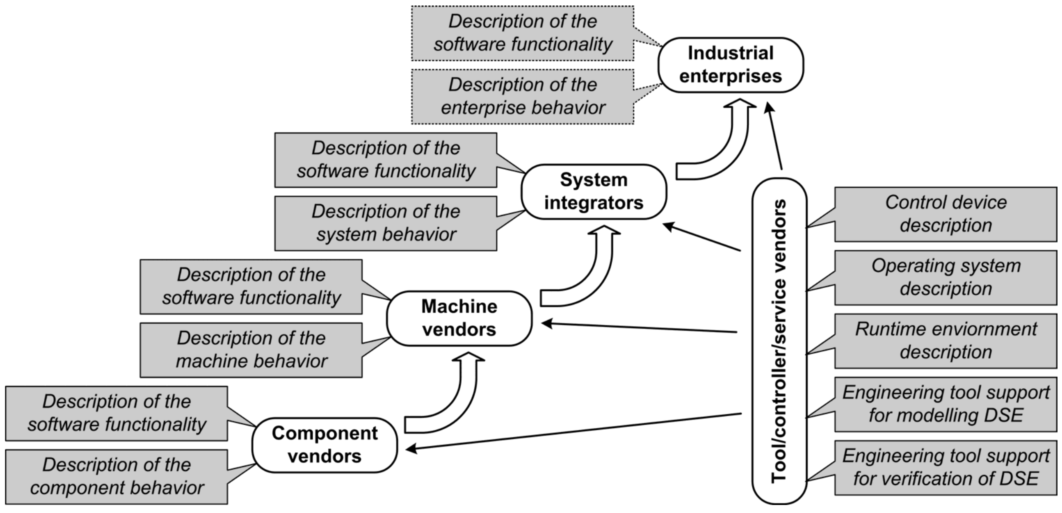

Description of the software functionality
Description of the enterprise behavior
Industrial enterprises
Description of the software functionality
Description of the system behavior
System integrators
Description of the software functionality
Description of the machine behavior
Machine vendors
Description of the software functionality
Description of the component behavior
Component vendors
Tool/controller/service vendors
Control device description
Operating system description
Runtime enviornment description
Engineering tool support for modelling DSE
Engineering tool support for verification of DSE

Figure 51: The DSE expertise of the different roles in ACS: the value-added chain for total evaluation

- ***Description of the behavior:*** Next to the pure software functionality also the behavior of the component, machine, system or even enterprise has to be described as input for the evaluation of DSE. Herein especially the hardware in terms of parts of the plant is taken into consideration together with the related plant behavior. For instance, the model of the linear axis described in Section 8.2.2 has to be provided by a component vendor. A machine vendor may use these models and adds the behavior of the machine which is in conjunction with the linear axis.

In order to efficiently use such a modular engineering approach for the description of components it is necessary to standardize the interfaces between the different components. As an example the interface for different tasks within the formal model of the operation system has been defined in Section 7.4.1, so different companies may put together their software functionalities in terms of tasks without taking care of each other (except interrelations exist between tasks, as for instance in the case of external event sources, which again may be standardized).

The special role of tool, controller, and service vendors

The main part of the control device description for the evaluation of and support for DSE has to be provided by tool vendors, controller vendors and service vendors. They take the central role within the application of this new methodology in industrial practice. Their responsibilities can be summarized as follows:

- ***Engineering tool support for modeling DSE:*** The basic prerequisite for DSE is an engineering tool that provides this new engineering methodology. The general description of the concept for modeling DSE has been given in Chapter 4. Next to the basic capability to model ECAs the tool support is one of the most important aspects for the acceptance by ACS customers. The detailed information represented in the KAPPA vector of the system can be used as basis for enhanced engineering support for DSE as well as modeling of control applications, e.g., automatic communication configuration, provision of template libraries, and support for the selection of the most appropriate template for a given change within the control application.
- ***Engineering tool support for the evaluation of DSE:*** Next to the overall engineering cycle for DSE especially the integration of the evaluation process plays an important role. The evaluation of properties for both KAPPA-based calculations and model-checking should be available in the same environment as the modeling of a system evolution step. For instance, the specification of properties may be enabled in terms of the programming language, e.g. events and data of IEC 61499 FBs.
- ***Operating system description:*** The exhaustive description of the operating system belongs to the provider of the RTOS. Herein next to models for the formal description of the RTOS also the basics for the KAPPA-based calculations concerning scheduling bounds, parameterization, and memory management have to be part of this description.
- ***Runtime environment description:*** Similar to the operating system the exact behavior description of the concepts implemented in the runtime environment regarding to the execution and dynamic reconfiguration of control logic need to be provided by the company who sells the runtime environment (in most cases this is the same company which provides also the engineering tool).
- ***Control device description:*** The control device is created from the different architectural elements as for instance the operating system or the runtime environment. The overall description of these elements, which have to be parameterized and enhanced according to the special configuration of the control device, has to be provided as a description file. The engineering tool should be capable to handle these type descriptions of the control device and utilize the information in the KAPPA vector, which includes the concrete instances of the different control devices.

As a unifying element for the different elements and companies the engineering tool acts as common basis. There will be companies who provide single parts that are composed to overall descriptions by other companies (e.g., the operating system vendor and the control device vendor), which is related to the development of the basic infrastructure of ACSs. And on the other hand there will be companies utilizing the given information in order to design more complex ACSs, e.g. the component and machine vendors. These companies are the applicants of ACS technology. Based on these two different viewpoints within a system, the occurrence of the engineering environment and the support of the engineering tool may differ.

9.2 Automation Extreme Programming

The possibility of keeping the system always running with the flexibility to adapt the system's functionality at any time is a big advantage of DSE. In order to utilize this advantage it is necessary to apply this new methodology according to certain rules, otherwise the system may become unmanageable (see also the discussion about software evolution in Section 3.4).

From the field of computer science many different concepts and paradigms for software development have been proposed. One of these concepts is *eXtreme Programming* (XP), which concentrates on a highly flexible design flow and changes of program functionality at any time in the engineering cycle. But it is not intended for applying changes during operation. Based on the principles of XP and the capabilities of DSE, we will develop a new paradigm for system design in ACSs.

Extreme programming

XP was developed by Kent Beck and does not introduce new concepts of software design. It represents a summary of common sense principles of software development, trying to strengthen their individual benefits on the one hand and to diminish the drawbacks by a compensation with capabilities of other principles on the other hand. A comprehensive introduction to XP is provided in Beck (2000). The different practices summarized within XP are listed in Beck (2000, Chapter 10) in the following way:

- "***The planning game***—Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan."
- "***Small releases***—Put a simple system into production quickly, then release new versions on a very short cycle."
- "***Metaphor***—Guide all development with a simple shared story of how the whole system works."
- "***Simple design***—The system should be designed as simple as possible at any given moment. Extra complexity is removed as soon as it is discovered."
- "***Testing***—Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished."
- "***Refactoring***—Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, and add flexibility."
- "***Pair programming***—All production code is written with two programmers at one machine."
- "***Collective ownership***—Anyone can change any code anywhere in the system at any time."
- "***Continuous integration***—Integrate and build the system many times a day, every time a task is completed."
- "***40-hour week***—Work no more than 40 hours a week as a rule. Never work overtime a second week in a row."
- "***On-site customer***—Include a real, live user on the team, available full-time to answer questions."
- "***Coding standards***—Programmers write all code in accordance with rules emphasizing communication through the code."

As a summary, "XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements" (Beck, 2000, Preface). Change is a common element within the methodology. As soon as e.g. simplification is possible, the corresponding part of the software will be changed. This has to be considered also from the economical point of view, as typically the cost of changing a program is expected to rise exponentially over time (Figure 52, upper part). As a consequence of this schematic, changes will be applied only in the early development phase because of the high costs in the later ones. XP expects a very different schematic for the cost of change, as depicted in the lower part of Figure 52. The cost of change rises much slower over time, eventually reaching an asymptote. "If the cost of change rose slowly over time, you would act completely different from how you do under the assumption that costs rise exponentially. You would make big decisions as late in the process as possible, to defer the cost of making the decisions and to have the greatest possible chance that they would be right. You would only implement what you had to, in hopes that the needs you anticipate for tomorrow wouldn't come true. You would introduce elements to the design only as they simplified existing code or made writing the next bit of code simpler." (Beck, 2000, Chapter 5). The different practices summarized in XP make this vision possible.

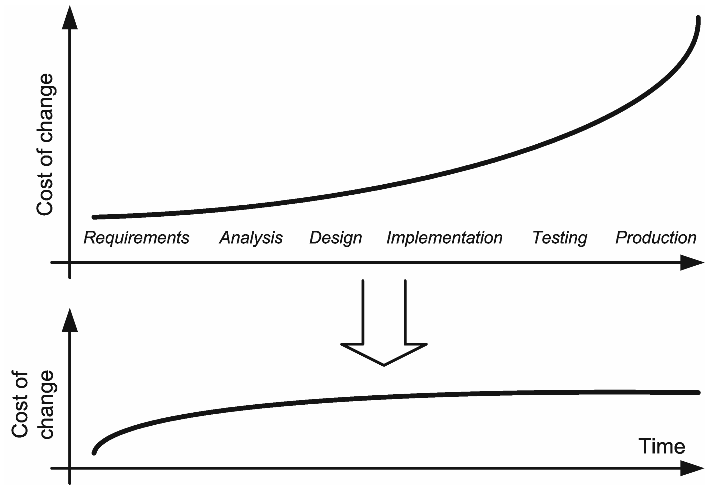


Figure 52: Cost of change in classical software project (top) and as premises of XP (bottom), Beck (2000, Figures 1 and 3)

Automation extreme programming

If we consider the typical engineering practice in ACSs, there seems to be only little similarities with software development known from computer science. Apart from completely different programming languages especially the interaction with real hardware is the most important aspect that has to be taken into account in ACSs. The functionality is dictated by the plant and its hardware capabilities; the control logic is responsible to keep the process under control. Of course, the software becomes an even more important part of the overall system, as for instance depicted in Bouyssounouse and Sifakis (2005, Section 28.2) for the mechatronics industry. Changes to the control application and also the hardware of the plant are related to high costs since the plant has to be stopped during the application of these changes usually. Each stop has to be scheduled and is often related to a significant time consumption for ramping down and up the plant.

The methodology of DSE changes the general assumption for the engineering of ACSs: changes to the system are applied during operation of the plant and (in the best case) with no disturbances to the process under control. The main hindering reason, high costs related to changes to the system, are neglected since there is no need to stop the operation of the plant. As an extension to pure dynamic reconfiguration, DSE incorporates also hardware into the scope of changes. The engineering cycle provides a clear guideline as well as a methodology for the evaluation of these changes. But the overall engineering cycle of the system design has to be adapted to these new possibilities, too. For this reason, the practices of XP may be used as basis for a new kind of engineering paradigm, the so-called *Automation eXtreme Pro-*

gramming (AXP). The following aspects of XP have to be adapted (all other practices should be used, too):

- ***Small releases***—Due to the possibility to apply changes at any time, the simplest version of a system can be the starting point for operation. Of course, due to the interrelation of hardware and software the volume of the first operating plant highly depends on the kind of process under control, but it is possible that also the hardware itself may be changed (and enhanced up to the final configuration) during operation. Typically any plant will be optimized to a high extend during the first phase of operation. The practice of small releases brings the engineer to think of the most important functionalities at first, receiving feedback from its operation already before all details of the plant (hardware and software) are designed. Therefore, optimization is done naturally during the development of the system by establishing the final configuration based on small releases during operation.
- ***Continuous integration***—Based on the engineering cycle for DSE each change of the system functionality will be applied immediately to the overall system.
- ***Simple design*** and ***refactoring***—There is no need to think about highly sophisticated functionalities until the point in time when this functionality should be incorporated in the next release. Therefore, the design will be kept as simple as possible. In addition, feedback from the operating system is available during the whole engineering process. The optimization of the plant is already part of the engineering process instead of an extra part afterwards. This feedback has to be used to reduce unnecessary complexity within the plant.
- ***Testing***—For software development the impact of testing is very high since in most cases it is simple to execute these tests automatically. For ACS the interrelation with hardware makes testing more complex or even not possible without information on the hardware behavior. Anyway it is important to follow the main idea of testing: describe the proposed functionality before starting programming. For DSE in addition it is necessary to define the disturbances to the system which should be neglected during the execution of a system evolution step. The scope for testing and the specification of tests will be enhanced: from testing of functionality to a definition of properties for the plant under operation as well as the system evolution steps. According to the engineering cycle for DSE, testing is necessary for both, the proposed new system state and the ECA.

By the use of this new methodology for the overall system design, the sketch for the engineering of ACSs with DSE depicted in Figure 9 gets more substance and clear guidelines for its application. AXP clearly states that the first version of the plant, which will be put in operation, is the simplest version with reduced functionality. Only the most important features will be part of this first version. Any further steps towards more functionality and the final version of the plant, the single releases, will be planned incorporating both the new system state and the ECA in order to reach this new state. AXP provides guidelines about the volume of changes and the procedure to define the priorities of changes. By evolving the plant from this very simple initial state to the fully functional final state of the plant (whereas it has to be stated that the evolution will take place over the whole life cycle of the plant), the total engineering time should be decreased to a large extent. Additionally, since the cost of change does not rise exponentially, the overall system costs will be low, too. Adaptive manufacturing becomes the normal situation, as already the initial engineering of the plant is based on the principle “change during operation at low costs”.

9.3 Summary

The acceptance in industrial practice is of crucial importance for new concepts in ACSs. We have considered two different aspects for the industrial application of DSE.

The provision of the necessary information concerning all elements within the ACS, the control devices, the plant, and the internals of each architectural element imposes additional effort to the engineering process. According to the role of the different companies within ACSs, a clear relation of responsibilities for the different aspects of descriptions can be established. As main guideline, those parties which are concerned with a topic also have to provide the necessary additional information for the DSE methodology.

Another aspect is the incorporation of the capabilities from DSE for the engineering of the overall plant. It is necessary to interrelate the new functionality of changes during operation to clear guidelines for its application within the engineering of the overall plant. The concepts of XP, which already put change into the center of engineering, have been adapted to the prerequisites of ACS with DSE. As a result, an improved engineering of ACSs with operation of a first version of the plant as early as possible and continuous enhancements during operation has been established.

Chapter 10

Towards Evaluation of logi.CAD εCEDAC Instant reload

The evaluation of DSE has been discussed based on the IEC 61499 standard and the basic reconfiguration services provided by R^{3}E. But the industrial practice is dominated by control devices based on the IEC 61131-3 standard. As already depicted in Section 3.2.1 no dedicated interface exists in order to modify control applications during operation for IEC 61131-3 (2003). Many IEC 61131-3 based engineering tools and runtime environments provide a proprietary interface based on the principle of changing the control logic between execution cycles (see Section 3.4.3). The procedure of changing the control logic during operation works almost automatically, but without any possibility to apply transition management policies. Typically only the internal states of POUs, which have not been changed between the current system state and the new system state, will be restored.

The company logi.cals Austria (former kirchner SOFT GmbH) [29] provides the IEC 61131-3 compliant engineering tool logi.CAD which includes the possibility to change the control logic during operation by using of the function Instant reload. Based on the results of the research project εCEDAC [8], where kirchner SOFT GmbH has been a consortium member, the existing functionality of Instant reload has been enhanced in order to achieve also DSE (called εCEDAC Instant reload). We will provide a short overview on the functionality of these two kinds of exchange of the control logic within this IEC 61131-3 based control system and discuss the application of the evaluation method proposed within this thesis for logi.CAD with εCEDAC Instant reload[28]. The information about the logi.CAD internal functionality is based on personal discussions with Thomas Baier, Heinrich Steininger, and Mario Semo from the company logi.cals Austria.

10.1 logi.CAD Instant reload

The Instant reload functionality of logi.CAD provides the possibility to change any element within the IEC 61131-3 control logic (functions, function blocks, programs, data types, tasks, resources) within a given project. The changes are applied abruptly in between the execution cycles of the different tasks at the runtime environment. But the granularity for changes modeled in the engineering tool and those parts exchanged within the runtime environment may differ. Within the runtime environment only the binary deployment of a resource is taken into consideration. For instance, if the only change within the current system state and the new system state is a new data connection, the overall binary deployment of the resource will be exchanged. From the ACS customer's point of view this behavior is not visible, because the Instant reload mechanism takes care of internal states of the elements within the resource. If the same element exists in the new system state all internal states will be restored (internal

[28] Within this chapter we will use only the terminology of IEC 61131-3. If we use the terms resource or function block, they are used as defined within IEC 61131-3. This is in contrast to the previous chapters which use the meaning based on the IEC 61499 standard.

states may be variables or active states within an SFC). A rough description of the Instant reload functionality within logi.CAD is provided in logi.cals Austria (2008).

Figure 53a depicts the procedure executed within the runtime environment during the Instant reload. First the binary deployment of the new resource is loaded and its feasibility is checked. The engineering tool generates special code based on the current system state in order to perform a change during operation; therefore the consistency with the current resources operated in the runtime environment has to be evaluated. Up to this point in time the old resource is still executed. Then the old resource is stopped, the internal states will be restored and the new resource will be started. The point in time when these actions are performed is determined based on the execution cycles of the different tasks within the resource. Between the actions 'Stop old resource' and 'Start new resource' no control logic is executed, neither the old nor the new resource. This time is crucial for the disturbances to the execution behavior of the control device.

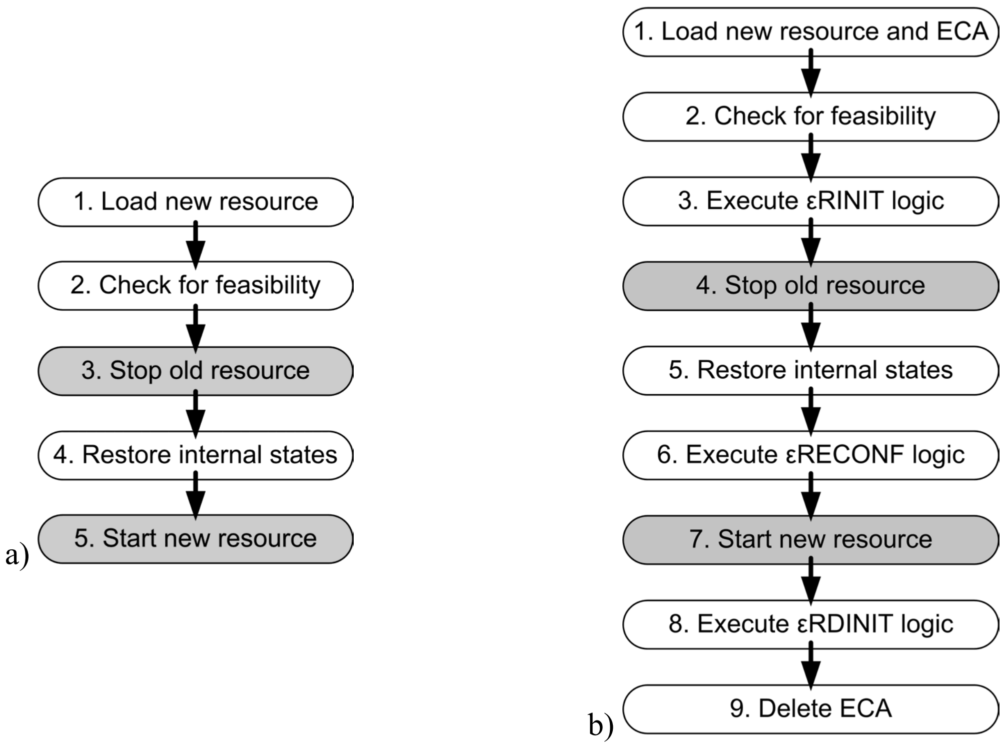


Figure 53: Comparison of a) Instant reload and b) εCEDAC Instant reload mechanisms within logi.CAD

10.2 logi.CAD εCEDAC Instant reload

The enhancement of the above given procedure according to the engineering cycle developed within the εCEDAC project is based on the different execution phases of a system evolution step, whereas the incorporated actions may be different. As depicted in Figure 53b the mechanism for applying changes has been enhanced by further steps that enable additional calculations for a DSE. Next to the new resource also an ECA is downloaded, which includes three different parts of the control logic: εRINIT, εRECONF, and εRDINIT logic. The execution of these three parts is sequential and in addition related to the actions provided for an Instant reload. After checking the feasibility of the new resource and the code for recovering the internal states from the old resource, the εRINIT logic is executed in parallel to the old resource. The set of commands in order to interact with the current system state is limited to READ and WRITE FBs (from the runtime's point of view apart from the exchange of the resource not more basic reconfiguration services are possible than READ and WRITE). Accordingly this sequence can be used to fetch any internal state of the old resource and also to provide some calculations for the transition management. Then the already described

Instant reload operations ‘Stop old resource’ and ‘Restore internal states’ are applied. But before ‘Start new resource’, the εRECONF logic is executed. Herein user-defined algorithms can be executed in addition to the already restored internal states of the resource, which is especially necessary for new elements within the new resource (the automatic restoring of internal states only includes unchanged elements). When the εRECONF logic has been executed (signaled by a Boolean value) the new resource is started and the critical phase where no control logic is executed is finished. The εRDINIT logic is executed next (in parallel to the new resource), providing the possibility to do some user-defined actions after the change. The εCEDAC Instant reload procedure is finished by the deletion of the ECA.

Figure 54 depicts a screenshot from an ECA within the logi.CAD engineering tool for a simple example of the exchange of a controller of a closed-loop control circuit (similar to the example presented in Section 4.2.3). The controller ‘SpeedCtrl’ within the program ‘CTRL’ will be changed during operation by another controller of a different type with the same name (it is assumed that a controller with a proportional part is exchanged by an enhanced controller including also an integral part). In contrast to the description given above it is possible to use similar names for the controller, because they belong to different resources within the εCEDAC Instant reload mechanism. The three parts of the ECA provide the following functionality:

- Within the εRINIT logic the current values of the gain ‘Kp’ and the control deviation ‘Inp’ are read and stored in the local variables ‘KpOld’ and ‘InpOld’. This happens while the old resource is still executed. The ECA is triggered by using the FB ‘εRINIT’, which provides a Boolean output set to true as soon as the εRINIT logic has to be executed. The finishing of the εRINIT logic is signaled by a Boolean input value of the FB ‘εRECONF’ set to true.
- Based on the input value of the FB ‘εRECONF’ the internal procedure of stopping the old resource and restoring the internal states is triggered. Afterwards the output value of ‘εRECONF’ is set to true and the εRECONF logic is executed. Herein the integral parts of the new controller ‘SpeedCtrl’ for the previous cycle ‘IOLD’ as well as for the current cycle ‘I’ incorporating an adaptation of the controller gain ‘KpNew’ are set. By the issue of a true value to the input of FB ‘εRDINIT’ the εRECONF logic is finished.
- Within the εRDINIT logic (triggered by a true value at the output of FB ‘RDINIT’) nothing happens and the ECA is finished by the issue of a true value to the input of FB ‘εDONE’.

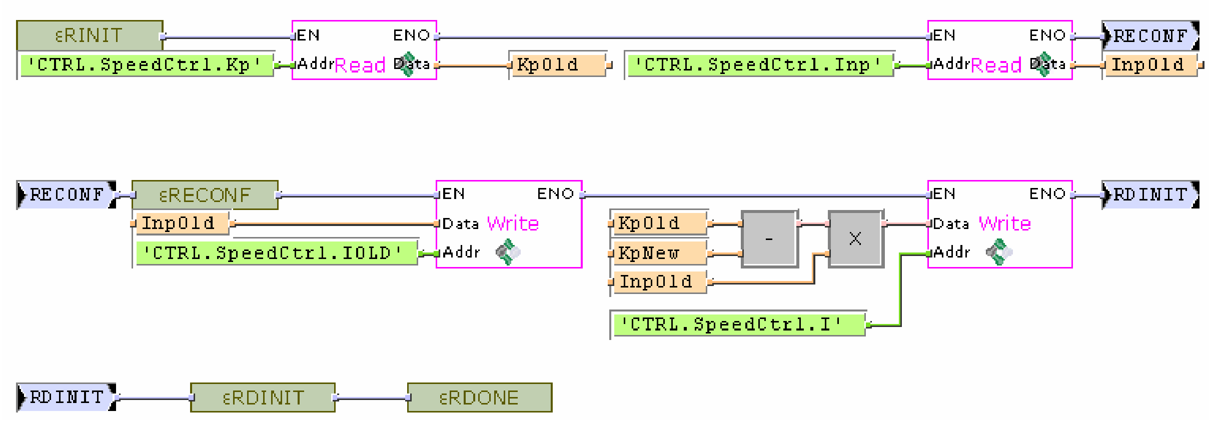


Figure 54: Example of an ECA in logi.CAD

10.3 Evaluation approach for the logi.CAD εCEDAC Instant reload

The application of an evaluation for DSE for an IEC 61499 based system environment has been presented in this thesis. In order to apply the evaluation methodology for a different system environment, as for instance the IEC 61131-3 based logi.CAD, the same steps need to be achieved:

- Analysis of the system properties in order to identify the possible change types according to Walsh *et al.* (2007b).
- Mapping the evolution modeling methodology.
- Identification of evaluation means necessary to check the involved system integrity characteristics.
- Formulation of algorithms and formal models for the different evaluation means.

We will discuss the first three tasks for the logi.CAD εCEDAC Instant reload within this section. A concrete formulation of the evaluation means need a detailed analysis of the internals of the system environment.

Analysis of system properties

The theory presented in Walsh *et al.* (2007b) is based on a component-based software development policy and the different change types are related to a component framework. As discussed in Section 3.2.1 it is not possible to consider the elements of IEC 61131-3 as software components apart from the element function. The most important hindering reason is the use of an interface description which only defines variables without details about the behavior of this interface. This is especially problematic for dynamic reconfiguration within an IEC 61131-3 based system such as logi.CAD.

According to the short introduction of the mechanisms logi.CAD Instant reload and logi.CAD εCEDAC Instant reload two different levels of changes exist that need to be taken into consideration. On the one hand the ACS customer may change any kind of element within a configuration and define adaptations to the system with fine granularity. On the other hand the runtime environment provides only changes of resources. If any adaptation is applied to the system under operation, always the overall resource will be exchanged. We will consider the change types of logi.CAD based on the capabilities of the runtime environment, because the evaluation of DSE should be as close as possible to the real implementation. Figure 55 depicts the interrelation of different change types according to the capabilities of the logi.CAD runtime environment. The main element is the substitution of resources, which may use any internal change defined in the engineering tool for any element of the resource. The interface of a resource is described especially by VAR_EXTERNAL, and based on the substitution of a resource also the interface may be changed. The overall change of configurations can be described as architectural changes.

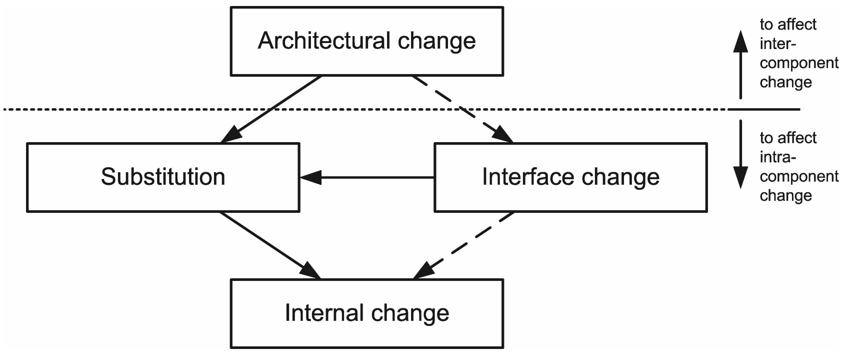

Architectural change
to affect inter-component change
Substitution
Interface change
to affect intra-component change
Internal change

Figure 55: Change types within logi.CAD εCEDAC Instant reload

The dashed arrows in Figure 55 represent logical dependencies between change types (as described in Section 3.4.1), but as the basis for dynamic reconfiguration is the exchange of resources they can not be used for modeling an ECA directly. The runtime environment provides only a small set of basic reconfiguration services, with limited possibilities for the usage within an ECA. But based on the concept of exchanging resources, this small set is sufficient for any change within the system:

- ***Exchanging resources:*** This command is integrated into the engineering tool and is not available as FB within control logic.
- ***Read:*** A separate FB for reading any variable within a resource may be used within the ECA.
- ***Write:*** A separate FB for writing any variable within a resource may be used within the ECA.
- ***Evolution Control FBs:*** Furthermore the FBs εRINIT, εRECONF, εRDINIT, and εRDONE exist which are used to define the borders of the different parts of the ECA in terms of defining the start and end point of the execution (see Figure 54 mentioned above).

Mapping the evolution modeling method

The evolution modeling method presented in Chapter 4 describes the use of an ECA for the modeling of a system evolution step. In addition, different ECAs may be encapsulated within an EECFB and so-called CECAs may be modeled in order to define the synchronization between different system evolution steps. The implementation of the logi.CAD εCEDAC Instant reload provides the possibility to model ECAs, CECAs are not supported. The different execution sequences within a system evolution step described in Section 5.2 can be mapped to the logi.CAD εCEDAC Instant reload methodology as follows:

- ***Download ECA:*** This step is similar in the logi.CAD εCEDAC Instant reload.
- ***Initialization sequence:*** The concept of the logi.CAD εCEDAC Instant reload is based on the exchange of a resource. There will be no actions performed within the ECA to change the old resource. The initialization sequence is executed by the engineering tool by downloading the new resource to the runtime environment.
- ***Reconfiguration sequence:*** The reconfiguration sequence is split up into two parts, the εRINIT logic and the εRECONF logic. The first one is able to work on the old resource, and the second is only capable to influence the new resource. Any transfer of state information for new elements of the resource is modeled within the ECA. All other state information is recovered automatically by the mechanism already used for the logi.CAD Instant reload.
- ***Deinitialization sequence:*** The εRDINIT logic provides the possibility to edit the new resource after the reconfiguration sequence. But based on the limited set of basic reconfiguration services only states within the new resource may be influenced. The deletion of the old resource is done automatically.
- ***Delete ECA:*** Together with the deletion of the old resource also the ECA is deleted automatically in the logi.CAD εCEDAC Instant reload mechanism.

Identification of evaluation means

The above given mapping of the logi.CAD εCEDAC Instant reload approach to the five execution sequences of a system evolution step implies that it will not be necessary to consider each execution sequence by itself for evaluation. The download of the ECA and the initialization sequence will be considered together as preparation sequence, and the deinitialization sequence and the deletion of the ECA will be summarized as post-processing sequence (see also Section 5.2).

Preparation sequence: There are several aspects that have been mentioned within the preparation of the ECA. In contrast to the considerations in Section 5.2.2 no free-programmable part exists in the initialization sequence. Nevertheless, it is necessary that the runtime environment provides appropriate means to download the ECA and the new resource without causing disturbances in the currently executed resources. Furthermore the requirements of resources need to be taken into consideration in advance. No actions within the execution of the ECA exist that influence the necessary amount of memory, but the download of the ECA and the new resource has to be considered. The type library for the new resource is an integral part of the source code, therefore this element of requirements of resources will not be violated due to the principles of the logi.CAD εCEDAC Instant reload. The check for dependent operations is also unnecessary, because no actions will be executed that are modeled by the ACS customer.

Reconfiguration sequence: The main evaluation means for the reconfiguration sequence is model checking, because the interaction of the ECA and the control application have to be checked in relation to the plant, possible network interactions, and other disturbances within the control device. For the logi.CAD εCEDAC Instant reload the situation is similar, because the changes within the runtime environment are related to user-defined logic within the εRECONF logic. The necessary formal model for a control device has to include the procedure mentioned in Figure 53b, but in contrast to the models especially for dynamic reconfiguration (see Section 7.3) only a switch between two resource models has to be modeled apart from the READ and WRITE commands. Furthermore it has to be evaluated if the included algorithm for transition management does not cause disturbances in the overall plant. According to the different system integrity characteristics, the following checks have to be fulfilled for a system evolution step:

- ***Global and local consistency:*** The plant, process, and product specifications need to be fulfilled also for the system model including the system evolution step.
- ***Active references***: The IEC 61131-3 standard especially provides active references in terms of global variables. The influence between different algorithms connected by global variables has to be analyzed carefully in order to do not violate specifications for the normal operation. Next to verification by model-checking the interrelation between algorithms may be evaluated also by an appropriate calculation. Whisnant *et al.* (2003) present an approach for dynamically reconfigurable software (in general, not focused on embedded systems) where the existing dataflow dependencies are analyzed based on the current system state and the new configuration. This methodology may be applied for IEC 61131-3 based systems, too.
- ***State management:*** The evaluation of the chosen transition management policy provides information about the possible disturbances caused by the system evolution step.
- ***Real-time constrained operation:*** The execution of the reconfiguration sequence within a certain time limit is of special interest for the logi.CAD εCEDAC Instant reload because the execution of the resource is interrupted. During the execution of the εRECONF logic the operation of the resource is stopped, therefore in any case it is necessary that this part signals its successful execution. The operation of the resource would not be continued otherwise. Further effects of the disruption of operation for any control application within the control device need to be evaluated.

In addition to model checking, also KAPPA-based calculations are mentioned for the reconfiguration sequence. For the logi.CAD εCEDAC Instant reload only dependent operation has to be taken into consideration. It has to be checked if the εRINIT logic does only include access to the old resource and the εRECONF logic to the new resource. No further checks for requirements of resources are necessary.

Post-processing sequence: Within Section 5.2.2 only dependent operation has been identified as a necessary property of the evolution specification. In the logi.CAD εCEDAC Instant reload approach, the εRDINIT logic has to be evaluated if situations with erroneous usage of basic reconfiguration services exist. In general it is necessary that the εRDINIT logic can be executed successfully.

10.4 Summary

The Instant reload mechanism provided by the IEC 61131-3 based engineering tool logi.CAD represents the current state of the art for the dynamic reconfiguration of ACSs. The control logic is changed in between the cyclic execution, whereupon the internal states of unchanged elements are restored. By enhancing this mechanism with the engineering approach developed in the εCEDAC project (the logi.CAD εCEDAC Instant reload mechanism) also DSE is possible.

As main differences to the approach described in this thesis only read and write commands exist within the ECA, because the main principle is based on the exchange of a resource. The new resource is deployed as an entity, in contrast to changing single connections or FBs within the existing resource. But on the other hand this principle is characterized by only little effort for the ACS customer, because he has to model only the reconfiguration sequence as well as some post-processing.

As a consequence the evaluation is focused on the reconfiguration sequence, which is especially critical because no control logic is operated during the execution of εRECONF logic. This part of the ECA is modeled by the ACS customer and has to be checked if there will be no influence to the specifications for normal operation and real-time constrained operation. In addition, active references caused by the use of global variables according to the concepts of IEC 61131-3 have to be taken into consideration.

Chapter 11

Outlook

The use of DSE and its evaluation is an important step towards agile manufacturing. The potential of time reduction for the reconfiguration and change based on the overall time-to-market for process and production plants, as described in Figure 1, provides a high incentive to utilize this new methodology within future ACSs. Next to the dramatically decrease of costs of change the paradigm of agile manufacturing will be put to a new level of quality: the continuous execution of changes during the overall evolution of a production plant according to the principles of AXP (see Section 9.2) without any shutdown has the potential to revolutionize the ACS industry.

This work sets a starting point by formulating a new engineering methodology and a new concept for the evaluation of DSE. But there is still work left, which is related to the practical realization as well as further theoretical investigations. We will discuss different aspects for further enhancements, starting with logical next steps based on the work presented here and ending up in visionary considerations about future application of DSE.

Standards for device description and parameters

The representation of the current system state, the KAPPA vector, is the basis for an improved engineering support in general and the evaluation process especially for DSE. Within this thesis the use of an open standard, FDCML, in conjunction with the models of IEC 61499 has been demonstrated. For the use of such a device description in industrial practice a common sense on the format and the usage of parameters needs to be established. FDCML provides a very general basis and does not define concrete parameters. Therefore it is highly flexible for the description of various situations, even if they are not known up to now. But on the other hand an engineering tool needs clear definitions of parameters and their semantics in order to automatically operate the information of a device description.

Based on the current status in ACSs detailed definitions of enhancements for standards are generated by user organizations such as PLCopen [44], OPC Foundation [41], or CiA [6]. Based on a standardized description format such as FDCML these organizations should provide detailed definitions for device descriptions, which may be used as common basis for engineering tools and the exchange of data between different companies.

Another aspect is related to the definition of parameters for the description of real-time behavior, e.g., the architectural elements of a control device or the communication network. A satisfactory situation within typically heterogeneous ACSs can only be achieved by a common sets of parameters and measurement specifications for their obtainment. This is an important prerequisite for both KAPPA-based calculations as well as formal models, and in addition provides the possibility to compare different platforms by standardized parameters, as proposed in Sünder *et al.* (2007d) for benchmarking IEC 61499 runtime environments.

System model for verification

A critical point for the evaluation of DSE is the provision of appropriate models of the control devices. Within this thesis the principles for modeling the most important architectural elements of a control device have been discussed based on NCES. For a practical application, these models should be generated and assembled to the overall system model with only little (or in the best case no) interaction with the user. First approaches exist already in this direction, as for instance presented in Pang and Vyatkin (2008) with a special focus on the IEC 61499 standard. Based on the exhaustive description of the current system state KAPPA an algorithm should be capable to put together the different fragments from the different architectural elements.

A principal problem in the application of NCES has been detected for the modeling of non-Boolean variables. The proposed solution for the representation of integer variables as a set of places enables simple calculations within the system model. But it should not be necessary to model arithmetic operations by NCES. An extension of the NCES formalism towards the use of data types would enhance the usability of formal methods in general and especially for the evaluation process of DSE.

A special problem for the design of the system model is related to the model of the plant. The ACS customer has to provide these models which are highly related to the concrete production plant. Of course, based on any formalized description, the appropriate models may be generated automatically, as for instance described in Lobov *et al.* (2006a) for UML. But the ACS customers normally use their own description formats, which are usually not related to UML. Therefore, it would highly decrease the effort for designing the plant model, if existing description formats, which are highly specific to a given application domain, would be supported by algorithms for the automatic generation of the plant model.

Enhanced system analysis and behavior

The basis for the evaluation of DSE, an exhaustive description of the KAPPA vector and a system model incorporating functional as well as temporal behavior of the overall configuration of a control device, may be used also for investigations in completely different fields, as for instance the design of control devices in general. The current situation in the design of real-time computer systems, for instance WCET analysis, is unsatisfactory as depicted in Bouyssounouse and Sifakis (2005, Section 7.3). The incorporation of real-time behavior to the formal model of the system provides the necessary information for such analysis by using model checking algorithms. Clarke *et al.* (1999, Chapter 16) describe algorithms for quantitative temporal analysis, which may be enhanced for a detailed consideration of the temporal behavior of real-time computer systems.

This work has presented an approach for DSE of an ACS. The laws of software evolution from Lehmann and Ramil (2000) as well as emerging challenges for software evolution as depicted in Mens *et al.* (2005) discuss this topic in a very general scope. All their statements are related to computer science and component-based software engineering. But the special needs and environmental conditions of ACSs need some more investigations to adapt these laws and challenges correspondingly to the industrial practice in ACSs. A set of laws directly related to DSE of ACSs would be an important input for further developments of engineering paradigms such as AXP.

Intelligent tool support

The acceptance of new technologies especially in the field of ACSs is highly related to an appropriate tool support, as outlined for instance in Hall *et al.* (2007). Of course many possibilities exist for tool support, especially based on the information related to the KAPPA vector. We will focus on possibilities related to DSE and the evaluation by model checking:

- As already described in Section 7.6.1 a simple mapping of elements of the programming language and the model language can be defined. An engineering tool, incorporating both the modeling of the control applications and the verification tool may provide the possibility to use only the elements of the programming language within specifications (which may be established by using patterns or wizards).
- The analysis of results for the verification by model checking may be difficult for ACS customers, because the formal model will not be familiar to them. Therefore, an intelligent engineering tool should be possible to visualize the verification results in terms of the control application. This aims at the representation of different paths within the reachability graph of a system (e.g. for visual verification) or the inspection of counter examples in case of the violation of some specifications. The VEDA tool presented in Vyatkin and Hanisch (2001a) already incorporated such capabilities.
- The continuous evolution of a plant sets high demands to the revision management for the different KAPPA vectors. Next to the pure documentation of the system's changes over time, the engineering tool may provide further possibilities for the ACS customer to browse the various configurations of the plant (the past and the already engineered future) in order to analyze the previous evolutions and to plan the next steps.

Automatic generation of ECAs

The engineering of DSE is of course a time consuming activity, which accompanies additional effort for its evaluation. Accordingly an advantageous enhancement may aim at the reduction of this effort by the automatic generation of at least parts of the ECA. The clear structure of the ECA provides a good basis for such automatisms. Based on the difference between the current system state and the new system state for instance the RINIT and RDINIT sequence may be generated automatically by analyzing the new and the deleted elements. Available templates for predefined evolutions, e.g., the exchange of an FB, may be adapted automatically by the engineering tool. The properties for the evaluation of the ECA may be used by such an algorithm in order to generate correct ECAs a priori.

But there is also a more visionary application for the automatic generation of ECAs, if we think of a production plant which is capable to autonomously react on disturbances and optimize its processes according to the current production order. Such a system may also be in the position to apply changes to the plant according to the current needs. The system would calculate the necessary changes in the control applications, generate the necessary ECAs for these changes, and apply them to the plant at run-time. This would increase the flexibility of agile manufacturing systems to a high extend, since the reconfigurability of the system is increased from the upper coordination level down to the low level control logic. Combined with further possibilities of physical dynamic reconfiguration (see next item) this would be a possible next step to autonomous RMSs.

Physical dynamic reconfiguration

The capabilities of DSE are highly related to the capabilities of the underlying system environment for dynamic reconfiguration. This work is focused on changes to control applications, triggered by basic reconfiguration services. The incorporation of changes to the hardware are related to additional human interaction in Section 4.4. But if also capabilities exist for changing the hardware configuration by using services, the methodology for DSE will become more powerful. There are two kinds of services possible as next steps:

- ***Physical reconfiguration within control devices:*** A control device may be capable to change its hardware configuration triggered from any service within the software architecture. This may be possible if a certain hardware part provides different configurations which may be changed during operation. But there may also be the possibility to freely change hardware functionality for instance based on reconfigurable Field Programmable Gate Arrays.

- ***Physical reconfiguration within a plant:*** The different components of a plant may be arranged automatically and therefore provide services for the reconfiguration of the hardware components of a plant. Already existing examples are CNC machines, which do change the tool according to the current production order. If the concept of automation objects becomes practically relevant, a hardware component will incorporate software components, too. An exchange of a component, which is not only related to the tool of a machine but also a modification of the overall hardware configuration, provides a new kind of flexibility for production plants. The DSE methodology may be enhanced by services which trigger the dynamic reconfiguration of hardware components.

Vision for future practical applications

How to apply the methodology of DSE in a future practical example? We will consider a virtual plant which is inspired by the testbed available in the Odo Struger Laboratory at the ACIN. The main element is a transfer system, which consists of autonomous components such as switches, crossings, index stations, and conveyor belts. These components are provided by a component vendor and include basic software functionalities. The transfer system is manufactured by a machine vendor, who does additionally includes functionality for control and scheduling. Finally, different machines are connected via the transfer system; the coordination of the whole plant is designed by a system integrator. What kind of scenarios for DSE may be possible within this virtual plant?

- ***Firmware update:*** The component vendor improves the basic functionality for instance of the crossings. He models an ECA in order to evolve existing components to the new firmware version and provides it to his customers. The machine vendor will check the correctness of the DSE within the transfer system and evolves the system afterwards. Based on the new functionality also the control functionality for the machine can be improved. The machine vendor models the appropriate ECAs and evolves the operating transfer system.
- ***Transfer system enlargement:*** Due to experiences from the operation of the transfer system new paths should be included. After designing the mechanical enlargements, the machine vendor engineers the evolution of the control functionality of the related components (switches and crossings). Now first tests with the new configuration are possible, which lead to enhancements within the scheduling functionality of the transfer system: the machine vendor provides appropriate ECAs and evolves the transfer system during operation.
- ***Autonomous system control:*** We assume an autonomously acting system control, which is responsible for the coordination within the plant, for this scenario. A failure happens within the system (e.g., a machine breaks down), and the system control calculates a new system configuration for the optimal plant operation. Therefore, changes in some machine functionalities are necessary. The system control calculates the necessary changes within the software functionality of the machines, generates the ECAs and applies the changes. Then the new control strategy is applied within the system control, by the execution of automatically generated ECAs.

Chapter 12

Conclusion

The capability of changing functionalities without interrupting the operation will become one of the main features of next generation automation and control systems. Different studies such as Favre-Bulle (2005) or the European High-Level Group Manufuture drastically point out, that ACSs have to become easily changeable in terms of software and hardware functionality. But the current state of the art is not able to satisfy these needs. The time-to-market for process and producting plants in case of reconfiguration and change (based on an existing plant) increases the time effort especially for the production process (see Figure 1). The paradigm of agile manufacturing, which claims highly flexible production facilities which are capable to adapt themselves to the fast changing markets and product portfolios, does not provide the promised efficiency in real-world applications up to now. Concepts for the dynamic reconfiguration especially focus on the pure capability to apply changes, but without considering of the necessary engineering process in behind.

This thesis focuses on the engineering process and introduces the new methodology of downtimeless system evolution, which uses dynamic reconfiguration capabilities as basis input and sets up new concepts for their application. The term DSE may be explained as follows:

- ***Downtimeless:*** Changes have to be applied to the running system with as little disturbances to the process under control as possible, in the best case without any disturbances.
- ***System:*** Although software is considered to be the central element that is changed, also changes to hardware—or more general changes to the overall system—are taken into consideration.
- ***Evolution:*** Changes of a system become normal operations within the plant, which are applied continuously as soon as any change is necessary. The overall system evolves during its life-cycle according to the changing requirements.

The engineering of DSE does not only describe the transition from the current system state to a new system state. We have to proof also the correctness of each system evolution step, because the main prerequisite for changes within this methodology is the execution during operation of the plant without disturbances. The three main topics, which have been developed in the scope of DSE, can be characterized by the following three questions:

How to model the transition of the system without disturbing its operation?

The starting point for DSE is a new engineering methodology, which puts change into the center of considerations. A plant is in most cases a unique part and the different processes within the plant are highly heterogeneous. We proposed a new kind of application, the evolution control application, which models the transition to the new system state freely programmable for the user. The different possibilities for dynamic reconfiguration, the so-called basic reconfiguration services, represent the basic building blocks for ECAs. The effects of dynamic reconfiguration have been structured in five sequences, which are executed sequentially: download of the ECA, initialization sequence, reconfiguration sequence, deinitialization sequence, and deletion of the ECA. The engineer follows a clear outline for modeling single system evolution steps as well as the interconnection of several steps to enable also the synchronization between different areas of the reconfiguration within the control application. Changes to the hardware configuration are possible by integrating human interaction into the execution of system evolution steps.

How to decide if this transition is free of failures?

The interruption of production processes or even break downs are very expensive and time consuming, therefore DSE promises high financial benefits by keeping the system running also during the execution of changes. But the transition to the new system state has to be correct, too. In order to evaluate the absence of failures within ECAs, we have analyzed the capabilities for dynamic reconfiguration based on the reference model of Walsh *et al.* (2007b), which provides also a list of system integrity characteristics according to the different change types. Due to the structured engineering approach for DSE and the different properties of system integrity, the five sequences for the execution of a system evolution step have been interrelated in order to identify the most appropriate evaluation means. As a result, two kinds of evaluation means were presented:

- ***Verification by model checking:*** This methodology provides due to its automatic character, the result as true/false decision with counter example, and the incorporation of a detailed system description into the formal model the best methodology for the evaluation of the reconfiguration sequence. This sequence includes the active change of the current system state and needs to be taken into consideration very carefully.
- ***KAPPA-based calculations:*** Several properties can be evaluated by calculations based on the current system state. The calculations are based on rules concerning resource properties. As there will be no active adaptations to the functional behavior of the control application within the preparation and post-processing of a system evolution step, these calculations are sufficient for the evaluation of DSE apart from the reconfiguration sequence, which is mainly evaluated by model checking.

How to model the system in order to provide the basis for the evaluation process?

Apart from the means and properties for the evaluation of DSE the representation of the system behavior within appropriate models has to be defined. Herein the system state, the so-called KAPPA vector, was the basis for an exhaustive description of the system and the control devices. We have used the general description format of FDCML and provided enhancements related to the IEC 61499 standard as well as parameters for the evaluation process.

Within the evaluation by KAPPA-based calculations especially the adaptations of the KAPPA vector during the evaluation process has to be pointed out. As DSE aims at changes to the system during operation, the KAPPA vector needs to be changed accordingly during the evaluation process.

For the verification by model checking, the representation of two important topics has to be taken into consideration in detail. On the one hand any ACS is characterized by its functional and temporal behavior. Therefore, especially the consideration of temporal behavior has taken an important role within the system model for model checking. The second topic is again dynamic reconfiguration, which is not part of typical verification means. Based on the restriction of verification by model checking to the reconfiguration sequence and the possible set of basic reconfiguration services in this area the effects of dynamic reconfiguration have been modeled by means of the chosen formal description language NCES.

Reconsideration of requirements

The analysis of requirements for DSE in Chapter 2 resulted in a set of eight claims that need to be fulfilled by the new methodology for the engineering and evaluation of changes to an ACS during its operation. We will reconsider these requirements and describe their fulfillment by the concepts of this thesis roughly:

- ***(1) Temporal behavior:*** A detailed analysis of the temporal behavior of the control device's elements were incorporated in both the KAPPA-based calculations and the formal model of the system.
- ***(2) Execution semantics:*** The execution semantics of the R^3E were analyzed and incorporated to evaluations by KAPPA-based calculations as well as the functional behavior of control logic execution within the system model.
- ***(3) Underlying system configuration:*** A control device was considered with all architectural elements, starting from the interaction with the environment (plant and communication network), the hardware platform, operating system, any kind of applications and programs, up to the ECA.
- ***(4) Modeling dynamic reconfiguration:*** A model for the effects of basic reconfiguration services within the reconfiguration sequence were specified in order to incorporate changes of the system within the verification by model checking. The KAPPA vector is a dynamically changing quantity during the evaluation of DSE.
- ***(5) Free programmable downtimeless system evolution:*** The new engineering methodology for DSE enables free programmable ECAs based on basic reconfiguration services and uses the programming languages common to the ACS customers.
- ***(6) Extensive engineering support:*** Any concept that was presented in this work is applicable by an engineering tool. Furthermore, the introduction of an exhaustive description of the current system state KAPPA represents the basis for this extensive support of the engineer.
- ***(7) Provision of formal models:*** The overall architecture of a control device was exemplarily provided with formal models in the modeling language NCES. Based on the concept of a value-added chain for total evaluation the ACS customer receives the necessary models by the different vendors or they may be generated by the engineering tool according to the presented transformations.
- ***(8) User-friendly definition of specifications:*** The different properties for the evaluation of DSE were simplified by separation to the most appropriate evaluation means. Therefore, many properties can be checked by simple definitions within rules based on the current system state. For the use of temporal logics the property specification patterns system was used, whereas the mapping to the elements of the programming language to the formal model has been explained exemplarily, which can be used to provide specifications by means of the programming language used for the control application.

Appendix A

Field Device Configuration Markup Language

The FDCML has been developed as a markup language for the description of ACS components. Information about the consortium and further information are available at [10]. The following description of the main elements of FDCML is based on the specification FDCML.org (2002), which provides the description of the XML Schema.

The requirements for the establishment of FDCML are defined as given in (FDCML.org, 2002, Chapter 2):

- ***Network independence:*** "FDCML is able to describe network components in a network/bus independent manner without loosing the ability to describe network specific properties."
- ***Multi language support:*** "FDCML is able to support descriptive text elements in multiple languages in one XML file."
- ***Extensibility:*** "FDCML is able to store more information as defined in " its specification "without the need to change the format of the device description."

The most important aspect for this work is extensibility, as it provides the basis to use FDCML for a device description that includes especially parameters necessary for the evaluation of DSE. Multi language support is achieved by the use of appropriate attributes within the XML schema elements that define the used language. Network independence is especially interesting for current ACS applications, since this is the field where device descriptions are used in most cases. Herein a very basic structure is defined that (similar to the elements that provide extensibility) provides a framework for the declaration and description of any parameters.

A.1 Basic elements of the FDCML schema definition

The FDCML schema definition is closely related to ISO 15745-1 (2003), which consists of the four elements device identity, device manager, device function, and application process (see also Section 3.3). The device model provides a modular structure. On the one hand single devices may be described according to these four elements. On the other hand a composition of single elements and their interrelation by connections can be incorporated, too.

Figure 56 depicts the basic structure of the FDCML schema definition and its main elements in a simplified manner[29]. The root element is 'ISO14745Profile', depicting the relation of FDCML to the ISO 15745 standard. 'ProfileHeader' includes information about the device

[29] The figures in this chapter show the structure of the XML schema and also incorporate multiplicities. Each rectangle depicts an element, whereas attributes of elements are neglected. A drawn through line represents mandatory elements, and a shadow depicts multiplicity. A dotted line represents optional elements. A switch represents a choice between elements, whereas also this choice may have multiplicity (represented as double switch in Figure 58b).

description file itself, since FDCML is a markup language and may be used in different kinds. It is possible to describe one single device or composite devices, according to the choice 'ProfileBody' (single device) or 'ProfilesBody' (composite devices).

If we consider a single device description, the four elements already mentioned above are available as schema elements. 'DeviceIdentity' includes a list of fixed elements which provide information about the device vendor or the product name, just to give two examples. 'DeviceFunction' as well as 'ApplicationProcess' are not within the scope of the FDCML definition and refer to any external schema that may be added. Additionally also a non-standardized extension to 'ProfileBody' is included by the element 'nonStandardizedExtension'. The element that is investigated in more detail is 'DeviceManager', that contains all information concerning to network configuration and device structure.

Figure 56 depicts the most important elements within 'DeviceManager', namely 'DeviceStructure', 'communicationEntity', and 'resourceEntity'. Next to these elements additional elements for documentation but also for arbitrary information (e.g., the link to an external XML schema definition) are included.

- ***'DeviceStructure':*** This element describes the physical structure of a device. Next to the elements 'channelList'(including physical and logical channels) and 'MAUList'(a collection of network interfaces) also indicators and slots are mentioned.
- ***'communicationEntity':*** This element describes a network facility within a device. A device may have different communication entities as this is common for ACSs. As one of the requirements for FDCML is network independence the 'communicationEntity' element includes a framework for the declaration of parameters concerning the communication facitiy. This is for instance a list of configuration items or process data descriptions. Further the association to a network interface is included next to arbitrary additional information (see Section A.2 below).
- ***'resourceEntity':*** This element is the counterpart of 'communicationEntity' as it describes facilities within a device that perform functionalities which are not related to network communication. The different elements within 'resourceEntity' include next to configuration items also logical connection points as well as arbitrary additional information.

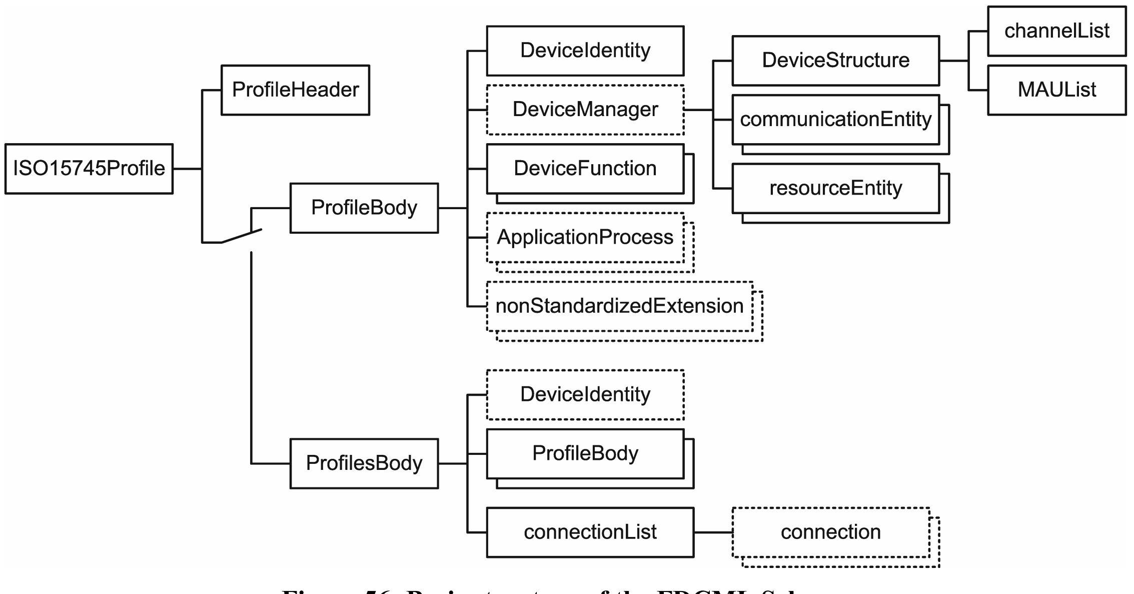


Figure 56: Basic structure of the FDCML Schema

A composite device is defined by the 'ProfilesBody' element, which again may include a 'DeviceIdentity' element in order to provide general information of the overall device. In this case 'ProfileBody' elements are used in order to define the components within the composite device. For the interrelation between these components a list of connections (‘connectionList’) is defined, which consists of ‘connection’ elements that provide information about source and destination of the interrelation as well as specific properties of the connection (see description below).

A.2 Elements that provide extensibility in FDCML

The FDCML schema includes different elements in order to achieve extensibility of the device description. Within the discussion above we have already mentioned the use of external schemas, the element ‘nonStandardizedExtension’ within ‘ProfileBody’, as well as configuration item lists that can be handled in a very free manner. Additionally there are two kinds of elements that further provide open space for arbitrary information within the elements of the FDCML schema: specific properties and additional items.

The ‘specificProperty’ element

A specific property is a pair consisting of a name and a value. Both are depicted in Figure 57 in detail. The name of a ‘specificProperty’ is given as a group (‘g_naming’) that may consist of a label (‘label’) with an reference for this label (‘labelRef’) and appropriate help information (‘help’, ‘helpRef’, and ‘helpFileRef’). The value of a specific property is given by the group ‘g_values’ and provides a set of possibilities in order to define the value of an element. Corresponding to the elements depicted in Figure 57 the following possibilities are provided:

- Constant values (‘const’).
- Editable element values (‘edit’).
- A set of valid element values or names for these values as enumeration (‘enumeration’).
- A set of ranges for valid element values (‘range’).
- Boolean element values (‘yes’ and ‘no’).
- References to other elements within the device description (‘reference’).
- The value of an instance (‘instanceValue’).

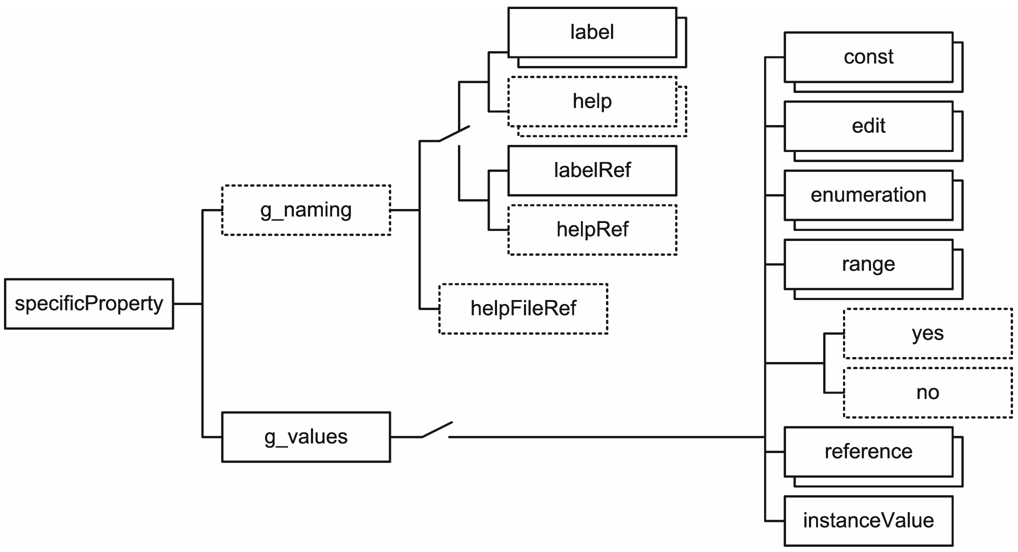


Figure 57: The FDCML element 'specificProperty'

The ‘additionalItem’ element

A more powerful means for describing any arbitrary information within the framework of FDCML schema elements is provided by ‘additionalItem’. The element is depicted schematically in Figure 58a. Next to the elements already mentioned for ‘specificProperty’ above, 'g_naming' and 'g_values', further information can be included as properties ('specificProperty') or further additional items ('additionalItem'). This offers the possibility to describe data in a hierarchical manner. Further elements such as 'picutureList' or 'instances' (information how to instantiate a certain element) are provided, too.

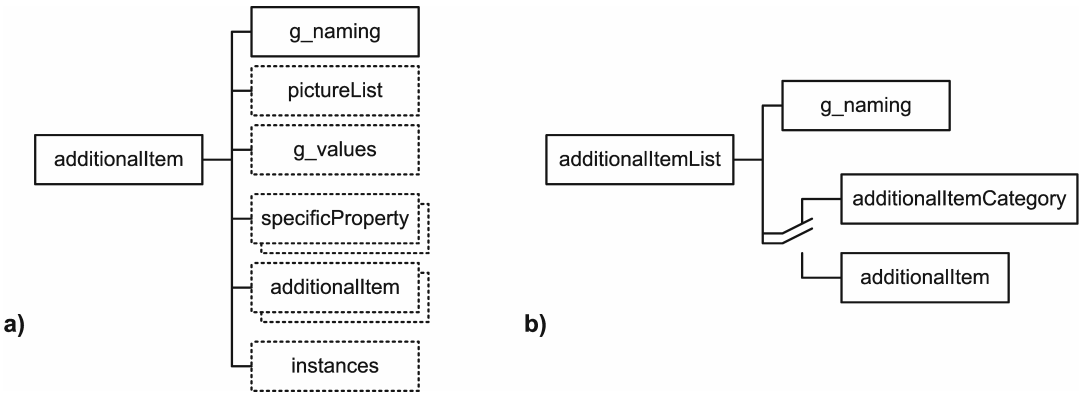


Figure 58: The FDCML elements a) 'additionalItem' and b) 'additionalItemList'

Based on the element 'additionalItem' FDCML provides also a collection of additional items as 'additionalItemList' (see Figure 58b). Next to the group 'g_naming' an arbitrary number of choices between 'additionalItemCategory' and 'additionalItem' are possible. 'additionalItemCategory' provides a means in order to define a vendor specific category of 'additionalItem' elements.

Appendix B

Real-time Reconfiguration Runtime Environment

The runtime environment which is taken as concrete example for the considerations within this thesis is characterized by three main features:

- ***IEC 61499 runtime environment:*** The R^3E is compliant to the definitions of (IEC 61499-1, 2005) as well as the additional definition of the IEC 61499 compliance profile for feasibility demonstration [17].
- ***Real-time execution:*** The R^3E is capable to execute FB networks with regard to real-time constraints. Therefore special SIFB types for the encapsulation of different sources of events are defined, which are the user interface for the runtime capability to separate between different event flows and match them to the scheduling algorithm of the operating system.
- ***Reconfiguration support:*** The R^3E further provides enhanced capabilities for the reconfiguration of control logic during execution. The management commands defined in (IEC 61499-1, 2005) are fully supported as well as additional commands necessary for DSE are part of the runtime environment.

The development of this runtime environment has been pushed by different parties within different research projects, but its main developer is Alois Zoitl from ACIN. The fundataion of the runtime environment has been established during the µCrons research project [36]. Zoitl (2007) gives a very detailed description of the internals of the runtime environment which will be used as main source for the following description. Additional enhancements and specializations have been added during the εCEDAC research project [8], especially within the set of basic reconfiguration services as depicted below. In parallel to this the runtime environment has been made public as an open source project called 4DIAC [12]. According to this history there exist different versions of the runtime environment which provide different sets of features and functionality. We will use the version of the runtime environment which was the result of the adaptions within the εCEDAC project for the considerations and experiments within this thesis. There may be discrepancies with other versions (e.g., the 4DIAC runtime environment). In the following we will describe the two aspects of the R^3E real-time execution and reconfiguration support.

B.1 Real-time execution of IEC 61499 applications

In order to provide real-time execution for FB networks according to IEC 61499 it is necessary to provide a mapping between the elements of IEC 61499 and the elements of a real-time system. The elements of IEC 61499 have been depicted already in Section 3.2.2 which are the models for system, device, resource, FBs and so on. The basic theory for real-time execution will be described roughly based on Douglass (1999, Chapter 2). As already depicted a real-time computer system has to execute programs under certain time constraints. Only if the results can be provided in time, the computation has been successful. The execution of different programs has to be performed concurrently in order to meet the requirements. Herein a lot of work is available that describes scheduling algorithms for concurrent programs. There exist different terms for the context of execution. We have used the word program in the previous description, but also thread or task may be used. For this work we will use task as key word for an execution context. The scheduling of concurrent tasks may be done very simple as for instance cyclic executive (the tasks are statically ordered and executed according to a fixed, cyclic schedule) or time-slicing round robin (tasks are preempted when they exceed a certain time, always the highest priority waiting task is executed, and different waiting tasks on the same priority are executed alternately based on time slices). The different tasks may communicate to each other via special means or share similar resources during their execution. Especially the second case is very challenging since the execution may be locked (deadlock) if a low priority task uses a resource which is necessary for the execution of a high priority task. Herein concepts such as semaphores or mutual exclusion are applied in order to avoid deadlocks.

Zoitl (2007) discusses the mapping of IEC 61499 elements to real-time scheduling theory in detail in Chapter 4. As a result he proposes the so-called event chain concept as appropriate realation ship of IEC 61499 execution and tasks within a RTOS. The following considerations are provided within this theory:

- ***Event sources:*** The most interesting elements within an application are those FBs which are capable to generate events. These are always SIFBs, as for instance E_CYCE which provides a cyclic event based on the timer functionality of the underlying system. Every execution within the FB network starts at this kind of FBs which receive an important rule for the real-time execution concept.
- ***Event sinks:*** The opposite of event sources are event sinks, simply speaking the end of execution within an application. This can be represented by an output event that is not connected or based on the internals of an FB, e.g., based on the current state of the ECC there is no event emitted.
- ***Event chains:*** Based on the sources and sinks of execution Zoitl (2007, Section 4.2.3) defines "an event chain as the chain of FB executions started through an even occurence at one event source FB and ending in an event sink". The event chain serves as the execution context that will be mapped to tasks within the operating system. Accordingly it is possible to add real-time constraints to event chains, in detail single calculations within the overall event driven FB network.

Figure 59 depicts the overall situation for the R^{3}E architecture deviding into two different aspects. Within the application level the FB network is visible and the different event source FBs ('ES 1', ES 2' and 'ES x') are visible as the anchor points for the user for the definition of event chains. Of course the event chains are not statically visible and may change based on the current state of the application (e.g., whether an FB will produce an output event or not) and the FB network may be interrelated between different event sources. But from the execution point of view the correlation of FBs and tasks is clearly defined: each event that is triggered based on a certain event source belongs to the execution context (event chain) of this event source. On the execution level there exist two different elements. On the one hand the underlying services for the event source have to be handled appropriately (one external event source may be related to different event source FBs). As the event sources are a critical part for the execution and especially an overload situation within the runtime environment, Zoitl (2007) proposes the use of a guarding of external events (suppress external events if they do not occure according to their specification). On the other hand the execution of tasks within the operating system has to be handled by appropriate scheduling algorithm. The execution of FBs within such a task is proposed according to the event dispatcher concept: Each FB that needs to be executed is inserted in a list by the input event which receives the event. The FBs are executed in a sequential manner according to this list, which is handled as first-in-first-out buffer.

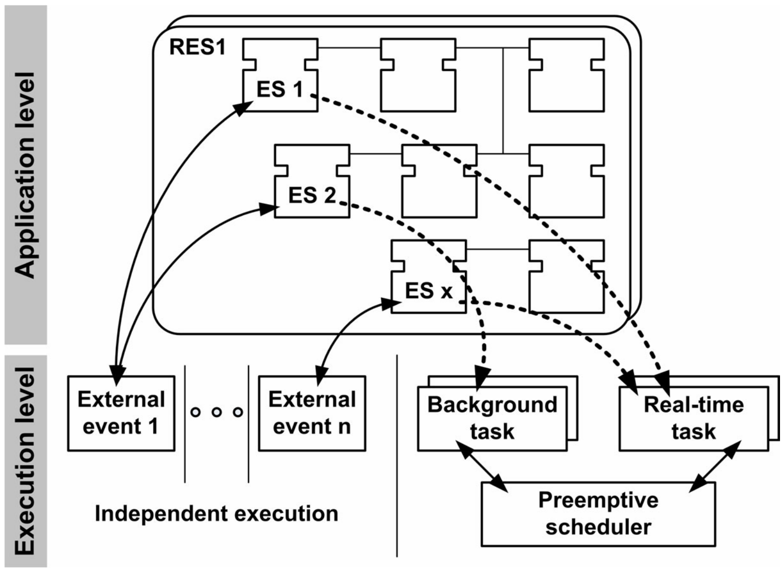


Figure 59: Interrelation of external events, event chains and tasks within the operating system (Zoitl, 2007, Figure 4.3)

The critical point for the real-time execution of FB networks is to prove whether an application will be schedulable (it is possible to fulfill all real-time constraints) or not. Zoitl (2007) provides a detailed analysis of schedulablity based on results from real-time scheduling theory based on the occurrence specification of the external event sources, a limited execution time of event chains, and the structure of the event chain itself. To give an example, there exist dependencies between different event chains based on a special class of event sources, the so called event chain couplers. Typical IEC 61499 applications will be interrelated by event and data connections. In terms of event chains this situation is not very satisfying as huge parts of the application will be executed within the same execution context. But in many cases the real-time constraints can be limited to rather small portions of the application. An event chain coupler can be used in order to change the execution context within a chain of executing FBs. Figure 60 depicst this situation schematically. Within the FB network there exists only one event source triggered by some external event ('ES'). In order to separate the overall execution into different portions with different real-time constraints ('deadline1', deadline2' and unconstrained execution) event chain coupler FBs ('EC coupler') are used.

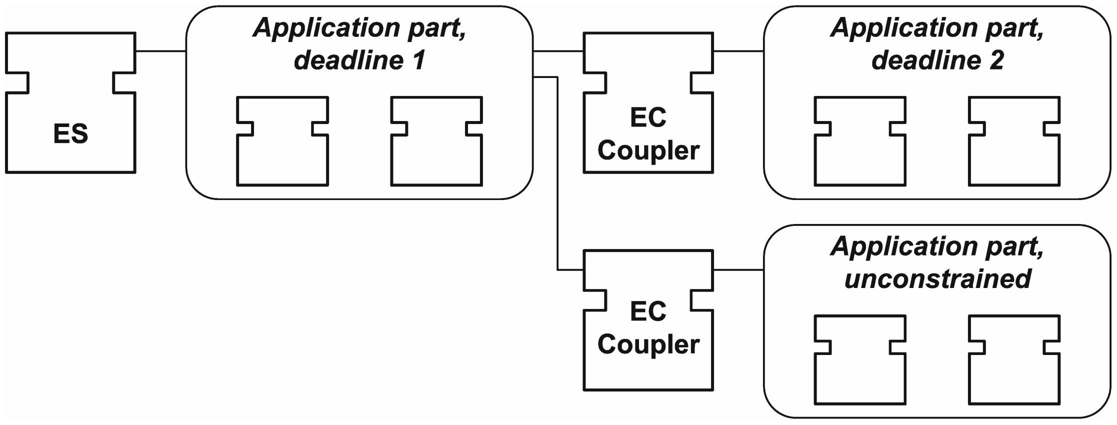

ES
Application part, deadline 1
EC Coupler
Application part, deadline 2
EC Coupler
Application part, unconstrained

Figure 60: Separation of execution contexts within a chain of executing FBs (Zoitl, 2007, Figure 4.9)

You can imagine that there exists a high variety of different FBs that serve as event sources or as coupling element in order to separate execution contexts within an application. Zoitl (2007, Appendix C) provides a list based on the event function blocks that are defined in (IEC 61499-1, 2005, Annex A). Among these FBs there are for instance the cyclic execution under real-time constraints as depicted in Figure 61a (‘RT_E_CYCLE’). An interesting FB emerges from the enhancement of an ‘E_SWITCH’ FB with real-time constraints (‘RT_E_SWITCH’ in Figure 61b). Based on the Boolean value ‘G’ different execution paths with different execution contexts will be triggered. The already describe coupler for the separation of one execution path into two execution contexts is presented in Figure 61c (‘RT_E_EC_COUPLER’).

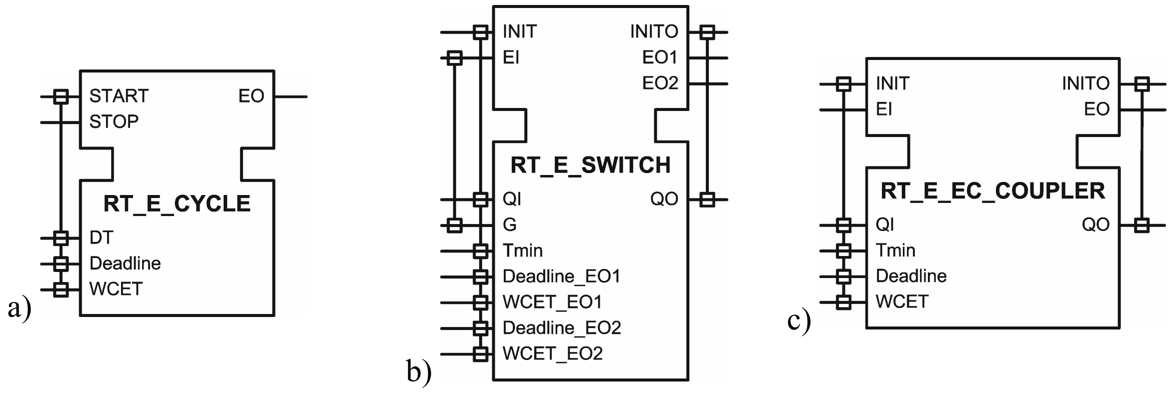


Figure 61: Different event source FBs a) real-time constrained cyclic execution b) data dependend splitting of an execution chain c) coupler FB for changing the execution context within an application (Zoitl, 2007, Appendix C)

B.2 Basic reconfiguration services

The reconfiguration approach described in Zoitl (2007) is based on the most important work on dynamic reconfiguration from Kramer and Magee (1985). But in contrast to the original definition of a configuration manager, which is responsible for the application of the configuration changes to the current system state in order to change from one configuration specificatio not another, Zoitl declares that the reconfiguration should be applied by a special application, the so called reconfiguration application. Herein the functionality for changing the current system state is incorporated as special FBs and the way how to change the system state can be used to model application specific (this concept has been discussed also in Section 4.2.1 for DSE).

The main elements are the basic reconfiguration services, which provide the necessary functionality for changing the current system state and are encapsulated as SIFBs. Zoitl (2007, Section 3.2.2) defines five categories of basic reconfiguration servces which are necessary for dynamic reconfiguration:

Structural services: The structural reconfiguration services provide mechanisms for changes to the structure of the control application. The device cannot be created by an appropriate service as it is the element which provides the basis for the application of these services (in detail the management application in order to access a device). But all elements within a device are affected by structural services:

- “CREATE resources within devices, FBs within resources, and connections (event and data) between them.”
- “DELETE resources, FBs, and connections.”
- “WRITE parameter values to device data inputs, resource data inputs, and FB data inputs.”

Zoitl (2007) defines a generic interface for the different FBs incorporating basic reconfiguration services in (Zoitl, 2007, Annex A). Figure 62a depicts the interface of the FB that is capable to create a (data or event) connection. The FB interface provides as input values the necessary parameters for the management command CREATE. These are the destination ‘DST’ (which resource is concerned) as well as the source ‘SOURCE’ and destination ‘DESTINATION’ of the connection. These parameters are provided as dot-seperated list as defined in (IEC 61499-1, 2005). In order to minimize the execution time of FB during the execution of an evolution step, this FB has been adapted as depicted in Figure 62b (similar adaptions have been applied also for other basic reconfiguration services). On the one hand an initialization of the FB has been added in order to move decoding of strings to a not time critical phase. Further the parameters have been split up, as for instance the instance name and the input/output name for the source and the destination of the connection.

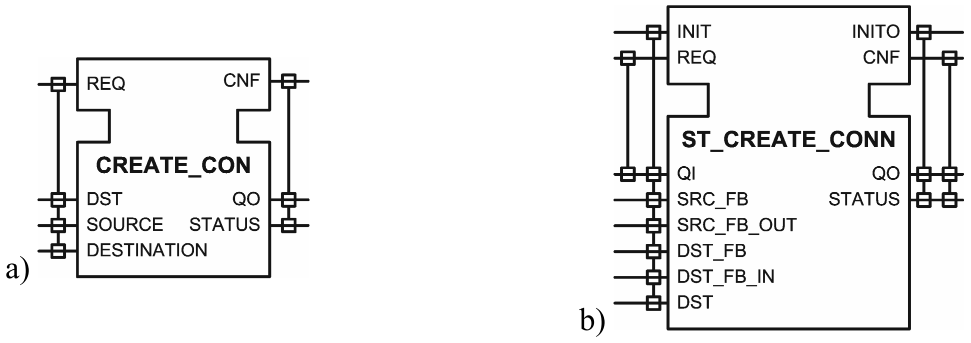


Figure 62: Interface of basic reconfiguration services for the creation of a connection a) as defined in Zoitl (2007, Appendix A) and b) as available in R^{3}E

Library services: The library reconfiguration services influence the library available within a device. Zoitl (2007) does not provide FBs in order to incorporate these basic reconfiguration services since library services are tightly coupled with the engineering tool and can be applied via the management application. The establishment of a library which is dynamically adaptable during runtime is a challenge especially for resource-limited devices. Zoitl (2007) proposed the use a virtual machine approach. The engineering tool transforms the type definition of a BFB into the machine code of this virtual machine, and the runtime environment interprets the machine code and simulates a situation as the virtual machine would be physically present.

Execution control services: The execution control reconfiguration services set the state of a managed FB or resource. These services are based on the state machine for managed FBs, which is defined in (IEC 61499-1, 2005, Section 3.3). There may exist also FBs that cannot be managed as they are fundamental part of the runtime environment. For instance the resource that includes the management application within a device is of such a type. But in common the FB networks and also resource within a device are established by management commands and therefore their behavior belongs to this state machine (see Figure 63). A transition within the state machine corresponds to the execution of the mentioned management command. Therefore, also structural services for the creation and the deletion of an FB instance are part of the state machine. The execution control services influence the FB instance during its execution:

- “START puts the FB in the running state. Input events are processed.”
- “STOP” stops the processing of input events. No further input events are processed.” If the management command occurs during the execution of an algorithm of the affected FB, then this algorithm will be completed.
- “KILL aborts the processing of input events. No further input events are processed.” In this case an algorithm that is just executed will not be completed and the FB may be in an inconsistent state.
- “RESET puts the FB back into the initial state.”

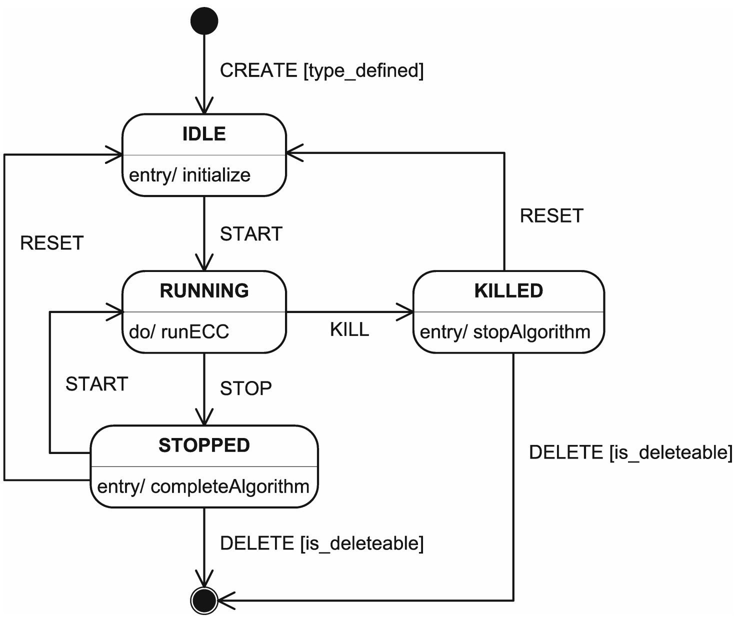


Figure 63: Operational state machine of a managed function block (IEC 61499-1, 2005, Figure 24)

State interaction services: The state interaction reconfiguration services provide access to the internals of an FB by the use of the management command READ and WRITE. This is an enhancement to the definitions of the IEC 61499 standard which claims this functionality only for input and output data. Zoitl (2007) defines a very simple FB interface for this basic reconfiguration service capable to handle input, output, and internal variables (see Figure 64a). Due to similar reasons as discussed already for above the R^3E uses a more specialized FB interface (and therefore also a higher number of different FB instances in the basic reconfiguration services library). Figure 64b depicts an FB for reading an internal variable of an FB instance. Again the parameters of the management command are already analyized during initialization of the FB. A further possibility to enhance execution speed of this kind of FB is the output variable 'VALUE'. Zoitl (2007) uses a string in order to provide any kind variable type. But encoding and (in case of transition management) calculation and again decoding of a certain data type from and to string data type can be omitted by provided specialized FB types based on the type of the internal variable.

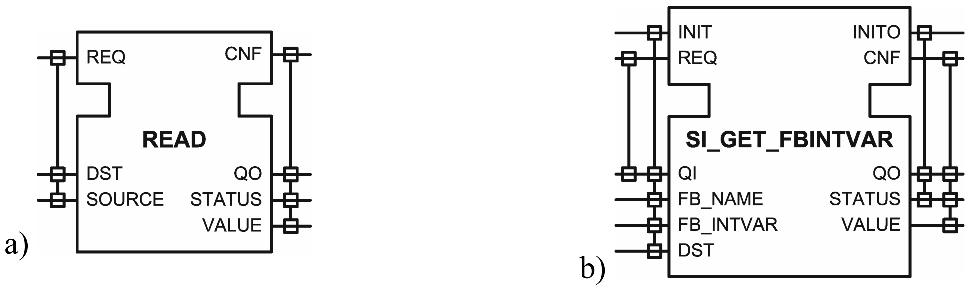


Figure 64: Interface of basic reconfiguration services for reading of values a) as defined in Zoitl (2007, Appendix A) and b) as available for internal variables in R^3E

Query services: The query reconfiguration services can be used to establish the current system state by interaction with the control devices. The following commands are mentioned in (Zoitl, 2007):

- "QUERY resources returns a list of the instantied resources within a device or for a resource's instance name the resource type name can be retrieved."
- "QUERY FBs returns a list of the instanced FBs within a resource or for an FB's instance name the FB type name can be retrieved."

- "QUERY FB state returns the current execution state of the FB." As depicted in the state machine in Figure 63 the possible states are 'IDLE', 'RUNNING', 'STOPPED', and 'KILLED'.
- "QUERY connections returns a list of all connections within a device or resource or retrieve the corresponding end point of a connection for a given connection's source or destination specification."
- "QUERY type returns the type definition of the queried type (resource, FB, data type)."

These basic reconfiguration services are mainly useful for an engineering tool, as the requested data will become very large in some cases. For instance, these query services provide a big amount of data necessary to acquire the current system state. This is the first step within the engineering cycle for DSE (see Section 4.1).

But query service may be useful also for the modeling of the ECA itself. For instance it provides the possibility to check in some aspects whether the different sequences of a system evolution step have been applied correctly. Another opportunity is modeling of failure handling procedures.

B.3 Measurement results

Zoitl (2007, Chapter 5) provides a detailed experimental analysis of the behavior of the implemented runtime environment. As there have been no changes to these parts of the runtime environment within R^3E, his results are applicable also for this thesis. The concepts have been applied on three different embedded hardware platforms in order to quantify also dependencies between the underlying system configurations.

The results for the pure real-time execution model of IEC 61499 applications can be summarized as follows:

- The concept of real-time execution holds for the different runtime platforms. If the schedulability bounds given in Equations 5 and 6 are fulfilled, the real-time constraints of the control application will be met.
- Because the test platforms used are very limited in memory usage and processing power it was possible to apply situations where the schedulability bounds are violated. "Even in these overload situations the runtime environment provided a deterministic execution of control applications. Furthermore the most important event chains (i.e. the event chains with the shortest deadlines) met their real-time constraints in this critical situation" (Zoitl, 2007, Section 5.2.3).
- A special target of the runtime design mentioned in Zoitl (2007) is independent execution behavior from the underlying systm confinguration, especially the RTOS. The experiments with three different test platforms showed that the runtime environment is not able to abstract all RTOS characteristics. But it was possible to observe general similarities in the execution behavior of the different test platforms.

Further Zoitl (2007) provides experiments which aim at the consideration of the execution of basic reconfiguration services during operation of real-time constrained event chains.

- The download of an application via the management application during execution of real-time constrained event chains does not influence the real-time constraints of the control application.
- A separated consideration has been applied for QUERY commands, because they have to be replied big amount of data as answer (e.g., a list of FB types or connections). Also for QUERY commands the operation of real-time constrained event chains has not been influenced.

- A simple ECA without real-time constraints, which acts on a real-time constrained event chain, does influence the execution behavior of the control application to an extent of one to ten percent of the deadline (based on a rather small application and deadline). It is important that there exists some spare execution time in order to provide the capability of execution of basic reconfiguration services within the preparation and the post-processing of a system evolution step.
- The disturbance of an ECA which is executed with real-time constraints has not been considered in detail as Zoitl (2007) states that therefore a careful planning and adjustment of the control application and the ECA is necessary. The content of this thesis provides exactly the missing context.

Appendix C

Net Condition/Event Systems

The formalism of Net Condition/Event Systems (NCESs) has been introduces in Rausch and Hanisch (1995) as a formal, modular modeling approach that is based on two concepts already known in literature:

- Sreenivas and Krogh (1991) introduced "condition/event systems as a class of continuous-time discrete event dynamic systems with two types of discrete-valued input and output signals: condition signals and event signals".
- Petri (1961) introduced his modeling approach for asychronous communication in computer systems based on places, transitions, their interconnection, and the token flow within the graph (see also Section 3.6.3). There exist many different dialects of so-called Petri nets.

The main idea of NCES is to combine these concepts in the following way. The dynamic behavior of modules is described as Petri net, which are extended with condition and event signals. Based on these modules composition can be modeled as condition/event systems by interconnecting the event and condition interfaces of the different modules. In this way a hierarchical model architecture can be established. In order to provide model checking for this kind of modeling approach, the overall system can be reduced into one single system without modules (flattening of the hierarchy). According to Starke and Roch (2002) such a system without inputs and outputs (so-called autonomous systems) are called *Signal-Net Systems* (SNSs), and an appropriate model checking algorithm is described. We will focus our considerations on a certain type of NCES, which provide the usage of timed arcs as well as multiple tokens within arcs. Further enhancements such as distinguishable tokens (coloured tokens) as described in Starke and Roch (2002, Section 3) or the combination of NCES with extended timestamp nets (a high-level Petri net class with tokens that may carry timestamps and additionally an arbitrary number of other elements—which may be especially continuous state variables for modeling of dynamic behavior) as depicted for instance in Hanisch *et al.* (2000) will not be taken into consideration.

C.1 Timed net condition/event systems

The above given extension of NCES with timed arcs is called *Timed Net Condition/Event Systems* (TNCES) and has been introduces in Hanisch *et al.* (1997). TNCES provide the basis for different applications of the modeling approach of NCES as for instance depicted in Hanisch *et al.* (2006) or Lobov *et al.* (2006a). Within this thesis we will always use timed NCES models. A graphical representation of a timed NCES model is depicted in Figure 65. The main elements are places (which may contain a certain number of indistinguishable tokens), transitions, and arcs between places and transitions. These arcs are called flow arcs as they enable the flow of tokens from places to transistions and transitions to places. The interface of a module is given by input and output events as well as input and output conditions. An event represents the firing of a transition (dynamic property); a condition represents the incorporation of tokens within a place (static property). Accordingly there exist two further kinds of arcs within a NCES module, the event arcs and the condition arcs. The first one is able to transmit events between conditions (or the interface of the NCES module), the second one provide the static property of tokens incorporated within a state from places to transitions. A condition represents a Boolean value, whether there exists a token in the related state (true) or not (false).

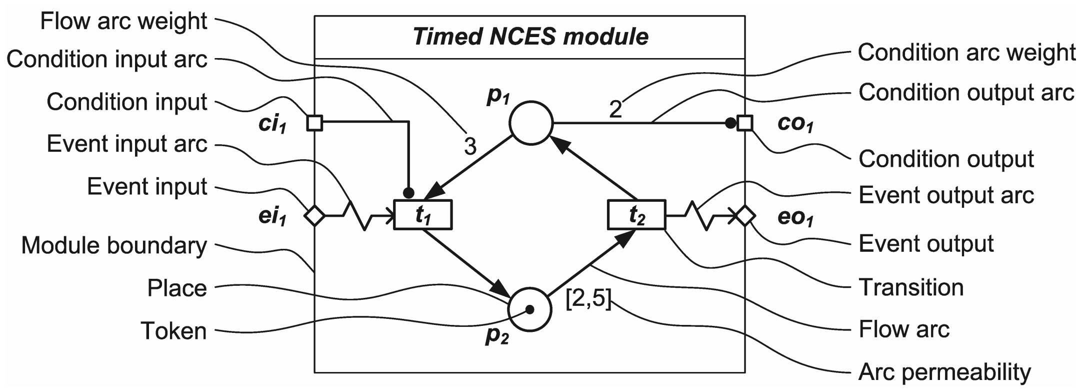


Figure 65: A timed Net Condition/Event Systems module

The behavior of a model represented as NCES is depicted by firing rules which describe when a transition is evaluated true. There exist two different types of transitions:

- ***Spontaneous transitions:*** A spontaneous transition does not have any incoming event signal. The transition clears if all incoming transition arcs are true and the incoming flow arcs do have a marking within the related place. Only if all incoming condition and flow arcs are evaluated true (logical AND relation), the transition may fire. The token flow via the transition is given by the flow arcs. If the transition fires an event is issued to every outgoing event arc.
- ***Forced transitions:*** A forced transition has at least one incoming event arc. Additionally there may be incoming condition and flow arcs as depicted for spontaneous transitions. The transition will be evaluated only if an event is issued to the transition via the incoming event arc. There may be several incoming event arcs, too, whereas a logical OR or AND relation may be used within the firing rule.

In addition to the firing rules described above two further situations have to be distinguished:

- ***Number of tokens:*** As there may be several tokens within a place (denoted by dots or a number within the place) also the condition represented via a condition arc can be separated according to the number of tokens. A condition arc may have a weight, which means an integer number which describes the threshold when the condition arc will issue a true condition to the related transition. Only if there are equal or more tokens within a place than given by the condition are weight, the condition arc will issue a true value. The situation is similar for flow arcs. Herein also a flow arc weight may be defined, which has two effects to the behavior of the NCES model. On the one hand a transition may fire only if there are equal or more tokens within a place as denoted in the flow are weight. But on the other hand the flow arc weight also gives the number of tokens that are removed from the place when the transition fires. A flow arc weight may be used also for arcs that interconnect a transition and a place. In this case the flow arc weight defines the number of tokes that are added to the place when the transition fires.
- ***Timed arcs:*** Time is represented as an integer value related to clocks within places. As soon as a place includes a token the clock is enabled. The execution of a NCES model is stepwise, which means based on the current situation (tokens within places) the

transitions are evaluated. Spontaneous transitions fire if the conditions given by incoming condition and flow arcs are fulfilled. A forced transition is evaluated as soon as it is enabled. Therefore a stepwise execution emerges, because as soon as all transitions have been evaluated once no more action is possible within the NCES model. As soon as this situation has been reached the enabled internal clocks within the places are increased by one and the next step is executed. The timed behavior within NCES modules is incorporated by the use of permeability intervals to flow arcs that connect places with transitions. The permeability interval provides a lower and upper bound for the value of the clock within the place. Only if the clock value is within this range, the corresponding flow arc will be evaluated true.

According to Hanisch *et al.* (1997) "a Timed Net Condition/Event System (TNCES) is a tuple

$$TNCES = \{\ P, T, F^{+}, F^{-}, M_0, \psi, CN, EN, DC\ \} \tag{27}$$

with:	P	finite, ordered set of n places p
	T	finite, ordered set of m transitions t
	F^{+}	$n \times m$ - matrix of input arcs
	F^{-}	$n \times m$ - matrix of output arcs
	M_0	initial marking, vector of dimension
	ψ	input/output structure
	CN	Condition signal matrix of dimension $n \times m$
	EN	Event signal matrix of dimension $m \times m$.

The input/output structure ψ is defined as follows:

$$\psi = \{\ C^{in}, E^{in}, C^{out}, E^{out}, Bc, Be, Cs, Dt\ \} \tag{28}$$

with:	C^{in}	finite, ordered set of r Condition input signals
	E^{in}	finite, ordered set of s Event input signals
	C^{out}	finite, ordered set of p Condition output signals
	E^{out}	finite, ordered set of q Event output signals
	Bc	Condition input matrix of dimension $r \times m$
	Be	Event input matrix of dimension $s \times m$
	Cs	Condition output matrix of dimension $n \times p$
	Dt	Event output matrix of dimension $m \times q$.

The time extension of a TNCES (…) is defined as follows:

$$DC = \{\ DR, DL, D_0\ \} \tag{29}$$

with:	DR	$n \times m$ – matrix of delay time
	DL	$n \times m$ – matrix of limitation time
	D_0	initial state of local clocks associated to the places.

The elements $DR(\ i, j\ )$ and $DL(\ i, j\ )$ of matrices DR and DL denote the associated delay and limitation times of the permeability of the arc from place i to transition j."

Next to this definition of basic NCES modules there exist also so-called composite NCES modules which are characterized by the component NCES modules interrelated via event and condition arcs (similar to condition/event systems). Vyatkin *et al.* (2003a) provides a description of problems and their solution by enhancements to the classical NCES formalism. Such a restriction has occurred for instance for interconnections of modules with arc weights of input and output condition arcs or the behavior of open (unconnected) event and condition inputs.

The solution may differ depending on the used model checking tool and algorithm. For this thesis we utilize the visual framework for verification of function blocks [61] with the following solution provided:

- The chain of condition arcs with different arc weight will be defined by the smallest arc weight within this chain.
- Every open input is considered as inactive. An open event input never receives an event and an open condition input is always false.

The use of a hierarchical engineering by the use of NCES is depicted for instance in Vyatkin and Hanisch (2003b). Figure 66a shows a simple situation of two different NCES modules interconnected as composite NCES module.

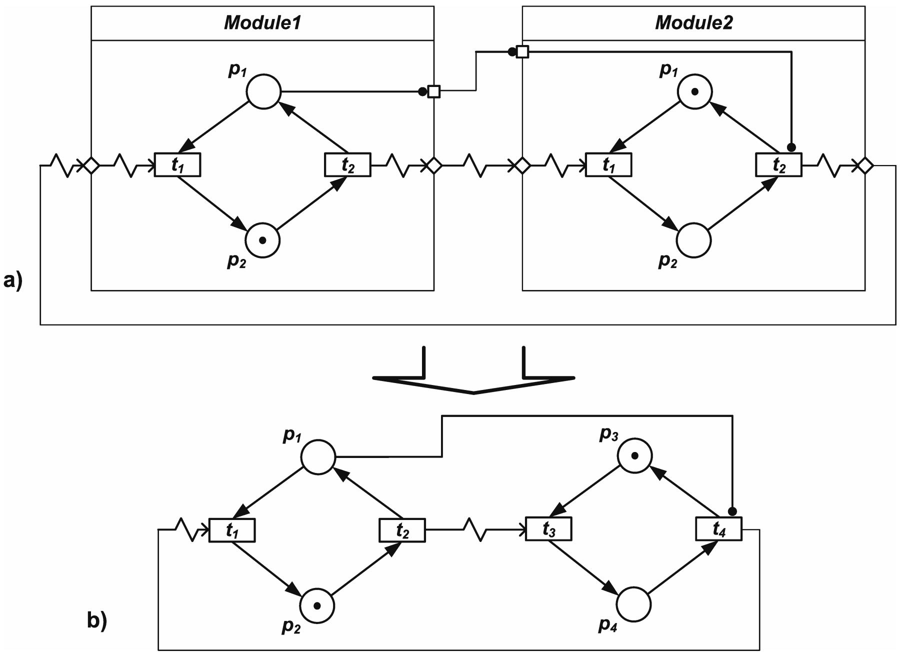


Figure 66: (a) Composition of NCES modules and (b) the corresponding SNS model

C.2 Signal net systems

The analysis of NCES is based on a flat model of the overall model without external inputs, the so-called Signal Net Systems. SNSs are described in detail in Starke and Roch (2002), which take into consideration different variants of SNS. For this thesis we use SNS under time constraints, which utilize a similar model of time as depicted above for TNCES. A SNS is generated based on a NCES model as depicted in Figure 66. The composite NCES models in the upper part of the figure are transformed into a SNS model by resolving the borders of the NCES modules.

According to Starke and Roch (2002, Section 1.1) a SNS (without timing constraints) can be described as

$$N = \{ P, T, F, V, B, W, S, M, m_0 \} \tag{30}$$

where 1. P is a non-empty finite set (of places),

2. T is a non-empty finite set (of transitions), disjoint with P,

3. F is a subset of $(P \times T) \cup (T \times P)$ (the flow relation, the set of flow arcs),

4. V is a mapping which atteches a positive integer to every arc (the arc weight, $V : F \rightarrow \mathbb{N}$),

5. B is a subset of $P \times T$ (the set of condition arcs)

6. W is a mapping which attaches a positive integer to every condition arc (the condition arc weight, $W : F \rightarrow \mathbb{N}$),
7. S is a subset of ($T \times T$) \ id_T, the irreflexive signal (flow) relation,
8. M is a mapping which attaches a (signal-processing) mode to every transition ($M : T \rightarrow \{ \wedge, \vee \}$, and, finally,
9. *m0* is a marking of *P* called the initial marking or the initial state of *N*.

In constrast to the definition of NCES herein the arc weights of flow arcs and condition arcs are especially mentioned within *V* and *W*, whereas in Equation 27 the matrices F^+, F^- (represented as *F* in Equation 30), and *CN* include this information as intereger values directly. In addition the parameter for the processing of events in forced arcs is also represented explicitely in *M*. The representation of time is added in a similar manner to SNSs as described above for NCES in Equation 29. A formal definition for timed SNSs is given in Starke and Roch (2002, Section 2, Definition 2.1).

The analysis of any modeling language is based on the possibilities incorporated within the model checking algorithm provided by an appropriate tool. In case of SNSs the model checker SESA, which has been developed at Humbold-Universität zu Berlin, is available. Starke and Roch (2002) provide a detailed description of the theory that provides the basis for SESA as well as a short tool description. SESA is part of the visual framework for verification of function blocks [61]. According to Starke and Roch (2002, Section 5) the following methods are incorporated within SESA: "Once we know that an SNS is bounded, we can (at least in principle) decide all further questions by construction of a reachability graph[30]. But the state explosion problem urges us to look for methods which, depending on the question at hand, avoid unnecessary computations, i.e. which compute only a subgraph of the reachability graph:

- restrict the depth of the computed graph (applicable in the unbounded case too),
- use a 'bad' predicate; only states (markings) not satisfying the predicate will be developed further while state satisfying the predicate will be included as dead states into the computed graph,
- use a CTL-formula: compute only that part of the reachability graph needed to check the formula,
- reduce the number of arcs by avoiding simultaneous firing of steps,
- use the stubborn set method to compute a reduced reachability graph,
- use symmetries of the net."

The SESA model checker provides different variants of CTL in order to define specifications.

- Pure CTL as described in Section 3.6.2.
- Extend CTL which utilizes so-called transition formulae that are able to contain certain state transitions between states (which is complicated in CTL). The transition information is based on the edges of the reachabilty graph. A detailed description of extend CTL is given in Starke and Roch (2002, Section 12), but as we will not use extend CTL within this thesis we lack for a more detailed description.
- Timed CTL is introduced in a similar manner as RTCTL described in Section 3.6.2, based on the structure of the reachability graph enhanced by a state delay. This delay

[30] The reachability graph is the state graph of a signal net system. It incorporates all possible states a system may have as well as the possible transitions between these states. In contrast to the unwinded Kripke structure mentioned in Section 3.6.1 each state of the system is only mentioned once.

describes the number of time units that have to elaps before a step can be executed. The satisfaction relation ⊨ is defined in Starke and Roch (2002, Section 13, Definition 13.3) for timed CTL in SESA by the use of intervals that refer to the state delay. An interval [*l*, *h*] with $0 \leq l \leq h$ may be used as time constraint for the operators **X**, **F**, and **U**.

C.3 Tool framework

As already mentioned above we will use the tool framework "visual framework for verification of function blocks" [61] provided by Valeriy Vyaktin within this thesis. The framework incorporates a model editor as well as a verification tool, the so-called *Visual Verifier* (ViVe). The model checking tool used in ViVe is on the one hand SESA and on the other hand a simple built-in model checker. Another framework for the use of TNCES is the MOVIDA tools framework [33] provided by Tampere Univeristy of Technology, which also uses SESA as model checking tool. For pure modeling of TNCES, also the TNCES editor developed from Martin-Luther-Universty Halle-Wittenberg [32] is available and has been used also for this work partially.

A detailed description of the functionality of the tool framework is available via [61]. We will provide a short description of the different options for model checking, as the creation of the reachability graph and therefore the system's behavior are influence by these options to a high extent.

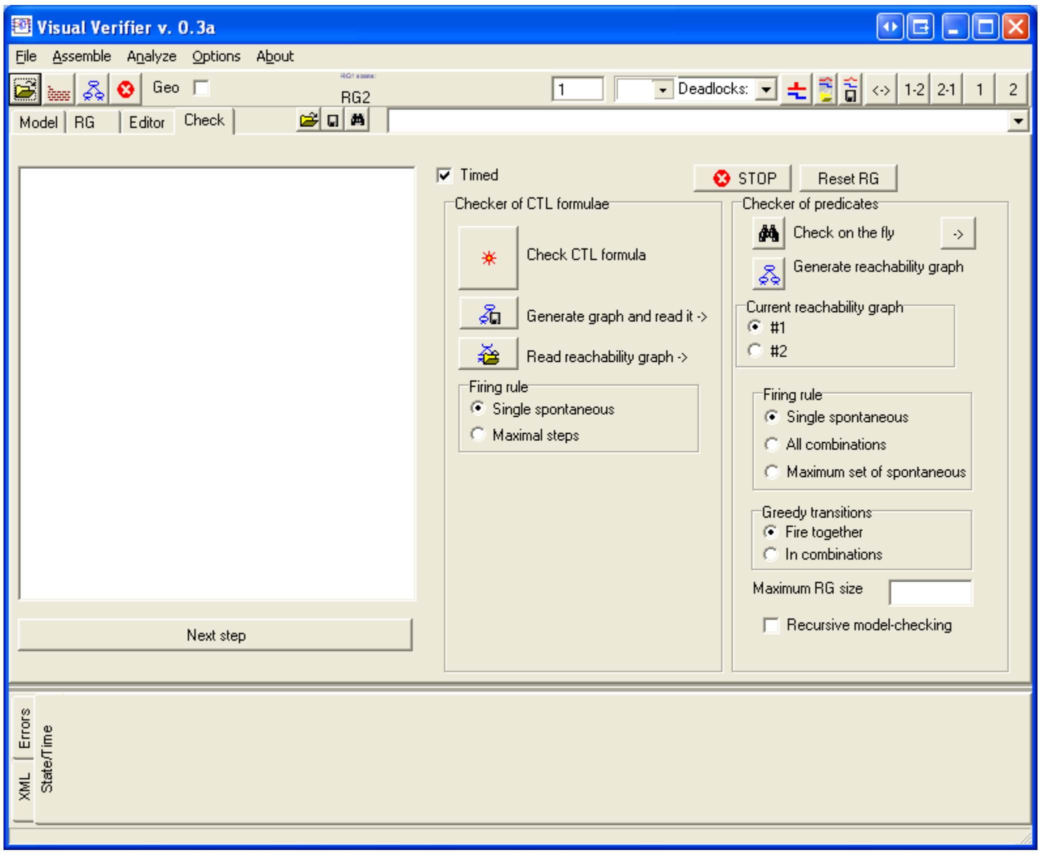
Figure 67 depicts a screeshot of the current version of ViVe with the appropriate pane for the use of the model checking tools. The elements summarized as 'Checker of CTL formulae' provide the interface for the SESA model checker, whereas the elements summarized as 'Checker of predicates' are related to the built-in model checker.

SESA provides many different options for the analysis of SNS as described in Starke and Roch (2002, Appendix). For the generation of the reachability graph especially the firing rule has to be taken into consideration as described in Starke and Roch (2002, Section 1.3). The following two settings can be used within ViVe:

- ***Single spontaneous:*** SESA computes a list of all executable steps at a given state within the reachability graph, including all combinations of enabled spontaneous transitions as well as all possible forced transitions. If 'Single spontaneous' is active this list will be reduced to only those steps that contain only one spontaneous transitions.
- ***Maximal steps:*** If 'Maximal steps' is active all elements within the list of executable steps at a given state will be included for the calculation of the reachability graph.

The options of the built-in model checker of ViVe are described in Vyatkin (2007b), which is part of the documentation of ViVe. The firing rules supported by this model checker are slightly different to SESA:

- ***Single spontaneous:*** The reachability graph is created taking into account only one single spontaneous transition (and as much as possible forced transitions) between two states of the system. This option is similar to the 'Single spontaneous' option mentioned above for SESA.
- ***All combinations:*** Herein any kind of combination of enabled spontaneous transitions (again with all possible forced transitions) is included in the reachabilty graph calculation. This option is similar to the 'Maximal steps' option mentioned above for SESA.
- ***Maximum set of spontaneous:*** This option provides a reachability graph where only the maximal set of spontaneous transitions enabled in a given state is incorporated.



[✓ Timed]

Checker of CTL formulae
Firing rule: [✓ Single spontaneous] Maximal steps

Checker of predicates
Current reachability graph: [✓ #1] #2
Firing rule: [✓ Single spontaneous] All combinations Maximum set of spontaneous
Greedy transitions: [✓ Fire together] In combinations
Maximum RG size
Recursive model-checking

Figure 67: Screenshot of the ViVe tool with the check pane

Appendix D

Embedded Configurable Operating System (eCos)

There are many different RTOSs available within the field of embedded system. Within this thesis we will utilize eCos, which is an open source and royalty-free RTOS available since 1998. Detailed information about the current status of eCos is available in [9]. Further the overall architecture of eCos has been described in Massa (2003), which will provide the basis for the following descriptions.

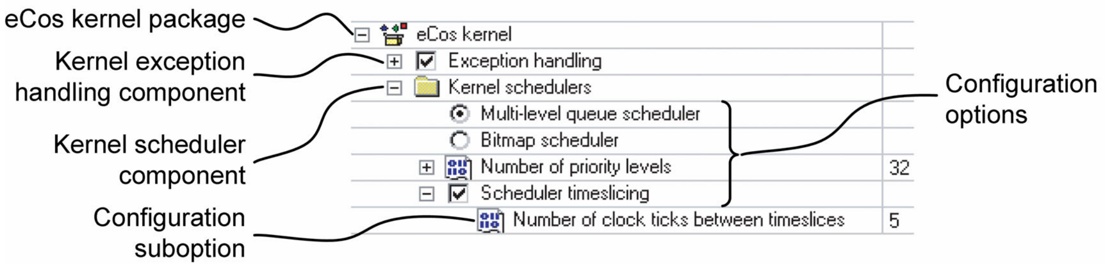
The main characteristics of eCos are:

- Low requirements for necessary memory and processing power
- Availability for various embedded platforms based on a hardware abstraction layer
- Configurability of the overall system (see Section D.1)
- Deterministic behavior for task scheduling and handling of interrups (see Section D.2)

D.1 Configurability

"In order to get an understanding of the eCos architecture, it is important to appreciate the component framework that makes up the eCos system. This component framework is specifically targeted at embedded systems and meeting the requirements associated in embedded design. Using this framework, an enormous amount of functionality for an application can be built from reusable software components or software building blocks. (…) Most embedded software today provides more functionality than what might actually be needed for a particular application. Often, extra code is included in a software system that gives generic support for functionality that embedded developers are not concerned with and is not needed. (…) eCos gives the developer ultimate control over run-time components where functionality that is not needed can easily be removed. eCos can be scaled from a few hundred bytes up to hundreds of kilobytes when features such as networking stacks are included and third-party contributions such as Web servers are used. Developers are able to select components that satisfy basic application needs, and configure that particular component for the specific implementation requirements for the application. This could mean enabling or disabling a particular feature within a component, or selecting a particular implementation for the component. An example of this is in the kernel scheduler configuration. eCos offers the developer options such as the ability to select the number of priority levels and whether time slicing is used. Any code unnecessary to meeting the developer's requirements is eliminated in the final image of the application." (Massa, 2003, Section 1.2.1)

"Figure 68 shows a portion of the eCos Kernel package from the Configuration Tool. The figure shows how the building blocks are encapsulated within each other to create a complete and independent package. We can see the hierarchy of the configuration from packages to components to configuration options to suboptions. Building blocks are grouped together in a package based on the functionality they include. In Figure 68, we see the 'eCos kernel' package, which contains the 'Exception handling' component and the 'Kernel Schedulers' component; the other eCos Kernel components are not shown in this figure. We can see in Figure 68 the nesting of configuration options, such as 'Scheduler timeslicing', and suboptions that compose the components." (Massa, 2003, Section 1.3.1.2)



eCos kernel
Exception handling ✓
Kernel schedulers
[✓ Multi-level queue scheduler]
Bitmap scheduler
Number of priority levels 32
Scheduler timeslicing ✓
Number of clock ticks between timeslices 5

Figure 68: Configuration tool options, (Massa, 2003, Figure 1.2)

D.2 The kernel component

The core of the eCos architecture is the kernel component, which includes different scheduling policies for tasks[31], mechanisms for synchronization of tasks, and the effect of interrupts on task execution. We will focus our consideration on the elements that have been mentioned for the formal model of a control device (see Section 7.4.1), which are the scheduling policies and the synchronization mechanisms.

Multilevel queue scheduler (MLQ)

eCos supports two different scheduling policies for tasks. "The multilevel queue scheduler allows the execution of multiple tasks at each of its priority levels. The number of priority levels is a configuration option from 1 to 32, corresponding to priority numbers 0 (highest priority) to 31 (lowest priority). The scheduler allows preemption between the different priority levels. (…) Preemption is a context switch halting execution of a lower priority task, thereby allowing a higher priority task to execute. The multilevel queue scheduler also allows timeslicing within a priority level. Timeslicing allows each thread at a given priority to execute for a specified amount of time, which is controlled by a configuration option. The queue implementation for the multilevel scheduler uses double linked circular lists to chain together threads within a priority level and threads at different priority levels.

In Figure 69, we see the multilevel scheduling queue representation along with an example of task execution using this scheduler. In the scenario shown in Figure 69, three tasks—'Task A', 'Task B', and 'Task C'—are configured during creation of the tasks at priority levels 0, 0, and 30, respectively. The state of the scheduler queue after thread creation is shown in Figure 69. For this scenario, timeslicing is enabled. The timeline is a snapshot that starts with 'Task C' executing. Next, 'Task A' becomes able to run, causing 'Task C' to be preempted and a context switch occurs. During the execution of 'Task A', 'Task B' also becomes able to run. 'Task A' continues until its timeslice period expires. Then, another context switch occurs allowing 'Task B' to run. 'Task B' completes within its given timeslice period. The descheduling of a thread can happen for various reasons; for example, by waiting on a mutex that is not free or delaying for a specified amount of time. Since 'Task A' has the highest priority of tasks waiting to execute, a context switch occurs and it runs next. After 'Task A' has completed, a context switch takes place allowing 'Task C' to execute." (Massa, 2003. Section 5.1.3.1)

[31] Massa (2003) uses the term thread in order to describe execution contexts, which is defined as „a single flow of execution through a program" which contains „its own context or workspace to perform its operations" (Massa, 2003, Section 6.1). This definition is similar to the term task which is used for independent exeuction contexts within this thesis. In order to provide a common terminology also citations from Massa (2003) will be changed to use the term task instead of thread.

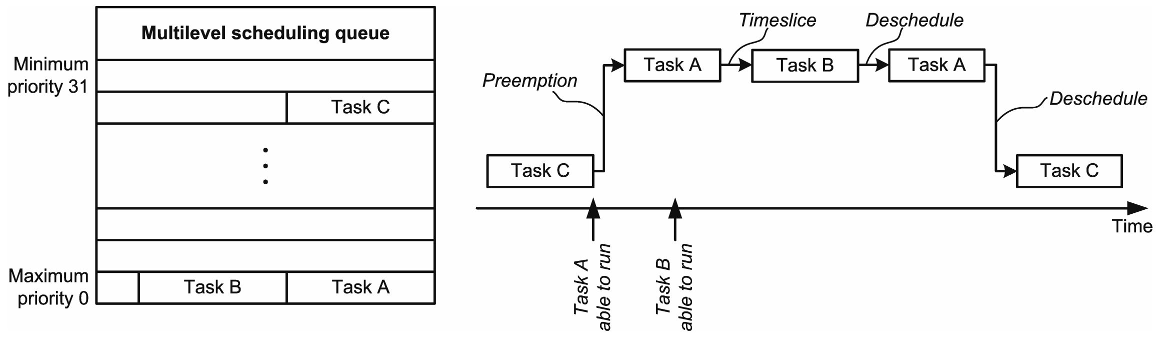


Figure 69: Execution example for MLQ scheduler, (Massa, 2003, Figure 5.3)

Bitmap scheduler

"The bitmap scheduler allows the execution of tasks at multiple priority levels; however, only a single task can exist at each priority level. This simplifies the scheduling algorithm and makes the bitmap scheduler very efficient. The number of priority levels is a configuration option from 1 to 32, corresponding to priority numbers 0 (highest priority) to 31 (lowest priority).

Figure 70 illustrates an example of task execution using the bitmap scheduler. In Figure 70, there are three tasks created at different priority levels: 'Task A'—priority 0 (highest), 'Task B' —priority 1, and 'Task C'—priority 30 (lowest). The state of the bitmap scheduler queue after the tasks are created is shown left to the task execution timeline. The timeline is a snapshot of task execution starting with 'Task C' running. Next, 'Task A' and 'Task B' are able to run, causing a context switch and 'Task C' is preempted. 'Task A' executes next because it has the highest priority of the waiting tasks. When 'Task A' completes, a context switch takes place, enabling 'Task BA' to execute. After 'Task B' completes, 'Task C' can finish its processing." (Massa, 2003, Section 5.1.3.2)

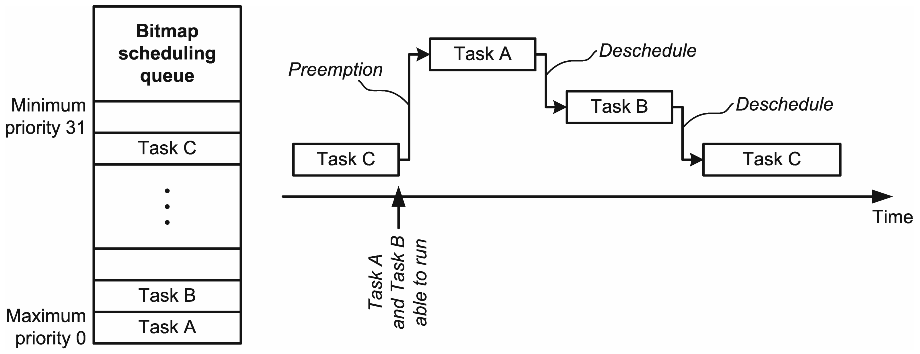


Figure 70: Execution example for bitmap scheduler, (Massa, 2003, Figure 5.4)

Task synchronization mechanisms

eCos supports different mechanisms for tasks in order to communicate with each other and synchronize access to common resources (as for instance the event dispatcher mentioned in Section 7.4.2). We will roughly mentioned the most important ones for the establishment of the formal model.

Mutexes: "A mutex (mutual exclusion object) allows multiple tasks to share a resource serially. The resource can be an area of memory or a piece of hardware, such as a direct memory access controller. A mutex is similar to a binary semaphore in that it only has two states—locked and unlocked. However, there are a couple of differences between a binary semaphore and a mutex. A mutex provides protection against priority inheritance, whereas a binary semaphore does not. (…) A mutex also has the concept of an owner, and only the owner can unlock the mutex. (…) A thread that attempts to lock a mutex that is currently owned by another thread will block until the owner unlocks the mutex.

One issue that arises in real-time systems when using mutexes is priority inversion. Priority inversion occurs when a high priority task is incorrectly prevented from executing by a low priority task. (…) eCos provides two solutions to the priority inversion problem that are selectable as configuration options. The first solution is a priority ceiling protocol. In the priority ceiling protocol, all tasks that acquire the mutex have their priority level raised to a preconfigured value. (…) A more elegant solution eCos provides is a priority inheritance protocol. The priority inheritance protocol allows a task that owns the mutex to be raised to the priority level equal to the highest level of all tasks waiting for the mutex. The priority inheritance protocol is only used when a higher priority task is waiting for the mutex." (Massa, 2003, Section 6.2.1)

Semaphores: "A semaphore is a synchronization mechanism that contains a count indicating whether a resource is locked or available. There are two types of semaphores, counting and binary. Binary semaphores are similar to counting semaphores; however, their count is never incremented past a value of one. Binary semaphores are in either a locked or unlocked state.

Counting semaphores can be in multiple states depending on their count value. Counting semaphore objects contain a value that is incremented when a task posts to a semaphore, and the value is decremented when a tasks completes a wait for the semaphore. Only the highest priority waiting task is executed when the semaphore count is above zero. Counting semaphores are often used when a higher priority task (…), which received data, needs to signal another thread to continue processing the data at a lower priority." (Massa, 2003, Section 6.2.2)

Condition variables: "Another available synchronization mechanism is the condition variable. Condition variables are used with mutexes that allow multiple tasks access to shared data. Typically, there is a single task producing the data, and one or more tasks waiting for the data to be available. The task producing the data can either signal a single task to wake up or all tasks to wake up, with a broadcast signal, when the data is available. The waiting tasks can then process the data as needed." (Massa, 2003, Section 6.2.3)

In addition to the above mentioned task synchronization mechanisms eCos provides the mechanisms flags, message boxes, and spinlocks.

Appendix E

Property Specification Patterns

Dwyer *et al.* (1998) introduce the idea of patterns for specifications in order to simplify the usage of temporal logic. "A property specification pattern is a generalized description of a commonly occurring requirement on the permissible state/event sequences in a finite-state model of a system. A property specification pattern describes the essential structure of some aspects of a system's behavior and provides expressions of this behavior in a range of common formalisms." Dwyer *et al.* (1998, Section 3)

A refined version of the property specification pattern system has been presented in Dwyer *et al.* (1999) based on a survey of specifications that have been published in literature. As a result of the survey on 555 example specifications collected, 511 specifications (92%) matched with one of the patterns mentioned by the authors. The pattern system has been made public available in [53] and has been adapted continuously. In order to provide a snapshot of the current state of the pattern system, this appendix reproduces the content of [53] with special focus on CTL as temporal logic for the formulation of patterns.

Pattern hierarchy

There exist different possibilities for a categorization of patterns. As already presented in Section 7.6.1 a classification in terms of the kinds of system behavior the pattern describe can be represented as pattern hierarchy (see Figure 39). We will use this kind of categorization in the following paragraphs. An alternative organization may be based on the formalisms used for the description in temporal logic. [53] provides a formulation of the patterns in LTL, CTL, Graphical Interval Logic, Quantified Regular Expressions, INCA Queries, Action Computation Tree Logic, and Regular Alternation-Free-Mu-Calculus. We will only provide the formulation in CTL in this appendix, because this is the kind of temporal logic that can be used with the model checking tool SESA (see Appendix C).

Property specifications scope

"Each pattern has a scope, which is the extent of the program execution over which the pattern must hold. There are five basic kinds of scopes: global (the entire program execution), before (the execution up to a given state/event), after (the execution after a given state/event), between (any part of the execution from one given state/event to another given state/event) and after-until (like between but the designated part of the execution continues even if the second state/event does not occur). The scope is determined by specifying a starting and an ending state/event for the pattern: the scope consists of all states/events beginning with the starting state/event and up to but not including the ending state/event.

Figure 71 illustrates the portions of an execution that are designated by the different kinds of scopes. We note that a scope itself should be interpreted as optional; if the scope delimiters are not present in an execution then the specification will be true.

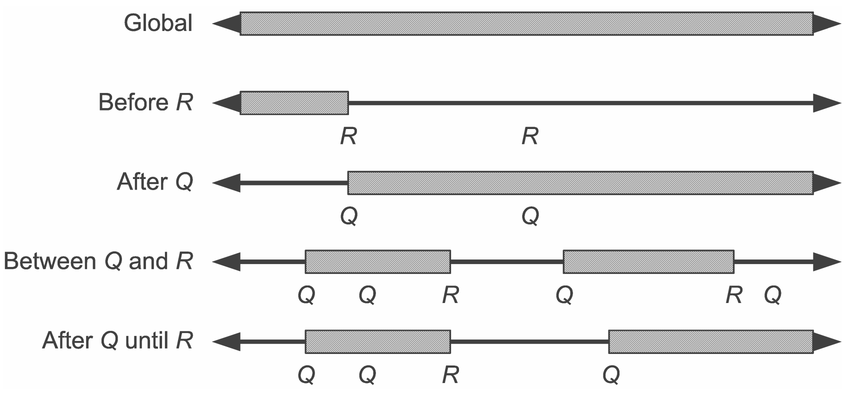


Figure 71: Pattern Scopes, [53]

Before and after scopes for our patterns are interpreted relative to the first occurrence of the designated state/event. We have done this because it matches our experience with real specifications. Note, however, that we could just as easily interpret these scopes relative to the last occurrence of the designated state/event (the mappings given in the patterns are easily transformed to match this interpretation). At present we do not see the need for supporting both first and last occurrence scopes, but as we gain experience applying the patterns we may wish to extend scopes in this way."

Weak until operator (W)

Within the formulation of the patterns in temporal logic the weak until operation **W** is used instead of the until operator **U**. The difference will become visible based on the definition of the two operators:

- p **U** q means that p is true until q is true, with q is true somewhere.
- p **W** q means that at all states p is true until q is true.

The same formula using **W** instead of **U** will be evaluated true even if q is never true. This can be simple expressed as $p \mathbf{W} q = p \mathbf{U} q \vee \mathbf{G} p$. As equivalence [53] provides the following formulae:

$$\begin{aligned} \mathbf{A}\,[\,x\,\mathbf{W}\,y\,] &= \neg\mathbf{E}\,[\,\neg y\,\mathbf{U}\,(\,\neg x \wedge \neg y\,)\,] \\ \mathbf{E}\,[\,x\,\mathbf{U}\,y\,] &= \neg\mathbf{A}\,[\,\neg y\,\mathbf{W}\,(\,\neg x \wedge \neg y\,)\,] \end{aligned} \tag{31}$$

E.1 Occurrence specification patterns

"Occurrence patterns are used to express requirements related to the existence or lack of existence of certain states/events during well-defined regions of system execution. As with our other patterns, the regions are defined using scopes.

There are four occurrence patterns:

- Absence, aka never
- Universality, aka globally, henceforth
- Existence, aka eventually, future
- Bounded existence"

E.1.1 Absence property pattern

"***Intent:*** To describe a portion of a system's execution that is free of certain events or states. Also known as never.

Mappings for CTL:

p is false:

Globally	$\mathbf{AG}\,(\, \neg p\,)$	(32)
Before R	$\mathbf{A}\,[\,(\,\neg p \vee \mathbf{AG}\,(\,\neg R\,))\;\mathbf{W}\;R\,]$	(33)
After Q	$\mathbf{AG}\,(\,Q \rightarrow \mathbf{AG}\,(\,\neg p\,))$	(34)
Between Q and R	$\mathbf{AG}\,(\,Q \wedge \neg R \rightarrow \mathbf{A}\,[\,(\,\neg p \vee \mathbf{AG}\,(\,\neg R\,))\;\mathbf{W}\;R\,]\,)$	(35)
After Q until R	$\mathbf{AG}\,(\,Q \wedge \neg R \rightarrow \mathbf{A}\,[\,\neg p\;\mathbf{W}\;R\,]$	(36)

Examples and known uses: The most common example is mutual exclusion. In a state-based model, the scope would be global and *p* would be a state formula that is true if more than one process is in its critical section. For an event-based model, the scope would be a segment of the execution in which some process is in its critical section (i.e., between an enter section event and a leave section event), and *p* would be the event that some other process enters its critical section.

Relationships: This pattern is the dual of the existence pattern. In fact, in many specification formalisms negation and explicit queries for existence will be used to formulate an instance of the absence pattern, as seen in the examples above.

Note that between scopes in this pattern (with a false proposition or empty event symbol) appear to be able to specify the same thing as a response pattern with global scope. This is not the case, however, since the cause-effect relationship is required for the response whereas the scope for this pattern is optional.

If one wishes to exclude states characterized by multiple propositions or multiple events one can do this by defining *p* appropriately. One common use is to fill the role of *p* in the above mappings with disjunctions of propositions or event symbols. For other parameterizations of patterns consult the pattern notes" (see Section E.3).

E.1.2 Universality property pattern

"***Intent:*** To describe a portion of a system's execution which contains only states that have a desired property. Also known as henceforth and always.

Mappings for CTL:

p is true:

Globally	$\mathbf{AG}\,(\,p\,)$	(37)
Before R	$\mathbf{A}\,[\,(\,p \vee \mathbf{AG}\,(\,\neg R\,))\;\mathbf{W}\;R\,]$	(38)
After Q	$\mathbf{AG}\,(\,Q \rightarrow \mathbf{AG}\,(\,p\,))$	(39)
Between Q and R	$\mathbf{AG}\,(\,Q \wedge \neg R \rightarrow \mathbf{A}\,[\,(\,p \vee \mathbf{AG}\,(\,\neg R\,))\;\mathbf{W}\;R\,]\,)$	(40)
After Q until R	$\mathbf{AG}\,(\,Q \wedge \neg R \rightarrow \mathbf{A}\,[\,p\;\mathbf{W}\;R\,]$	(41)

Examples and known uses: This pattern can be applied in most situations where the absence pattern can be applied. This is especially true for state-based formalisms, e.g., where mutual exclusion could be formulated as absence or universality with a between scope.

Relationships: This pattern is closely related to the absence and existence patterns. Universality of a state can be viewed as absence of its negation. For event-based formalisms, we look for the existence of the positive event and absence of the negative event."

E.1.3 Existence property pattern

"***Intent:*** To describe a portion of a system's execution that contains an instance of certain events or states. Also known as eventually.

Mappings for CTL:

p becomes true:

Globally
$$\mathbf{AF}\,(p) \tag{42}$$

Before R
$$\mathbf{A}\,[\,\neg R\;\mathbf{W}\,(p \wedge \neg R)\,] \tag{43}$$

After Q
$$\mathbf{A}\,[\neg Q\;\mathbf{W}\,(Q \wedge \mathbf{AF}\,(p))\,] \tag{44}$$

Between Q and R
$$\mathbf{AG}\,(Q \wedge \neg R \rightarrow \mathbf{A}\,[\,\neg R\;\mathbf{W}\,(p \wedge \neg R)\,]\,) \tag{45}$$

After Q until R
$$\mathbf{AG}\,(Q \wedge \neg R \rightarrow \mathbf{A}\,[\,\neg R\;\mathbf{U}\,(p \wedge \neg R)\,]\,) \tag{46}$$

Examples and known uses: The classic example of existence is specifying termination, e.g., on all executions do we eventually reach a terminal state.

Relationships: This pattern is the dual of the absence pattern. In fact, in many specification formalisms negation and explicit queries for existence will be used to formulate an instance of the absence pattern.

We may wish to specify that a state/event occur at most some bounded number of times. The bounded existence pattern handles that case.

If one wishes to require the existence of a state characterized by multiple propositions or multiple events one can do this by defining p appropriately. One common use is to fill the role of p in the above mappings with disjunctions of propositions or event symbols. For other parameterization of patterns consult the pattern notes" (see Section E.3).

E.1.4 Bounded existence property pattern

"***Intent:*** To describe a portion of a system's execution that contains at most a specified number of instances of a designated state transition or event.

Mappings for CTL: In these mappings we illustrate one instance of the bounded existence pattern, where the bound is at most 2 designated states. Other bounds can be specified by variations on this mapping.

Transitions to p-states occur at most 2 times:

Globally
$$\neg\mathbf{EF}\,(\neg p \wedge \mathbf{EX}\,(p \wedge \mathbf{EF}\,(\neg p \wedge \mathbf{EX}\,(p \wedge \mathbf{EF}\,(\neg p \wedge \mathbf{EX}\,(p)))))) \tag{47}$$

Before R
$$\begin{aligned}&\neg\mathbf{E}\,[\,(\,\neg R\;\mathbf{U}\,(\,\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\quad \mathbf{E}\,[\,(\,\neg R\;\mathbf{U}\,(\,\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\qquad \mathbf{E}\,[\,(\,\neg R\;\mathbf{U}\,(\,\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \neg R))\,]\,))\,]\,))\,]\end{aligned} \tag{48}$$

After Q
$$\begin{aligned}&\neg\mathbf{E}\,[\,\neg Q\;\mathbf{U}\,(\,Q \wedge \mathbf{EF}\,(\,\neg p \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\quad \mathbf{EF}\,(\,\neg p \wedge \mathbf{EX}\,(p \wedge \mathbf{EF}\,(\neg p \wedge \mathbf{EX}\,(p)))))))\,]\end{aligned} \tag{49}$$

Between Q and R
$$\begin{aligned}&\mathbf{AG}\,(\,Q \rightarrow \neg\mathbf{E}\,[\neg R\;\mathbf{U}\,(\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\quad \mathbf{E}\,[\neg R\;\mathbf{U}\,(\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\qquad \mathbf{E}\,[\neg R\;\mathbf{U}\,(\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \neg R \wedge \mathbf{EF}\,(R)))\,]\,))\,]\,))\,]\,)\end{aligned} \tag{50}$$

After Q until R
$$\begin{aligned}&\mathbf{AG}\,(\,Q \rightarrow \neg\mathbf{E}\,[\neg R\;\mathbf{U}\,(\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\quad \mathbf{E}\,[\neg R\;\mathbf{U}\,(\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \ldots\\ &\qquad \mathbf{E}\,[\neg R\;\mathbf{U}\,(\neg p \wedge \neg R \wedge \mathbf{EX}\,(p \wedge \neg R))\,]\,))\,]\,))\,]\,)\end{aligned} \tag{51}$$

Examples and known uses: Bounded overtaking properties can naturally be expressed using instances of this pattern. For example, if we wish to say that process 1 can enter its critical region at most twice while process 2 is waiting to enter its region we would use a between scope (delimited by process 2 entering and exiting its waiting region) with 2-bounded existence for process 1 entering its critical region.

Mappings of bounds other than two can be constructed relatively simple from the given mappings. (…) For LTL and CTL we simply add additional copies of the nested until structures.

Relationships: This pattern is related to existence and chains. Note that this pattern does not require the occurrence of any number of instances of the given states/events (rather it bounds the number of instances). Single instances can be required with existence patterns. Multiple instances can be required with a slight variant to the above mappings.

Note that response chain patterns are different than bounded existence in two ways: response chains require the responding sequence to be of the designated length (whereas here we only bound a sequence length), and the notion of an instance of a state/event differs between the two. In particular, a stuttered instance (i.e., in consecutive states on a path) counts as multiple instances with the chain whereas it is a single instance with bounded existence."

E.2 Order specification patterns

"Order patterns are used to express requirements related to pairs of states/events during well-defined regions of system execution. As with our other patterns, the regions are defined using scopes.

There are two basic order-related patterns:

- Precedence
- Response, aka follows, leads-to

Chain patterns are used to express requirements related to complex combinations of individual state/event relationships. These include precedence/response relationships consisting of sequences of individual states/events. We call these chain patterns.

There are two variations of chain patterns:

- Response chains
- Precedence chains

A variation of the chain patterns is to constrain the regions between the state/events that constitute the chain sequence." The constrained chain property pattern will not be mentioned in the following discussion (see [53] for more details).

E.2.1 Precedence property pattern

"***Intent:*** To describe relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for an occurrence of the second. We say that an occurrence of the second is enabled by an occurrence of the first.

Mappings for CTL:

In these mappings p is the consequent and s is the enabling state/event.

Globally $$\mathbf{A}\,[\neg p\ \mathbf{W}\ s\,] \tag{52}$$

Before R $$\mathbf{A}\,[\,(\neg p \vee \mathbf{AG}\,(\,\neg R\,))\ \mathbf{W}\,(\,s \vee R\,)\,] \tag{53}$$

After Q $$\mathbf{A}\,[\,\neg Q\ \mathbf{W}\,(\,Q \wedge \mathbf{A}\,[\,\neg p\ \mathbf{W}\ s\,]\,)\,] \tag{54}$$

Between Q and R $$\mathbf{AG}\,(\,Q \wedge \neg R \rightarrow \mathbf{A}\,[\,(\,\neg p \vee \mathbf{AG}\,(\,\neg R\,))\ \mathbf{W}\,(\,s \vee R\,)\,]\,) \tag{55}$$

After Q until R $$\mathbf{AG}\,(\,Q \wedge \neg R \rightarrow \mathbf{A}\,[\,\neg p\ \mathbf{W}\,(\,s \vee R\,)\,]\,) \tag{56}$$

Examples and known uses: Precedence properties occur quite commonly in specifications of concurrent systems. One common example is in describing a requirement that a resource is only granted in response to a request.

Relationships: Note that a precedence property is like a converse of a response property. Precedence says that some cause precedes each effect, and response says that some effect follows each cause. They are not equivalent, because a response allows effects to occur without causes (precedence similarly allows causes to occur without subsequent effects).

Note that this pattern does not require that each occurrence of a consequent will have its own occurrence of an enabling condition."

E.2.2 Response property pattern

"***Intent:*** To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect. Also known as follows and leads-to.

Mappings for CTL:

In these mappings p is the cause and s is the effect.

Globally $$\mathbf{AG}\,(p \rightarrow \mathbf{AF}\,(s)) \tag{57}$$

Before R $$\mathbf{A}\,[\,((p \rightarrow \mathbf{A}\,[\,\neg R\;\mathbf{U}\,(s \wedge \neg R)\,]\,) \vee \mathbf{AG}\,(\neg R))\;\mathbf{W}\;R\,] \tag{58}$$

After Q $$\mathbf{A}\,[\,\neg Q\;\mathbf{W}\,(Q \wedge \mathbf{AG}\,(p \rightarrow \mathbf{AF}\,(s)))\,] \tag{59}$$

Between Q and R $$\mathbf{AG}\,(Q \wedge \neg R \rightarrow \mathbf{A}\,[\,((p \rightarrow \mathbf{A}\,[\,\neg R\;\mathbf{U}\,(s \wedge \neg R)\,]\,) \vee \ldots \; \mathbf{AG}\,(\neg R))\;\mathbf{W}\;R\,]\,) \tag{60}$$

After Q until R $$\mathbf{AG}\,(Q \wedge \neg R \rightarrow \mathbf{A}\,[\,(p \rightarrow \mathbf{A}\,[\,\neg R\;\mathbf{U}\,(s \wedge \neg R)\,]\,)\;\mathbf{W}\;R\,]\,) \tag{61}$$

Examples and known uses: Response properties occur quite commonly in specifications of concurrent systems. Perhaps the most common example is in describing a requirement that a resource must be granted after it is requested.

Relationships: Note that a response property is like a converse of a precedence property. Precedence says that some cause precedes each effect, and response says that some effect follows each cause. They are not equivalent, because a response allows effects to occur without causes (precedence similarly allows causes to occur without subsequent effects).

Note that this pattern does not require that each occurrence of a cause will have its own occurrence of an effect."

E.2.3 Response chain property pattern

"***Intent:*** This is a scalable pattern. We describe the intent of the 1 stimulus – 2 response version here.

To describe a relationship between a stimulus event (p) and a sequence of two response events (s, t) in which the occurrence of the stimulus event must be followed by an occurrence of the sequence of response events within the scope. In state-based formalisms, the states satisfying the response must be distinct (i.e., s and t must be true in different states to count as a response), but the response may be satisfied by the same state as the stimulus (i.e., p and s may be true in the same state).

Mappings for CTL:

s, t responds to p:

Globally $$\mathbf{AG}\,(p \rightarrow \mathbf{AF}\,(s \wedge \mathbf{AX}\,(\mathbf{AF}\,(t)))) \tag{62}$$

Before R $$\neg\mathbf{E}\,[\,\neg R\;\mathbf{U}\,(p \wedge \neg R \wedge (\mathbf{E}\,[\,\neg s\;\mathbf{U}\;R\,] \vee \mathbf{E}\,[\,\neg R\;\mathbf{U}\,(s \wedge \neg R \wedge \mathbf{EX}\,(\mathbf{E}\,[\,\neg t\;\mathbf{U}\;R\,]\,))\,]\,))\,] \tag{63}$$

After Q $$\neg\mathbf{E}\,[\,\neg Q\;\mathbf{U}\,(Q \wedge \mathbf{EF}\,(p \wedge (\mathbf{EG}\,(\neg s) \vee \mathbf{EF}\,(s \wedge \mathbf{EX}\,(\mathbf{EG}\,(\neg t))))))\,] \tag{64}$$

Between Q and R
$$\mathbf{AG}\,(\,Q \rightarrow \neg\mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,p \wedge \neg R \wedge (\,\mathbf{E}\,[\,\neg s\;\mathbf{U}\;R\,] \vee \mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,s \wedge \neg R \wedge \mathbf{EX}\,(\,\mathbf{E}\,[\,\neg t\;\mathbf{U}\;R\,]\,))\,]\,))\,]\,) \tag{65}$$

After Q until R
$$\mathbf{AG}\,(\,Q \rightarrow \neg\mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,p \wedge \neg R \wedge (\,\mathbf{E}\,[\,\neg s\;\mathbf{U}\;R\,] \vee \mathbf{EG}\,(\,\neg s \wedge \neg R) \vee \mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,s \wedge \neg R \wedge \mathbf{EX}\,(\,\mathbf{E}\,[\,\neg t\;\mathbf{U}\;R\,] \vee \mathbf{EG}\,(\,\neg t \wedge \neg R\,)))\,]\,))\,]\,) \tag{66}$$

Examples and known uses: If a resource allocator grants a process access to a resource (*GrantRes*), the process will start using the resource (*BeginRes*) and finish using the resource (*EndRes*).

Relationships: This pattern is a generalization of the response pattern. If you wish to restrict the occurrence of states/events between stimuli or responses use the constrained response chain pattern."

E.2.4 Precedence chain property pattern

"***Intent:*** This is a scalable pattern. We describe the intent of the 1 stimulus – 2 response version here.

To describe a relationship between an event/state p and a sequence of two events/states (s, t) in which the occurrence of s followed by t within the scope must be preceded by an occurrence of the sequence p within the same scope. In state-based formalisms, the beginning of the enabled sequence (s, t) may be satisfied by the same state as the enabling condition (i.e., p and s may be true in the same state).

Mappings for CTL:

p precedes s, t:

Globally
$$\neg\mathbf{E}\,[\,\neg p\;\mathbf{U}\,(\,s \wedge \neg p \wedge \mathbf{EX}\,(\,\mathbf{EF}\,(\,t\,)))\,] \tag{67}$$

Before R
$$\neg\mathbf{E}\,[\,(\,\neg p \wedge \neg R\,)\;\mathbf{U}\,(\,s \wedge \neg p \wedge \neg R \wedge \mathbf{EX}\,(\,\mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,t \wedge \neg R\,)\,]\,))\,] \tag{68}$$

After Q
$$\neg\mathbf{E}\,[\,\neg Q\;\mathbf{U}\,(\,Q \wedge \mathbf{E}\,[\,\neg p\;\mathbf{U}\,(\,s \wedge \neg p \wedge \mathbf{EX}\,(\,\mathbf{EF}\,(\,t\,)))\,]\,)\,] \tag{69}$$

Between Q and R
$$\mathbf{AG}\,(\,Q \rightarrow \neg\mathbf{E}\,[\,(\,\neg p \wedge \neg R\,)\;\mathbf{U}\,(\,s \wedge \neg p \wedge \neg R \wedge \ldots\; \mathbf{EX}\,(\,\mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,t \wedge \neg \mathrm{R} \wedge \mathbf{EF}\,(\,R\,))\,]\,))\,]\,) \tag{70}$$

After Q until R
$$\mathbf{AG}\,(\,Q \rightarrow \neg\mathbf{E}\,[\,(\,\neg p \wedge \neg R\,)\;\mathbf{U}\,(\,s \wedge \neg p \wedge \neg R \wedge \ldots\; \mathbf{EX}\,(\,\mathbf{E}\,[\,\neg R\;\mathbf{U}\,(\,t \wedge \neg \mathrm{R}\,)\,]\,))\,]\,) \tag{71}$$

Examples and known uses: An example of this pattern, assuming reliable communication between client and server, is that 'If a client makes a request and there is no response, then the server must have crashed.' This would be expressed by parameterizing the constrained variant of the 1-2 precedence chain pattern as: *ServerCrash* precedes *ClientRequest*, $\mathbf{G}\;\neg$*Response* without *Response* in LTL.

Relationships: Note that this pattern does not require that each occurrence of the enabled sequence will have its own occurrence of the enabling condition."

E.3 Property specification pattern notes

"The patterns provided in this system cover a broad range of requirements for real systems. Your requirement, however, may require you to adapt existing patterns slightly to better express your intended property. There are a number of ways in which this variation can take place, e.g., parameterization of patterns, combination of patterns, and variation in pattern scopes.

Pattern parameterization

"Pattern mappings are presented in terms of place-holder symbols (e.g., *p*, *Q*, *R*, *s*) that are to be replaced by users when writing actual specifications. These place holders are filled with descriptions of specific system states or event of interest. These descriptions can be more complex than just a single proposition or event name. Here are a few examples (for logics, e.g., CTL, LTL):

- Purely propositional formula can always be used to describe a state. This includes simple negations, disjunctions, conjunctions, and implications.
- State-formulae that include temporal operators can also be used (…). Care must be taken in using such state-formulae, since the meaning of the resulting specification can be quite subtle. This is especially true when using scopes that may have an end point, i.e., before, between, and after-until."

Pattern combinations

"A system's specification usually consists of a collection of property specifications.

Conjunctions: It is most often the case that all such property specifications should hold. In this case, one could simply check all specifications individually and require that all are successful. For the logical formalisms, an alternative is to conjoin the individual specifications into a single larger specification. While this is semantically equivalent, it may be the case that a larger specification is less efficient to verify (e.g., the cost of LTL in Buchi automaton construction can be large for automata-based model checkers). For this reason, preserving, and verifying, the individual property specifications is suggested.

Disjunctions: There are two views of pairs (or more generally a collection) of alternative individual property specifications:

- System behaviors all correspond to one specification or they all correspond to the other specification.
- Some of the system behaviors correspond to one specification and the rest of the behaviors correspond to the other specification.

The first of these alternatives corresponds to the checking of individual property specifications independently and disjoint the results. This is true for all specification formalisms. The latter situation can be achieved in formalisms that allow specifications to be disjoined under the same universal path qualifier.

For LTL and Quantified Regular Expression there is an implicit outer universal path qualifier, thus checking of a top-level disjunction of specifications will achieve these semantics. To achieve the first alternative (above) one must check LTL and Quantified Regular Expression specifications separately.

This is not the case for CTL, where two specifications cannot be disjoined directly under the same path quantifier. A top-level disjunction CTL achieves the first alternative and the second cannot be achieved directly (although one might be able to rewrite a combined version of the two specifications)."

Scope variations

"Most event-based formalisms use some version of an interleaved model of concurrent computation. In such formalisms, two events cannot coincide. Event-delimited scopes are thus open at both ends: an event that occurs within the scope cannot occur at the same time as an event that marks the beginning or end of the scope. For state-based formalisms, the situation is different. Consider, for instance, a scope that begins with a state in which proposition *Q* holds and ends with the next state in which *R* holds. If we want to specify that proposition *p* does not hold within the scope, we have to decide what should happen if *p* is true at either of the states marking the endpoints of the scope."

Appendix F

FDCML-based description of the demonstration control device

The basis for the evaluation of DSE is a very detailed description of the current system state. Appropriately, also each control device needs to be described with all its parameters. In order to support the engineering process, a device description based on the FDCML format has been proposed in Section 5.3. The following listing includes excerpts of the device description of the demonstration control device (see also Section 8.1.1) without any control applications already included. If an ACS customer buys a new control device, a similar situation occurs and the device vendor would provide an analogue device description file as presented above for the demonstration control device.

```
<?xml version="1.0" encoding="UTF-8"?>
<hcd:ISO15745Profile xmlns:hcd="http://www.ecedac.org/61499/hcd" xmlns:data="http://www.ecedac.org/61499/data"
  xmlns:lib="http://www.ecedac.org/61499/lib" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ecedac.org/61499/hcd eCEDAC_HCD.xsd http://www.ecedac.org/61499/data
DataType_elements_cs.xsd http://www.ecedac.org/61499/lib LibraryElement_elements_cs.xsd ">

  <hcd:ProfileHeader>
    <ProfileIdentification>eCEDAC HCD</ProfileIdentification>
    <ProfileRevision>V 0.1</ProfileRevision>
    <ProfileName> eCEDAC Hardware Capability Description </ProfileName>
    <ProfileSource>eCEDAC Consortium</ProfileSource>
    <ProfileDate>2006-11-14</ProfileDate>
  </hcd:ProfileHeader>

  <hcd:ProfileBody fileCreationDate="2007-05-30" fileCreator="Christoph Suender" fileModificationDate="2007-05-30"
    fileName="DevBoard_ARM7.xml" fileVersion="V0.1">

    <hcd:DeviceIdentity>
      <hcd:vendorName>
        <hcd:label>PHYTEC Messtechnik GmbH</hcd:label>
      </hcd:vendorName>
      <hcd:productName>
        <hcd:label>phyCORE-AT91M55800A</hcd:label>
      </hcd:productName>
      <hcd:productText>
        <hcd:label> phyCORE Development Board HD200 mit phyCORE-AT91M55800A </hcd:label>
      </hcd:productText>
    </hcd:DeviceIdentity>

    <hcd:DeviceManager>
      <hcd:deviceStructure>
        <hcd:channelList>
          <hcd:channel uniqueID="Supply.VCC" channelType="Supply" direction="I">
            <hcd:label>X1</hcd:label>
            <hcd:help>Controller Supply Voltage</hcd:help>
            <hcd:specificProperty propertyType="Voltage">
              <hcd:label>nomVoltage [V]</hcd:label>
              <hcd:help> Nominal supply voltage of the controller measured in Volts [V] </hcd:help>
              <hcd:instanceValue>5</hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty propertyType="Current">
              <hcd:label>maxCurrent [mA]</hcd:label>
              <hcd:help> Maximum current consumption of the cntroller measured in Miliamperes [mA] </hcd:help>
              <hcd:instanceValue>500</hcd:instanceValue>
            </hcd:specificProperty>
          </hcd:channel>
```

List of further channels listed here in the original description file.

```
  </hcd:channelList>
  <hcd:MAUList>
    <hcd:MAU uniqueID="TP1" interfaceType="RJ45" protocol="10BaseT Ethernet IEEE 802.3">
      <hcd:label>TP</hcd:label>
      <hcd:help>Ethernet network connector</hcd:help>
      <hcd:specificProperty
        propertyType="DataTransmissionSpeed">
        <hcd:label>maxDataSpeed [MBit/s]</hcd:label>
        <hcd:help> Maximum data transmission speed measured in [MBit/s] </hcd:help>
        <hcd:instanceValue>10</hcd:instanceValue>
      </hcd:specificProperty>
      <hcd:specificProperty
        propertyType="MACAddress">
        <hcd:label>MAC Address</hcd:label>
        <hcd:help> printed on the bar code sticker attached to the phyCORE module </hcd:help>
        <hcd:instanceValue> 00:00:00:00:00:00  </hcd:instanceValue>
      </hcd:specificProperty>
      <hcd:specificProperty
        propertyType="MediaType">
        <hcd:label>full/half duplex</hcd:label>
        <hcd:instanceValue> needs to be defined </hcd:instanceValue>
      </hcd:specificProperty>
    </hcd:MAU>
```

List of further MAUs listed here in the original description file.

```
  </hcd:MAUList>
  <hcd:indicatorList>
    <hcd:LEDList>
      <hcd:LED LEDType="DeviceStatus" uniqueID="PowerON">
        <hcd:label>D2 - PowerON</hcd:label>
        <hcd:specificProperty propertyType="Colors">
          <hcd:label>LED Colors</hcd:label>
          <hcd:instanceValue> red </hcd:instanceValue>
        </hcd:specificProperty>
        <hcd:specificProperty
          propertyType="Programmable">
          <hcd:label> LED's ability to be programmable </hcd:label>
          <hcd:instanceValue> false </hcd:instanceValue>
        </hcd:specificProperty>
      </hcd:LED>
```

List of further LEDs listed here in the original description file.

```
    </hcd:LEDList>
  </hcd:indicatorList>
</hcd:deviceStructure>

<hcd:communicationEntity protocol="TCP/IP" uniqueID="TCP" enabled="YES">
  <hcd:label>TCP/IP</hcd:label>
  <hcd:identity>
```

```
        <hcd:vendorName> <hcd:label>Beck</hcd:label> </hcd:vendorName>
        <hcd:typeName> <hcd:label>TCP Stack</hcd:label> </hcd:typeName>
      </hcd:identity>
      <hcd:cfgItemList>
        <hcd:label>Protocol Settings</hcd:label>
        <hcd:dedicatedCfgItem uniqueID="TCP_IPAddress" dedicatedCfgItemType="IPAddress">
          <hcd:label>IP Address</hcd:label>
          <hcd:instanceValue> 128.130.200.162 </hcd:instanceValue>
        </hcd:dedicatedCfgItem>
        <hcd:dedicatedCfgItem uniqueID="TCP_SubnetMask" dedicatedCfgItemType="SubnetMask">
          <hcd:label>Subnet Mask</hcd:label>
          <hcd:instanceValue> 255.255.255.128 </hcd:instanceValue>
        </hcd:dedicatedCfgItem>
      </hcd:cfgItemList>
      <hcd:MAUUsageList>
        <hcd:MAUUsage ref="TP1"></hcd:MAUUsage>
      </hcd:MAUUsageList>
    </hcd:communicationEntity>
```

List of further communicationEntity elements listed here in the original description file.

```
    <hcd:resourceEntity resourceType="ComputationUnit" uniqueID="cpu0">
      <hcd:label>phyCORE-AT91M55800A</hcd:label>
      <hcd:help>single board computer module</hcd:help>
      <hcd:identity>
        <hcd:vendorName> <hcd:label>Phytec</hcd:label> </hcd:vendorName>
        <hcd:typeName> <hcd:label>L-618e_3</hcd:label> </hcd:typeName>
      </hcd:identity>
      <hcd:additionalItemList
        additionalItemsType="ProcessingUnit">
        <hcd:label>ProcessingUnit</hcd:label>
        <hcd:additionalItem additionalItemType="Processor" uniqueID="Processor0">
          <hcd:label>Processor0</hcd:label>
          <hcd:instanceValue> AT91M55800A </hcd:instanceValue>
          <hcd:specificProperty propertyType="Vendor">
            <hcd:label>Vendor</hcd:label>
            <hcd:instanceValue>ATMEL</hcd:instanceValue>
          </hcd:specificProperty>
          <hcd:specificProperty
            propertyType="ClockRate">
            <hcd:label>ClockRate</hcd:label>
            <hcd:help> The processor's clock rate measured in [MHz] </hcd:help>
            <hcd:instanceValue> 32 MHz </hcd:instanceValue>
          </hcd:specificProperty>
```

List of further specificProperty elements listed here in the original description file.

```
        </hcd:additionalItem>
      </hcd:additionalItemList>
      <hcd:additionalItemList
        additionalItemsType="MemoryUnit">
        <hcd:label>MemoryUnit</hcd:label>
        <hcd:additionalItem additionalItemType="Memory" uniqueID="Memory0">
          <hcd:label>Memory0</hcd:label>
          <hcd:instanceValue>U6</hcd:instanceValue>
          <hcd:specificProperty propertyType="Type">
            <hcd:label>Type</hcd:label>
            <hcd:instanceValue>Flash</hcd:instanceValue>
          </hcd:specificProperty>
          <hcd:specificProperty
            propertyType="MemorySize">
            <hcd:label>MemorySize</hcd:label>
            <hcd:help> Memory size measured in [KB] </hcd:help>
```

```
                <hcd:instanceValue> 4096 KB </hcd:instanceValue>
            </hcd:specificProperty>
```

List of further specificProperty elements listed here in the original description file.

```
        </hcd:additionalItem>
        <hcd:additionalItem additionalItemType="Memory" uniqueID="Memory1">
            <hcd:label>Memory1</hcd:label>
            <hcd:instanceValue> U14, U15, U16, U17 </hcd:instanceValue>
            <hcd:specificProperty propertyType="Type">
                <hcd:label>Type</hcd:label>
                <hcd:instanceValue>SRAM</hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty propertyType="MemorySize">
                <hcd:label>MemorySize</hcd:label>
                <hcd:help> Memory size measured in [KB] </hcd:help>
                <hcd:instanceValue> 2048 KB </hcd:instanceValue>
            </hcd:specificProperty>
```

List of further specificProperty elements listed here in the original description file.

```
        </hcd:additionalItem>
    </hcd:additionalItemList>
</hcd:resourceEntity>
```

List of further resourceEntity elements listed here in the original description file.

```
<hcd:resourceEntity resourceType="OperatingSystem" uniqueID="os0">
    <hcd:label>eCos</hcd:label>
    <hcd:identity>
        <hcd:vendorName> <hcd:label>Open Source</hcd:label> </hcd:vendorName>
        <hcd:typeName> <hcd:label>Realtime OS</hcd:label> </hcd:typeName>
    </hcd:identity>
    <hcd:additionalItemList additionalItemsType="OSCharacteristics">
        <hcd:label>OSCharacteristics</hcd:label>
        <hcd:additionalItem additionalItemType="Scheduler" uniqueID="Scheduler0">
            <hcd:label>Scheduler0</hcd:label>
            <hcd:instanceValue> Bitmap Scheduler </hcd:instanceValue>
            <hcd:specificProperty propertyType="Type">
                <hcd:label>Type</hcd:label>
                <hcd:instanceValue> Bitmap Scheduler </hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty propertyType="numberPriorities">
                <hcd:label>numberPriorities</hcd:label>
                <hcd:instanceValue>32</hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty propertyType="numberThreadsperPriority">
                <hcd:label> numberThreadsperPriority </hcd:label>
                <hcd:instanceValue>1</hcd:instanceValue>
            </hcd:specificProperty>
        </hcd:additionalItem>
        <hcd:additionalItem additionalItemType="Scheduler" uniqueID="Scheduler1">
            <hcd:label>Scheduler1</hcd:label>
            <hcd:instanceValue> Multi Level Queue Scheduler </hcd:instanceValue>
            <hcd:specificProperty propertyType="Type">
                <hcd:label>Type</hcd:label>
                <hcd:instanceValue>MLQ</hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty
                propertyType="numberPriorities">
                <hcd:label>numberPriorities</hcd:label>
                <hcd:instanceValue>32</hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty propertyType="numberThreadsperPrioritiy">
                <hcd:label> numberThreadsperPriority </hcd:label>
                <hcd:instanceValue> infinity </hcd:instanceValue>
            </hcd:specificProperty>
         </hcd:additionalItem>
         <hcd:additionalItem
            additionalItemType="CompilerOptions"
      <hcd:additionalItemList
         additionalItemsType="OSConfiguration">
         <hcd:label>OSConfiguration</hcd:label>
         <hcd:additionalItem additionalItemType="Scheduler" uniqueID="Scheduler_current">
            <hcd:label>Scheduler_current</hcd:label>
            <hcd:instanceValue> Scheduler1 </hcd:instanceValue>
         </hcd:additionalItem>
      </hcd:additionalItemList>
      <hcd:additionalItemList
         additionalItemsType="ModelExecutionTime">
         <hcd:label>ModelExecutionTime</hcd:label>
         <hcd:additionalItem additionalItemType="TaskSwitch" uniqueID="TaskSwitch">
            <hcd:label>TaskSwitch</hcd:label>
            <hcd:instanceValue>820</hcd:instanceValue>
         </hcd:additionalItem>
         <hcd:additionalItem
            additionalItemType="ThreadSuspension" uniqueID="ThreadSuspension">
            <hcd:label>ThreadSuspension</hcd:label>
            <hcd:instanceValue>101</hcd:instanceValue>
         </hcd:additionalItem>
         <hcd:additionalItem
            additionalItemType="ThreadResumption" uniqueID="ThreadResumption">
            <hcd:label>ThreadResumption</hcd:label>
            <hcd:instanceValue>133</hcd:instanceValue>
         </hcd:additionalItem>
      </hcd:additionalItemList>
      <hcd:internalConnectionPointList>
         <hcd:internalConnectionPoint ref="cpu0" uniqueID="os0_with_cpu0" enabled="YES">
         </hcd:internalConnectionPoint>
      </hcd:internalConnectionPointList>
   </hcd:resourceEntity>

   <hcd:resourceEntity resourceType="IEC61499Runtime" uniqueID="rt1499_0">
      <hcd:label>MARTE</hcd:label>
      <hcd:identity>
         <hcd:vendorName> <hcd:label> Alois Zoitl </hcd:label> </hcd:vendorName>
         <hcd:typeName>
            <hcd:label> IEC 61499 Runtime Environment </hcd:label>
         </hcd:typeName>
      </hcd:identity>
      <hcd:additionalItemList
         additionalItemsType="ModelExecutionTime">
         <hcd:label>ModelExecutionTime</hcd:label>
         <hcd:additionalItem
            additionalItemType="DispatcherParams" uniqueID="DispatcherParams">
            <hcd:label>DispatcherParams</hcd:label>
            <hcd:instanceValue> DispatcherParams </hcd:instanceValue>
            <hcd:specificProperty
               propertyType="VerificationParam">
               <hcd:label>T_FETCH</hcd:label>
               <hcd:help>Ticks in 0.1 µs</hcd:help>
               <hcd:instanceValue> 29,6 µs </hcd:instanceValue>
            </hcd:specificProperty>
            <hcd:specificProperty
               propertyType="VerificationParam">
               <hcd:label>T_EV_OUT</hcd:label>
               <hcd:help>Ticks in 0.1µs</hcd:help>
               <hcd:instanceValue> 27,4 µs </hcd:instanceValue>
            </hcd:specificProperty>
          </hcd:additionalItem>
```

List of further additionalItem elements listed here in the original description file.

```
        </hcd:additionalItemList>
        <hcd:additionalItemList additionalItemsType="TypeLibraryParameters">
          <hcd:label>TypeLibraryParameters</hcd:label>
```

List of additionalItem elements listed here in the original description file, e.g. FB Types, Data Types, Resource Types.

```
        </hcd:additionalItemList>
        <hcd:additionalItemList additionalItemsType="ComplianceProfiles">
          <hcd:label> supported IEC 61499 Compliance Profiles </hcd:label>
          <hcd:additionalItem additionalItemType="ComplianceProfile" uniqueID="CPFD">
            <hcd:label>CPFD</hcd:label>
            <hcd:instanceValue> IEC 61499 Compliance Profile for Feasibility Demonstration </hcd:instanceValue>
          </hcd:additionalItem>
        </hcd:additionalItemList>

        <hcd:IEC61499TypeLibrary>
```

List of type definitions available within the control device according to the XML format defined in IEC 61499-2 (2005, Annex A).

```
        </hcd:IEC61499TypeLibrary>

        <hcd:internalConnectionPointList>
          <hcd:internalConnectionPoint ref="cpu0" uniqueID="rt1499_0_with_cpu0" enabled="YES">
          </hcd:internalConnectionPoint>
        </hcd:internalConnectionPointList>
      </hcd:resourceEntity>

    </hcd:DeviceManager>

  </hcd:ProfileBody>

</hcd:ISO15745Profile>
```

Appendix G

Additional information for demonstration example

This appendix includes additional information for the demonstration example used in Section 8.1. The following aspects will be depicted in the figures presented here without further detailed explaination:

- Figure 72: Internal model of 'Subtract' depicted in Figure 43
- Figure 73: Control application after execution of RINIT sequence
- Figure 74: Control application after execution of RECONF sequence
- Figure 75: Control application after execution of RDINIT sequence
- Table 7: Status output of dependent operation check within Page 3 (RINIT sequence)
- Table 8: Status output of dependent operation check within Page 4 (RECONF sequence)
- Table 9: Status output of dependent operation check within Page 5 (RDINIT sequence)
- Figure 76: NCES model for practical example (Addition/Subtracion)
- Figure 77: NCES module 'Thread_APP', the control application of Addition/Subtraction example at the beginning of the RECONF sequence
- Figure 78: NCES module 'Thread_RECONF', the RECONF seqeunce of ECA within Addition/Subtraction example
- Figure 79: NCES model of Addition/Subtraction example without downtimeless system evolution
- Figure 80: NCES model for position controller including plant model (velocity closed-loop control)

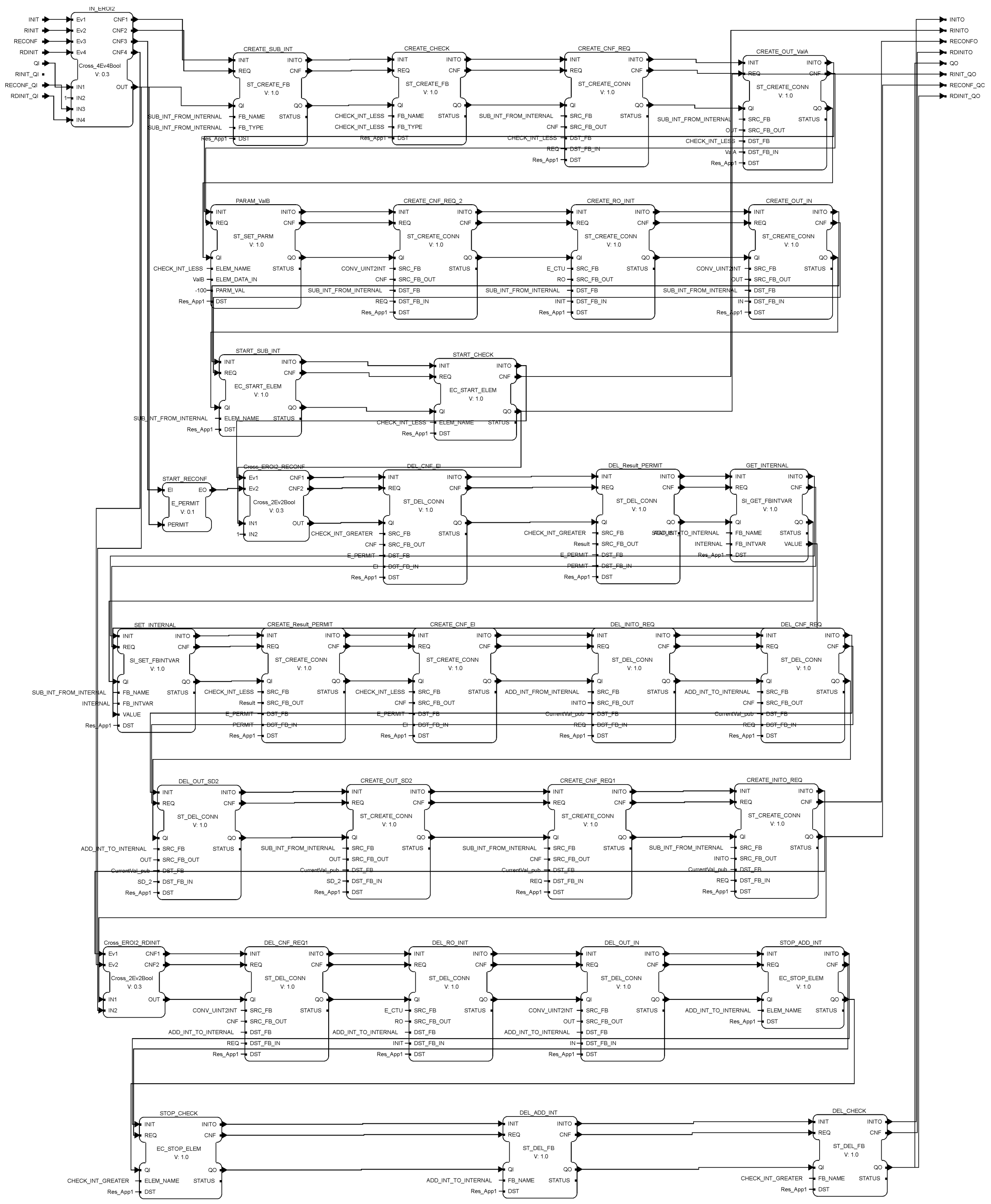


Figure 72: Internal model of 'Subtract' depicted in Figure 43

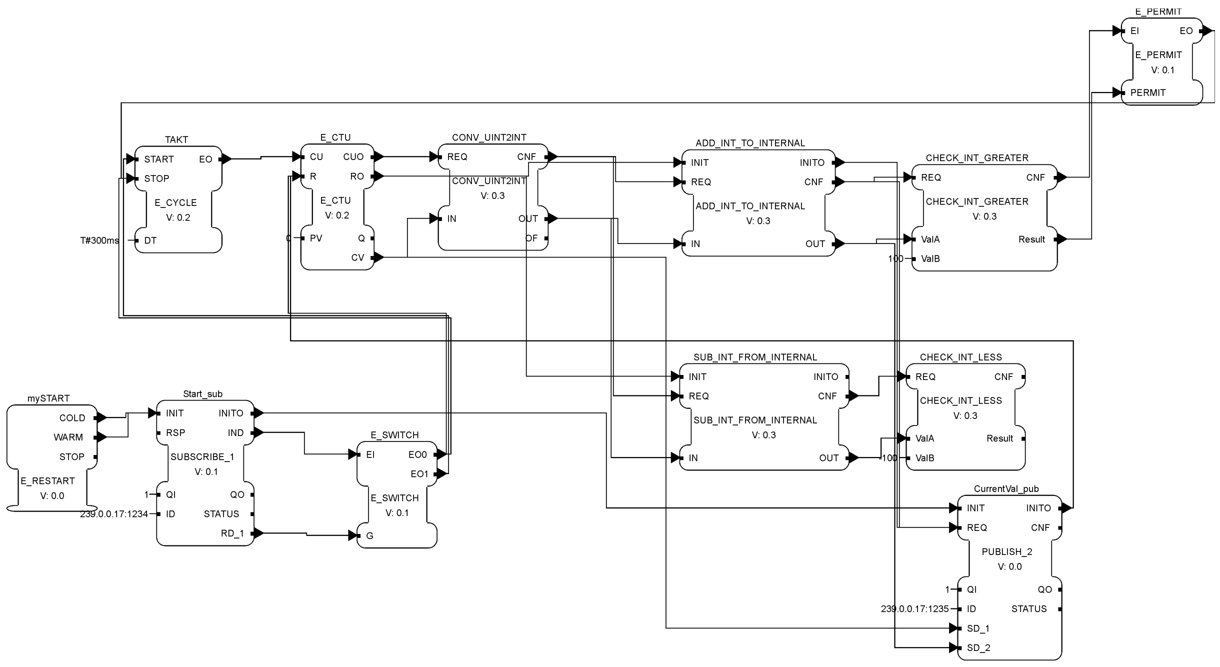


Figure 73: Control application after execution of RINIT sequence

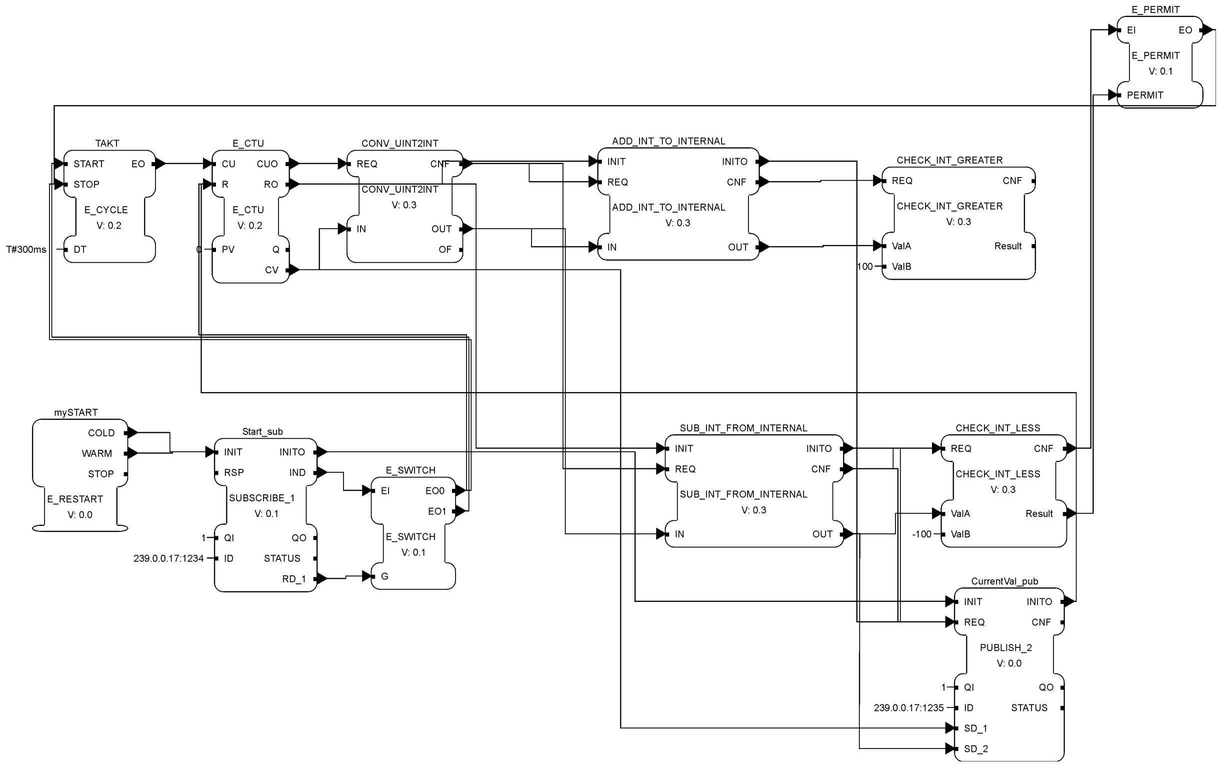


Figure 74: Control application after execution of RECONF sequence

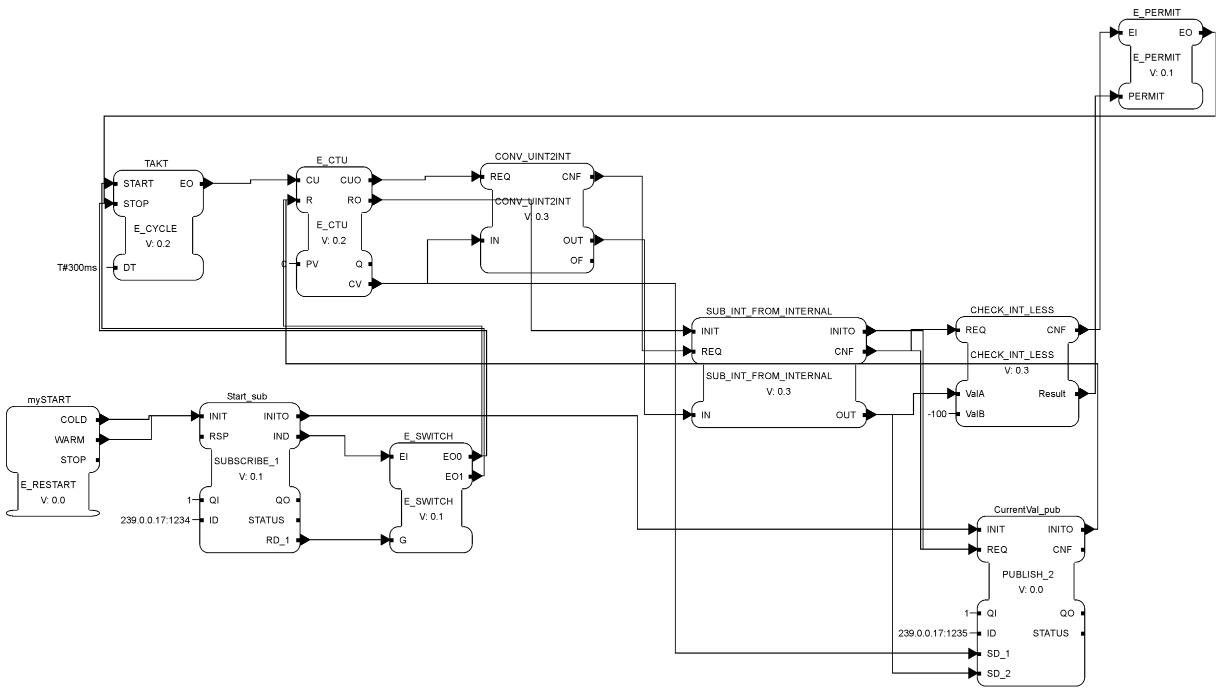


Figure 75: Control application after execution of RDINIT sequence

Initial RINIT system state calculated!	Establish a copy of the system state.
List of FBs contained in RINIT sequence: IN_EROI2 CREATE_SUB_INT CREATE_CHECK CREATE_CNF_REQ CREATE_OUT_ValA PARAM_ValB CREATE_CNF_REQ_2 CREATE_RO_INIT CREATE_OUT_IN START_SUB_INT START_CHECK	Extract a list of FBs that are executed within the RINIT sequence.
CALCULATE Logical order rule: Param: CREATE_SUB_INT.FB_NAME = SUB_INT_FROM_INTERNAL Param: CREATE_SUB_INT.FB_TYPE = SUB_INT_FROM_INTERNAL Param: CREATE_SUB_INT.DST = Res_App1 Destination resource Res_App1 is valid. FB type SUB_INT_FROM_INTERNAL is present.	First basic reconfiguration service: • CREATE FB • Target resource available • Type available
Param: CREATE_CHECK.FB_NAME = CHECK_INT_LESS Param: CREATE_CHECK.FB_TYPE = CHECK_INT_LESS Param: CREATE_CHECK.DST = Res_App1 Destination resource Res_App1 is valid. FB type CHECK_INT_LESS is present.	Second basic reconfiguration service: • CREATE FB • Target resource available • Type available

Param: CREATE_CNF_REQ.SRC_FB = SUB_INT_FROM_INTERNAL Param: CREATE_CNF_REQ.SRC_FB_OUT = CNF Param: CREATE_CNF_REQ.DST_FB = CHECK_INT_LESS Param: CREATE_CNF_REQ.DST_FB_IN = REQ Param: CREATE_CNF_REQ.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points SUB_INT_FROM_INTERNAL.CNF, CHECK_INT_LESS.REQ are valid.	Third basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: CREATE_OUT_ValA.SRC_FB = SUB_INT_FROM_INTERNAL Param: CREATE_OUT_ValA.SRC_FB_OUT = OUT Param: CREATE_OUT_ValA.DST_FB = CHECK_INT_LESS Param: CREATE_OUT_ValA.DST_FB_IN = ValA Param: CREATE_OUT_ValA.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points SUB_INT_FROM_INTERNAL.OUT, CHECK_INT_LESS.ValA are valid.	Fourth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: PARAM_ValB.ELEM_NAME = CHECK_INT_LESS Param: PARAM_ValB.ELEM_DATA_IN = ValB Param: PARAM_ValB.PARM_VAL = -100 Param: PARAM_ValB.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB CHECK_INT_LESS is valid. Destination ELEM_DATA_IN ValB is valid.	Fifth basic reconfiguration service • WRITE parameter • Target resource available • Target FB available • Target parameter available
Param: CREATE_CNF_REQ_2.SRC_FB = CONV_UINT2INT Param: CREATE_CNF_REQ_2.SRC_FB_OUT = CNF Param: CREATE_CNF_REQ_2.DST_FB = SUB_INT_FROM_INTERNAL Param: CREATE_CNF_REQ_2.DST_FB_IN = REQ Param: CREATE_CNF_REQ_2.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CONV_UINT2INT.CNF, SUB_INT_FROM_INTERNAL.REQ are valid.	Sixth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: CREATE_RO_INIT.SRC_FB = E_CTU Param: CREATE_RO_INIT.SRC_FB_OUT = RO Param: CREATE_RO_INIT.DST_FB = SUB_INT_FROM_INTERNAL Param: CREATE_RO_INIT.DST_FB_IN = INIT Param: CREATE_RO_INIT.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points E_CTU.RO, SUB_INT_FROM_INTERNAL.INIT are valid.	Seventh basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: CREATE_OUT_IN.SRC_FB = CONV_UINT2INT Param: CREATE_OUT_IN.SRC_FB_OUT = OUT Param: CREATE_OUT_IN.DST_FB = SUB_INT_FROM_INTERNAL Param: CREATE_OUT_IN.DST_FB_IN = IN Param: CREATE_OUT_IN.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CONV_UINT2INT.OUT, SUB_INT_FROM_INTERNAL.IN are valid.	Eighth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: START_SUB_INT.ELEM_NAME = SUB_INT_FROM_INTERNAL Param: START_SUB_INT.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB SUB_INT_FROM_INTERNAL is valid.	Nineth basic reconfiguration service: • START element • Target resource available • FB available
Param: START_CHECK.ELEM_NAME = CHECK_INT_LESS	Tenth basic reconfiguration service: • START element

Param: START_CHECK.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB CHECK_INT_LESS is valid.	• Target resource available • FB available

Table 7: Status output of dependent operation check within Page 3 (RINIT sequence)

Initial RECONF system state calculated!	Establish a copy of the system state.
List of FBs contained in RECONF sequence: IN_EROI2 START_RECONF Cross_EROI2_RECONF DEL_CNF_EI DEL_Result_PERMIT GET_INTERNAL SET_INTERNAL CREATE_Result_PERMIT CREATE_CNF_EI DEL_INITO_REQ DEL_CNF_REQ DEL_OUT_SD2 CREATE_OUT_SD2 CREATE_CNF_REQ1 CREATE_INITO_REQ	Extract a list of FBs that are executed within the RINIT sequence.
CALCULATE Logical order rule: Param: DEL_CNF_EI.SRC_FB = CHECK_INT_GREATER Param: DEL_CNF_EI.SRC_FB_OUT = CNF Param: DEL_CNF_EI.DST_FB = E_PERMIT Param: DEL_CNF_EI.DST_FB_IN = EI Param: DEL_CNF_EI.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CHECK_INT_GREATER.CNF, E_PERMIT.EI are valid. Connection CHECK_INT_GREATER.CNF-E_PERMIT.EI is present.	First basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: DEL_Result_PERMIT.SRC_FB = CHECK_INT_GREATER Param: DEL_Result_PERMIT.SRC_FB_OUT = Result Param: DEL_Result_PERMIT.DST_FB = E_PERMIT Param: DEL_Result_PERMIT.DST_FB_IN = PERMIT Param: DEL_Result_PERMIT.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CHECK_INT_GREATER.Result, E_PERMIT.PERMIT are valid. Connection CHECK_INT_GREATER.Result-E_PERMIT.PERMIT is present.	Second basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: GET_INTERNAL.FB_NAME = ADD_INT_TO_INTERNAL Param: GET_INTERNAL.FB_INTVAR = INTERNAL Param: GET_INTERNAL.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB ADD_INT_TO_INTERNAL is valid. Internal Variable INTERNAL is valid.	Third basic reconfiguration service: • READ variable • Target resource available • FB and variable available
Param: SET_INTERNAL.FB_NAME = SUB_INT_FROM_INTERNAL Param: SET_INTERNAL.FB_INTVAR = INTERNAL Param: SET_INTERNAL.DST = Res_App1 Destination resource Res_App1 is valid.	Fourth basic reconfiguration service: • WRITE variable • Target resource available

Destination FB SUB_INT_FROM_INTERNAL is valid. Internal Variable INTERNAL is valid.	• FB and variable available
Param: CREATE_Result_PERMIT.SRC_FB = CHECK_INT_LESS Param: CREATE_Result_PERMIT.SRC_FB_OUT = Result Param: CREATE_Result_PERMIT.DST_FB = E_PERMIT Param: CREATE_Result_PERMIT.DST_FB_IN = PERMIT Param: CREATE_Result_PERMIT.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CHECK_INT_LESS.Result, E_PERMIT.PERMIT are valid.	Fifth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: CREATE_CNF_EI.SRC_FB = CHECK_INT_LESS Param: CREATE_CNF_EI.SRC_FB_OUT = CNF Param: CREATE_CNF_EI.DST_FB = E_PERMIT Param: CREATE_CNF_EI.DST_FB_IN = EI Param: CREATE_CNF_EI.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CHECK_INT_LESS.CNF, E_PERMIT.EI are valid.	Sixth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: DEL_INITO_REQ.SRC_FB = ADD_INT_FROM_INTERNAL Param: DEL_INITO_REQ.SRC_FB_OUT = INITO Param: DEL_INITO_REQ.DST_FB = CurrentVal_pub Param: DEL_INITO_REQ.DST_FB_IN = REQ Param: DEL_INITO_REQ.DST = Res_App1 Destination resource Res_App1 is valid. At least one of the connection points ADD_INT_FROM_INTERNAL.INITO, CurrentVal_pub.REQ is invalid.	Seventh basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: DEL_CNF_REQ.SRC_FB = ADD_INT_TO_INTERNAL Param: DEL_CNF_REQ.SRC_FB_OUT = CNF Param: DEL_CNF_REQ.DST_FB = CurrentVal_pub Param: DEL_CNF_REQ.DST_FB_IN = REQ Param: DEL_CNF_REQ.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points ADD_INT_TO_INTERNAL.CNF, CurrentVal_pub.REQ are valid. Connection ADD_INT_TO_INTERNAL.CNF-CurrentVal_pub.REQ is present.	Eighth basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: DEL_OUT_SD2.SRC_FB = ADD_INT_TO_INTERNAL Param: DEL_OUT_SD2.SRC_FB_OUT = OUT Param: DEL_OUT_SD2.DST_FB = CurrentVal_pub Param: DEL_OUT_SD2.DST_FB_IN = SD_2 Param: DEL_OUT_SD2.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points ADD_INT_TO_INTERNAL.OUT, CurrentVal_pub.SD_2 are valid. Connection ADD_INT_TO_INTERNAL.OUT-CurrentVal_pub.SD_2 is present.	Ninth basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: CREATE_OUT_SD2.SRC_FB = SUB_INT_FROM_INTERNAL Param: CREATE_OUT_SD2.SRC_FB_OUT = OUT Param: CREATE_OUT_SD2.DST_FB = CurrentVal_pub Param: CREATE_OUT_SD2.DST_FB_IN = SD_2 Param: CREATE_OUT_SD2.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points SUB_INT_FROM_INTERNAL.OUT, CurrentVal_pub.SD_2 are valid.	Tenth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: CREATE_CNF_REQ1.SRC_FB = SUB_INT_FROM_INTERNAL	

Param: CREATE_CNF_REQ1.SRC_FB_OUT = CNF Param: CREATE_CNF_REQ1.DST_FB = CurrentVal_pub Param: CREATE_CNF_REQ1.DST_FB_IN = REQ Param: CREATE_CNF_REQ1.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points SUB_INT_FROM_INTERNAL.CNF, Current-Val_pub.REQ are valid.	Eleventh basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available
Param: CREATE_INITO_REQ.SRC_FB = SUB_INT_FROM_INTERNAL Param: CREATE_INITO_REQ.SRC_FB_OUT = INITO Param: CREATE_INITO_REQ.DST_FB = CurrentVal_pub Param: CREATE_INITO_REQ.DST_FB_IN = REQ Param: CREATE_INITO_REQ.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points SUB_INT_FROM_INTERNAL.INITO, CurrentVal_pub.REQ are valid.	Twelfth basic reconfiguration service: • CREATE connection • Target resource available • Source and destination available

Table 8: Status output of dependent operation check within Page 4 (RECONF sequence)

Initial RDINIT system state calculated!	Establish a copy of the system state.
List of FBs contained in RDINIT sequence: IN_EROI2 Cross_EROI2_RDINIT DEL_CNF_REQ1 DEL_RO_INIT DEL_OUT_IN STOP_ADD_INT STOP_CHECK DEL_ADD_INT DEL_CHECK	Extract a list of FBs that are executed within the RINIT sequence.
CALCULATE Logical order rule: Param: DEL_CNF_REQ1.SRC_FB = CONV_UINT2INT Param: DEL_CNF_REQ1.SRC_FB_OUT = CNF Param: DEL_CNF_REQ1.DST_FB = ADD_INT_TO_INTERNAL Param: DEL_CNF_REQ1.DST_FB_IN = REQ Param: DEL_CNF_REQ1.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CONV_UINT2INT.CNF, ADD_INT_TO_INTERNAL.REQ are valid. Connection CONV_UINT2INT.CNF-ADD_INT_TO_INTERNAL.REQ is present.	First basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: DEL_RO_INIT.SRC_FB = E_CTU Param: DEL_RO_INIT.SRC_FB_OUT = RO Param: DEL_RO_INIT.DST_FB = ADD_INT_TO_INTERNAL Param: DEL_RO_INIT.DST_FB_IN = INIT Param: DEL_RO_INIT.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points E_CTU.RO, ADD_INT_TO_INTERNAL.INIT are valid. Connection E_CTU.RO-ADD_INT_TO_INTERNAL.INIT is present.	Second basic reconfiguration service: • DELETE connection • Target resource available • Source and destination available
Param: DEL_OUT_IN.SRC_FB = CONV_UINT2INT Param: DEL_OUT_IN.SRC_FB_OUT = OUT	Third basic reconfiguration service: • DELETE connection

Param: DEL_OUT_IN.DST_FB = ADD_INT_TO_INTERNAL Param: DEL_OUT_IN.DST_FB_IN = IN Param: DEL_OUT_IN.DST = Res_App1 Destination resource Res_App1 is valid. Both connection points CONV_UINT2INT.OUT, ADD_INT_TO_INTERNAL.IN are valid. Connection CONV_UINT2INT.OUT-ADD_INT_TO_INTERNAL.IN is present.	• Target resource available • Source and destination available
Param: STOP_ADD_INT.ELEM_NAME = ADD_INT_TO_INTERNAL Param: STOP_ADD_INT.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB ADD_INT_TO_INTERNAL is valid.	Fourth basic reconfiguration service: • STOP element • Target resource available • FB instance available
Param: STOP_CHECK.ELEM_NAME = CHECK_INT_GREATER Param: STOP_CHECK.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB CHECK_INT_GREATER is valid.	Fifth basic reconfiguration service: • STOP element • Target resource available • FB instance available
Param: DEL_ADD_INT.FB_NAME = ADD_INT_TO_INTERNAL Param: DEL_ADD_INT.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB ADD_INT_TO_INTERNAL is valid.	Sixth basic reconfiguration service: • DELETE FB • Target resource available • FB instance available
Param: DEL_CHECK.FB_NAME = CHECK_INT_GREATER Param: DEL_CHECK.DST = Res_App1 Destination resource Res_App1 is valid. Destination FB CHECK_INT_GREATER is valid.	Seventh basic reconfiguration service: • DELETE FB • Target resource available • FB instance available

Table 9: Status output of dependent operation check within Page 5 (RDINIT sequence)

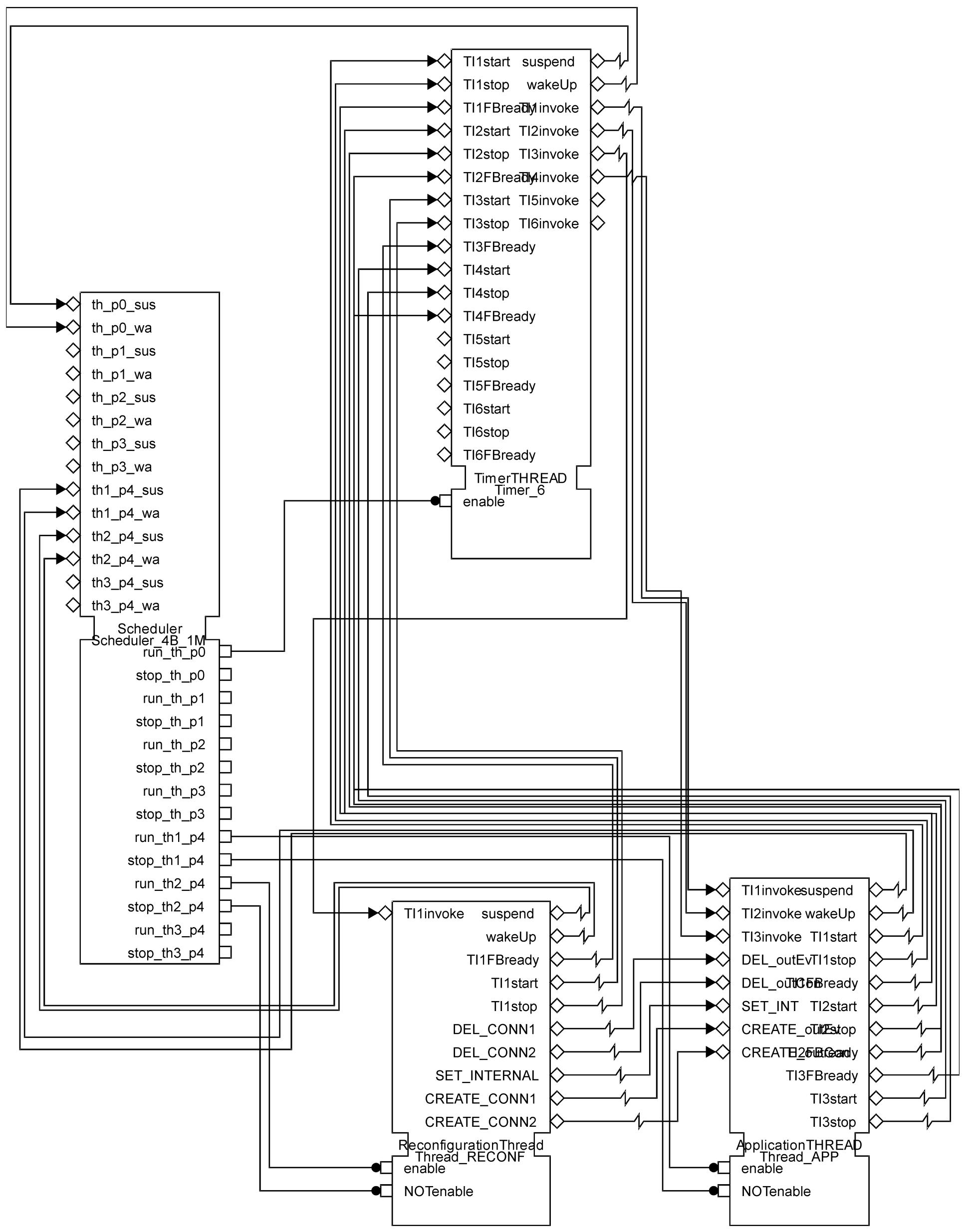


Figure 76: NCES model for practical example (Addition/Subtracion)

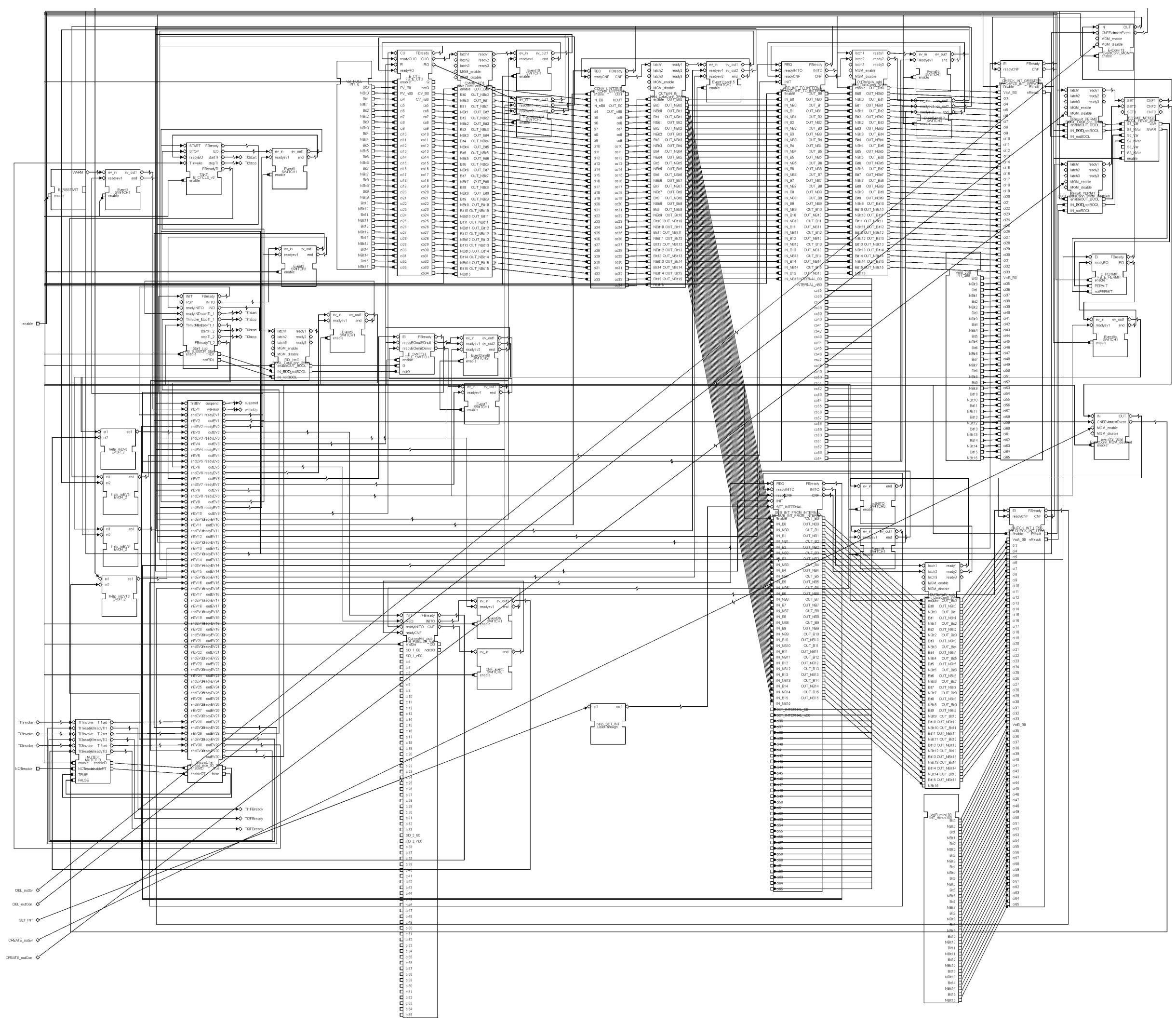


Figure 77: NCES module 'Thread_APP', the control application of Addition/Subtraction example at the beginning of the RECONF sequence

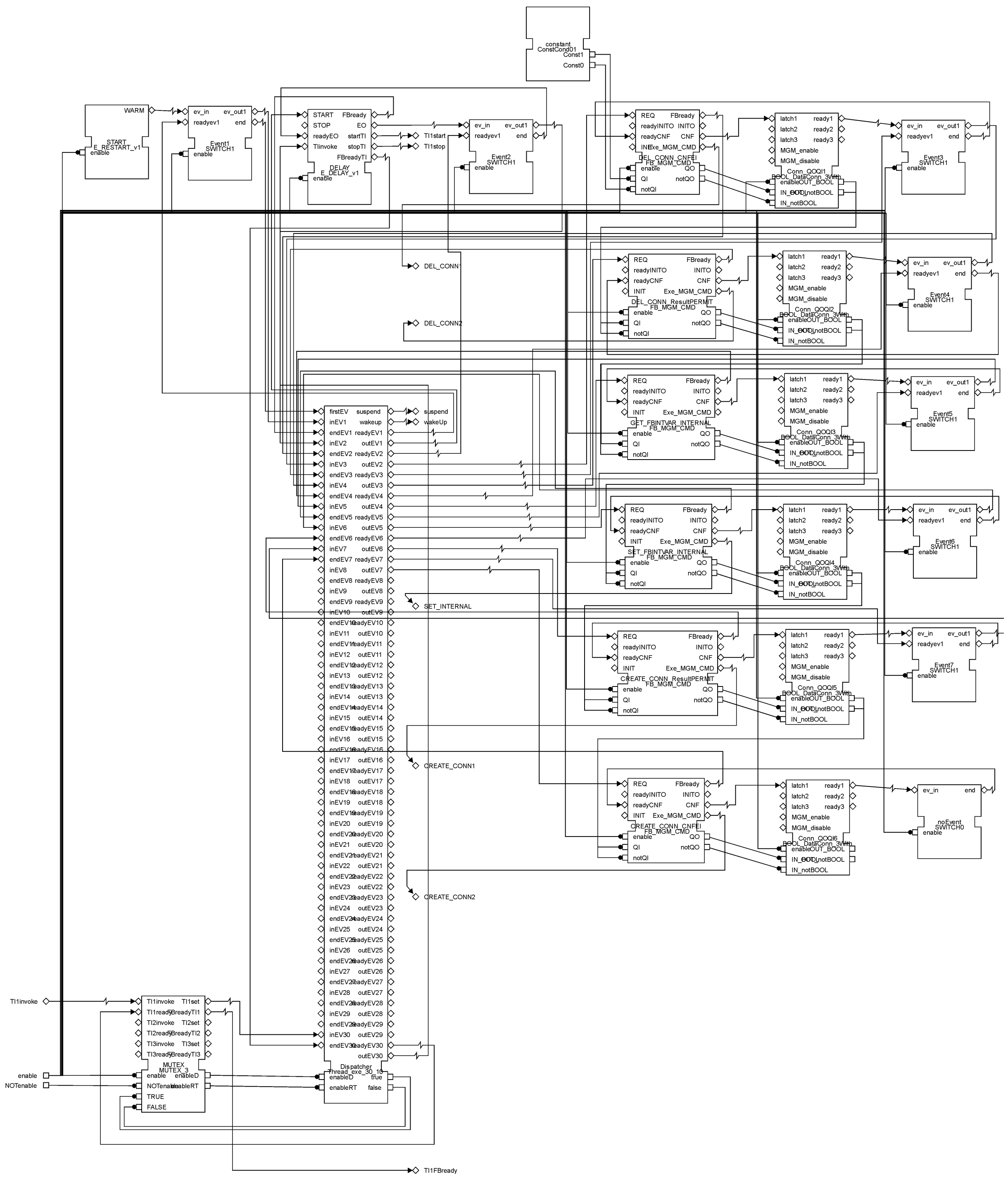


Figure 78: NCES module 'Thread_RECONF', the RECONF seqeunce of ECA within Addition/Subtraction example

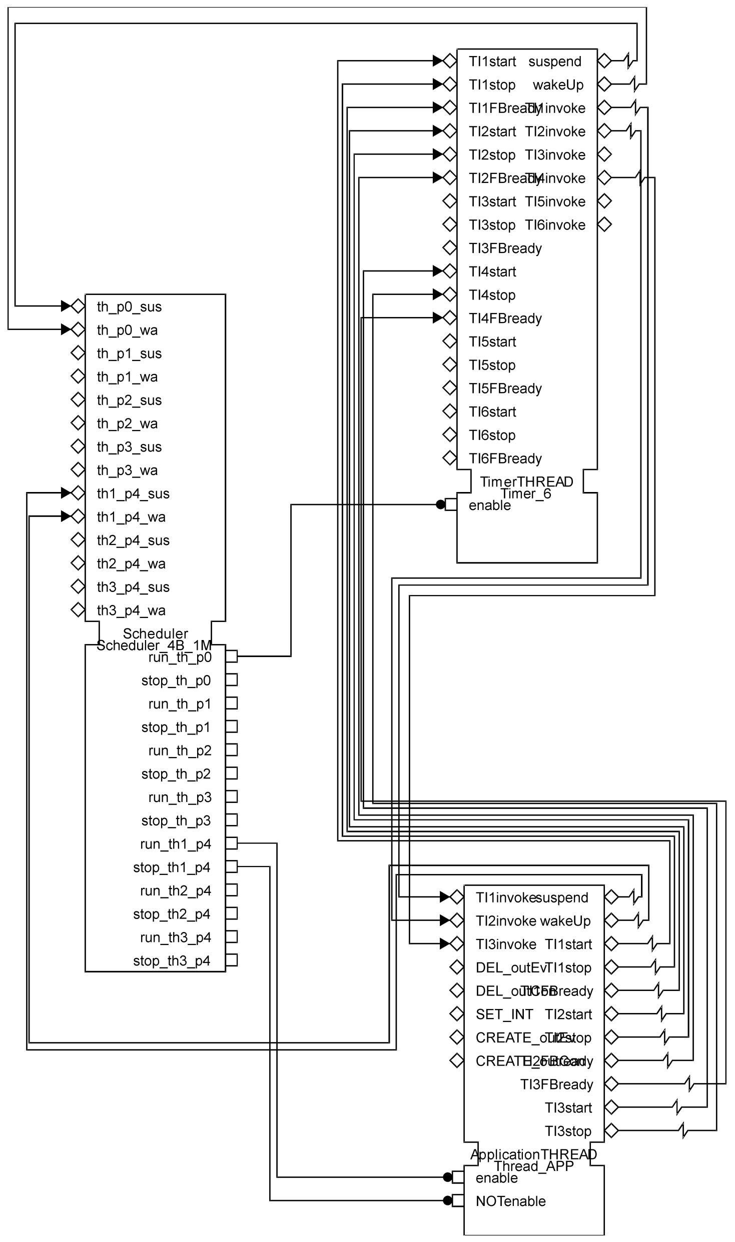


Figure 79: NCES model of Addition/Subtraction example without downtimeless system evolution

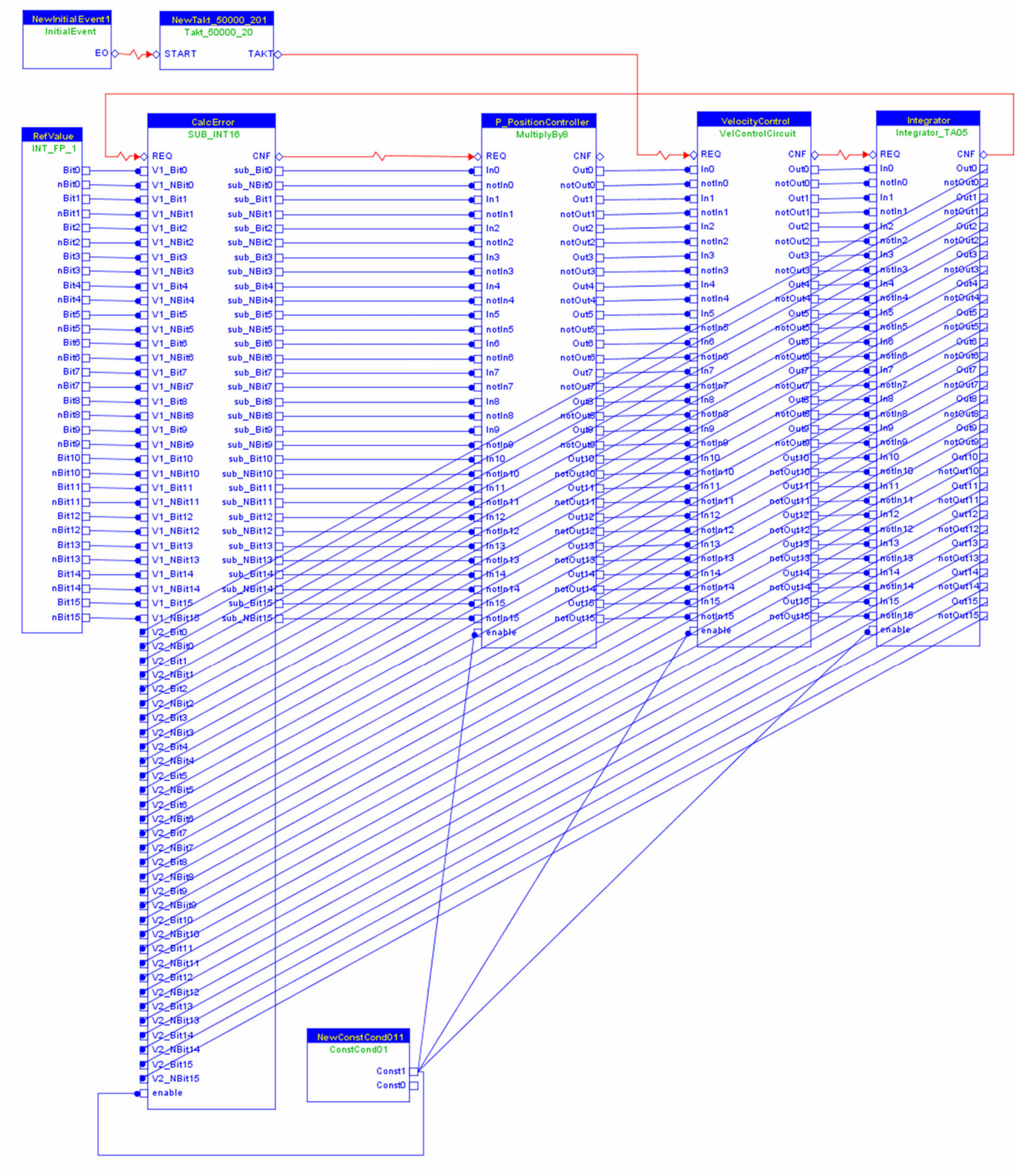


Figure 80: NCES model for position controller including plant model (velocity closed-loop control)

Used terms and abbreviations

New terms introduced in the thesis

Automation and Control System (ACS). ACS is used as an umbrella term for any kind of system[32] that is able to control some kind of process. This thesis uses this term for equipment for production automation (in contrast to product automation, see Favre-Bulle (2004). In common, such a system is programmable and flexible configurable to fit to the process under control. A typical kind of ACS is a PLC. An ACS may be any kind of feedforward or feedback system. Due to the intensive usage of sensor systems as input for the control logic, we commonly consider an ACS as a feedback system.

Automation extreme programming (AXP): Following the idea of extreme programming stated in Beck (2000) the application for ACS with regard to the possibilities of DSE leads to automation extreme programming. As changes can be applied during operation of the plant and the cost of change is limited although for late changes in the engineering cycle a continuous evolution of the plant will take place starting from a very simple first version.

Basic reconfiguration services. A set of basic services that allows all necessary modifications of control logic in order to conduct any system evolution. The basic reconfiguration services are based on the management commands from (IEC 61499-1, 2005) and can be categorized in five groups: structural services, library services, execution control services, state interaction services, and query services. More detailed information can be found in Zoitl (2007).

Composite Evolution Control Application (CECA). An ECA which includes several system evolution steps can be represented in a simplified manner by the use of EECFBs for each system evolution step. Such an ECA is called CECA.

Downtimeless system evolution (DSE). The term downtimeless system evolution describes the combination of software evolution and dynamic reconfiguration for automation and control systems. Its characteristics are that it faces the adaptation of system at run-time with at less disturbances as possible, it aims at overall system configurations, and it claims for a continuous evolution process for the overall life cycle of ACSs.

Dynamic reconfiguration. Dynamic reconfiguration is used as an umbrella term for methodologies and techniques for the change of software architectures at run-time. Wermelinger (1999) provides an overview on three different approaches to dynamic reconfigureation, an reference architecture for dynamic reconfiguration is presented in Walsh *et al.* (2007b) who define a categorization of change types as well as various integrity characteristics.

[32] The term system is widely used in literature as the interconnection of parts that exhibit as a whole one or more properties. If these properties are not obvious from the properties of the individual parts, a complex system is described. ACS are considered as complex systems in this thesis.

Evolution Control Application (ECA): An ECA describes the transition from a current system state to a new system state by the use of an IEC 61499 application. Herein special function blocks that provide access to the device management of the IEC 61499 device are used. As the ECA is free programmable, the DSE can be adapted to the special needs of an ACS: wide range of application, transition management, and the incomplete representation of ACS programming languages as software components. The use of ECAs in an IEC 61131-3 based system environment is described in Chapter 10.

Evolution Execution Control Function Block (EECFB). This special kind of FB includes an ECA and provides a separate interface for each of the three sequences within the ECA. Therefore, an ECA can be simply combined with other ECAs in order to model enhanced DSE scenarios again by means of IEC 61499 standard.

Evolution Region Of Interest (EROI). The EROI defines the region within an control application that will be changed during a system evolution step. As the ECA of a system evolution step is represented as EECFB, the EROI depicts the corresponding area within the control application that is related to the EECFB.

Evolution specification. Evolution specifications are a list of properties that specially aim at the execution of DSE. Next to the preservation of the properties of system without evolution (plant, process, and product specifications) these properties include active references to underlying services, state management within the control application, dependent operation of basic reconfiguration services, real-time constrained operation, and requirements of resources.

KAPPA calculus. The term KAPPA calculus is used as umbrella term and represents any calculation that is based on the current system state (respectively the KAPPA vector). This is a very broad field of calculations since many aspects of engineering support can be described as KAPPA calculus as for instance depicted in Sünder *et al.* (2007c). In terms of evaluation of DSE especially the evaluation of the properties of the evolution specification by the use of calculations based on the KAPPA vector are summarized as KAPPA calculus.

KAPPA vector. The KAPPA vector is the representation of the current system state. It is a structured list of parameters that can be devided according to their dependency in application dependent and/or device dependent elements. During normal operation the KAPPA vector is static, but during the execution of DSE the KAPPA vector is a highly versaticle quantity as it changes due to each execution of a basic reconfiguration service.

Physical reconfiguration. This term depicts dynamic reconfiguration of hardware configurations. During a system evolution step software as well as hardware may be changed. As there exist no automatisms for changes to the hardware configuration, the necessary tasks (addition or removal of hardware) need to be executed manually during the DSE.

Real-time Reconfiguration Runtime Environment (R^3E). The R^3E is the IEC 61499 runtime environment which provides the basis for the execution of DSE within this work. Its main principles are described in Zoitl (2007), although several minor changes and enhancements have been applied to it during the εCEDAC project.

Value-added chain for total evaluation: The provision of the exhaustive description for all elements within the architecture of a control device can be based on roles of the different companies involved in ACSs. Each company adds a special expertise to the ACS, therefore the necessary information for the evaluation of DSE regarding to this expertise, has to be provided by this company, too. As a result, the effort for providing DSE support is reduced and spread up between the different roles in ACS and the application of DSE will achieve high benefits related to the necessary effort.

General terms and abbreviations

Accompanying Measure on Advanced Real-Time Systems (ARTIST). The ARTIST project [1] was funded by the European Union in Frame Program 5. ARTIST gathered together 30 leading European research institutes in the area of embedded systems design. One major output was a roadmap on future directions in advanced real-time systems (Bouyssounouse and Sifakis, 2005).

Ambigrams. Ambigrams can be letters, words, or numbers that are ambiguous. Some ambigrams are mirror reversible, others can be read inverted, others have two meanings hidden in the word.

Automation and Control Institute (ACIN). ACIN is located in the faculty for electro technique and information technology at Vienna University of Technology, [2]

Automation object. Following the idea of Vyatkin *et al.* (2005, Section 3), "an automation object is an abstraction for a mechanical device associated with its embedded intelligence, i.e., software components of different functional domains. For example, the layout of a component is related to its appearance on its visualization screen (View). The View component can receive data about the dynamic state of the object either from actual process interface or from a simulation model. Control functions can refer directly to the process interface of the object or to the low-level control functions. The HMI component communicates with both process and controller."

Basic Function Block (BFB). A BFB type is defined in (IEC 61499-1, 2005, Section 3.8) as "function block type that cannot be decomposed into other function blocks and that utilizes an execution control chart (ECC) to control the execution of its algorithms".

Best Case Execution Time (BCET): Analogous to the definition of WCET given in (Kopetz, 1997, Section 4.5) the BCET of a task is a lower bound for the time between task activation and task termination.

CAN in Automation (CiA). CiA [6] is the international users' and manufactures' group that develops and supports CANopen and other CAN-based higher-layer protocols. Especially for CANopen a very broad range of specifications for different devices profiles have been developed.

Component framework. "A component framework is a dedicated and focused architecture, usually around a few key mechanisms, and a fixed set of policies for mechanisms at the component level. Component frameworks often implement protocols to connect participating components and enforce some of the policies set by the framework." (Szyperski, 2005, Section 20.3)

Component system architecture. "A component system architecture consists of a set of platform decisions, a set of component frameworks, and an interoperation design for the component frameworks. A platform is the substrate that allows for installation of components and component frameworks, such that these can be instanced and activated. (…) An interoperation design for component frameworks comprises the rules of interoperation among all the frameworks joined by the system architecture." (Szyperski, 2005, Section 20.3)

Composite Function Block (CFB). A CFB type is defined in (IEC 61499-1, 2005, Section 3.16) as "function block type whose algorithms and the control of their execution are expressed entirely in terms of interconnected component function blocks, events, and variables".

Computability. The theory of computability can be described by two questions (Hopcraft *et al.* , 2001, Section 1.1.3): „What can a computer do at all?" and „What can a computer do efficiently?". Herein abstract models such as finite automata and formal grammars are used in design and construction of software. Other concepts like Turing machines are used to understand what we can expect from a software program.

Computation Tree Logic (CTL). CTL is a temporal logic is a temporal logic that utilizes a model of time that is a tree-like structure. “The future is not determined; there are different paths in the future, any one of which might be the ‘actual’ path that is realized.” (Huth and Ryan, 2004, Section 3.4.1) There exist different dialects such as CTL*, CTL, ACTL, TCTL, or RTCTL in literature.

Computer Numerical Control (CNC) machines. The combination of a machine tool, an actuator system to force the axis of the machine tool, and an integrated numerical control by a computer is called CNC machine. These machines are specialized to operations on a workpiece, e.g. drilling, milling, turning, or grinding, and provide means for coordinated motion of the tool center point. (Favre-Bulle, 2004, Section 2.4.5)

Continuous Stochastic Logic (CSL). The logic CSL is a stochastic version and variant of the temporal logic CTL. It permits expessing steady-state probabilities. CSL properties are verified at the state-space level using model checking. A detailed definition of CSL is given for instance in D’Aprile *et al.* (2004, Section 2).

Control Area Network (CAN): The CAN network has been developed especially for the automotive sector in order to connect typically sensors, actuators, and control devices. Each node is able to send and receive messages, conflicts are resolved by the identification of each node which represents the header of a message.

Deductive verification. Deductive verification is a technique for validation of the functionality of a system and usually refers to the use of axioms and rules to prove the correctness of systems. It requires expert knowledge and is applied to highly sensitive systems such as security protocols.

Document Type Definition (DTD). “The XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.” [64], Section 2.8

Electronic Device Description Language (EDDL). EDDL has been standardized in (IEC 61804-3, 2006) and is a generic description language for ACS components. EDDL is used to create an Electronic Device Description, which is a textual file including the properties of an ACS components in terms of the EDDL, such as device parameters and their dependencies, device functions, graphical representations, interaction with control devices, and persistent data store.

Enterprise Resource Planning (ERP). A ERP system is a information system, that takes care on all business processes within an enterprise in trans-sectoral way. The information processes are at least partly automated. (Favre-Bulle, 2004, Section 3.5.3)

Event-Condition-Action (ECA). Dynamically reconfigurable software systems such as active databases or expert systems are often specified by ECA rules. Almeida *et al.* (2007) describes an approach to use ECA rules for modeling reconfigurable logic controllers. The elements of an ECA rule are the event, which is used to trigger evaluation of the condition of the rule, and in case of positive evaluation the action that is executed.

Embedded Configurable Operating System (eCos): eCos is an open source, royalty-free, real-time operating system intended for embedded applications. There exists a high variety of different hardware platforms which are already supported by eCos (see [9]).

Based on the highly configurable nature eCos can be adapted to the requirements of a given application very flexible. A detailed description of the concepts implemented by eCos is provided in Massa (2003).

Embedded systems. "We will refer to embedded systems as electronic programmable subsystems that are generally an integral part of a larger heterogeneous system. An embedded system acts within—and in many cases on—the physical environment. Embedded systems are, by nature, inherently real-time computer systems." (Bouyssounouse and Sifakis, 2005)

Evolution Control Environment for Distributed Automation Components (εCEDAC). The εCEDAC project aims at a fundamentally new application centered engineering method for efficient component based modeling of applications for controlled, fault-tolerant and safe system evolution in order to overcome the limitations of current embedded industrial automation and control engineering methods.

The εCEDAC project has been funded by the Austrian government within the FIT-IT embedded systems program under contract number FFG-809447/7126. The project lasted from May 2005 to July 2007. The project consortium consists of the companies Bachmann electronic [3], kirchner SOFT[33] [29], Loytec electronics [28], and Siemens VAI [49], as well as the research institutes Profactor [45] and ACIN [1]. More detailed information is available on the corresponding web page [3].

Execution Control Chart (ECC). The ECC is defined in (IEC 61499-1, 2005, Section 3.40) as "graphical or textual representation of the causal relationships among events at the event inputs and event outputs of a function block and the execution of the function block's algorithms, using execution control states, execution control transitions, and execution control actions".

Field bus. A common means for communication in ACS are field bus systems. They fulfill special requirements such as real-time characteristics, reliability, and rough ambient conditions. There exists a huge variety of field bus systems as describe by Favre-Bulle (2004, Section 2.7). In recent years especially the introduction of Ethernet networks for industrial communication tremendously increase the variety of available systems.

Field Device Configuration Markup Language (FDCML). FDCML (FDCML.org, 2002) has been developed by DRIVECOM User Group e.V. [24] and Interbus Club [6] as a general markup language for device description files based on XML. It is based on the (ISO 15745-1, 2003) basic elements and fulfills the basic requirements network independence, multi language support, and extensibility. Further information is available via [10].

Field Device Tool (FDT). The FDT concept defines the interface between device-specific software components provided by the device supplier and the engineering tool of the control system manufacturer. The focus of the FDT technology lies on engineering, commissioning, diagnostics and documentation of field bus-based ACSs. The engineering tool is able to interact actively with the field bus device. FDT has been specified by the FDT Group [11], where further information is available.

Framework for Distributed Automation and Control (4DIAC). The general aim of the 4DIAC initiative is to provide an open, IEC 61499 standard compliant basis, that gives the opportunity to establish an automation and control system based on the targets portability, configurability and interoperability. The open source project includes a runtime environment as well as an engineering tool. Further information is available via [12].

[33] Company kircher SOFT has changed its name to „logi.cals Austria" in 2008.

Function Block (FB). The term function block is used in several standards, for this work especially the definitions of IEC 61131-3 (2003) and IEC 61499-1 (2005) are of special interest.

Function Block Diagram (FBD). FBD is a graphical programming language defined in (IEC 61131-3, 2003). FBs or functions are interconnected via flow lines, the signal flow shall be from the output (right-hand) side to the input (left-hand) side. The execution of FBD is defined so that an element is evaluated only if its predecessor elements are evaluated. Additionally, implementation dependent rules can be applied.

Function Block Development Kit (FBDK). The FBDK [15] has been developed by James H. Christensen in parallel to the development of the IEC 61499 standard. It consists of two parts, an engineering tool as well as a runtime environment (FBRT) for the elements of IEC 61499 standard. It is the first IEC 61499 engineering tool and still enhanced by James H. Christensen.

Function Block Run Time (FBRT). The FBRT is the runtime environment within the FBDK engineering tool. It is implemented in Java and is characterized by its implementation of the event propagation by direct function call. The detailed execution semantics of the FBRT are described in Sünder *et al.* (2006a).

Function Block Execution Runtime (FUBER). This IEC 61499 runtime environment has been published as open source project [13] by Goran Čengić (Automation Research Group, Chalmers University of Technology). A description of FUBER can be found in Čengić *et al.* (2006).

Generalized Stochastic Petri Nets (GSPN). "Generalized Stochastic Petri Nets (…) have two different classes of transitions: immediate transitions and timed transitions. Once enabled, immediate transitions fire in zero time. Timed transitions fire after a random, exponentially distributed enabling time as in the case of Stochastic Petri Nets." (Bause and Kritzinger, 1996, Chapter 8)

Holonic Manufacturing Systems (HMS). A HMS is "a holarchy (a system of holons—autonomous and cooperative building blocks of a manufacturing system—which can cooperate to achieve a goal or objective) which integrates the entire range of manufacturing activities from order booking through design, production and marketing to realize the agile manufacturing enterprise" (Christensen, 1994). The HMS project [22] took place within the Intelligent Manufacturing Systems initiative [23] from 1991 to 2004.

Human Machine Interface (HMI). The HMI is the interface between a machine and a human being. Generally the system state of the machine is displayed in some kind, e.g. by some graphics, and there exists the possibility to influence the machine by the use of buttons or textual inputs.

Instruction List (IL). IL is a textual programming language defined in (IEC 61131-3, 2003). IL is composed of a sequence of instruction, whereas each instruction shall begin on a new line. A line may consist of an optional label, an operator, optional operands, and an optional comment. The operators are very simple such as load, Boolean operations as well as mathematical operations. Further, jump operators to labels and FB/function calls are possible.

International Electrotechnical Commission (IEC). International Organization [25] that provides international standards and conformity assessment for government, business, and society for all electrical, electronic, and related technologies.

Intellectual Property (IP). IP is a used in law as umbrella term for a bundle of exclusive rights concerning information, ideas, and so on. In the contrast of ACS, IP means the special knowledge of a vendor concerning his products. It is necessary to provide means for the protection of a vendor's IP to build an open, knowledge-based economy in industrial automation. (Vyatkin *et al.*, 2005)

Industrial Personal Computer (IPC). An IPC is in general a personal computer that fulfills special requirements according to is use as replacement for PLCs in ACSs. The additional features are robustness, use of standards, type of protection, operating system, and communication with other system components. (Favre-Bulle, 2004, Section 2.4.4)

Kripke structure. "A Kripke structure consists of a set of states, a set of transitions between states, and a function that labels each state with a set of properties that are true in this state. Paths in a Kripke structure model computations of the system. Although these models are very simple, they are sufficiently expressive to capture those aspects of temporal behavior that are most important for reasoning about reactive systems." (Clarke *et al.*, 1999, Chapter 2)

Ladder Diagram (LD). LD is a graphical programming language defined in (IEC 61131-3, 2003). The symbols are laid out in a similar manner to a rung of a relay ladder logic diagram. The LD network is located between the left and right power rail. The used symbols are contacts, coils as well as functions and FBs (by the use of the EN/ENO construct). The execution order is rung by rung from top to bottom. Within a rung, an element is evaluated only if its predecessor element has been already evaluated.

Linear-time Temporal Logic (LTL). "LTL is a temporal logic (…) that models time as a sequence of states, extending infinitely into the future. (…) In general, the future is not determined, so we consider several paths, representing different possible futures." (Huth and Ryan, 2004, Section 3.2) LTL consists "of formulas that have the form $\mathbf{A} f$ where f is a path formula in which the only state subformulas permitted are atomic propositions." (Clarke *et al.*, 1999, Section 3.2)

Middleware. "Category of software that is neither dedicated to the operation of a specific system (handled by the operating systems) nor to the functionality of specific applications. Middleware typically addresses cross-system concerns, such as communication, synchronization, and coordination, that are of importance to multiple applications." (Szypersky, 2005, Glossary)

Micro Holons for Next Generation Distributed Embedded Automation and Control Systems (µCrons). The aim of the µCrons project is to overcome limitations of state of the art PLC-based automation technology and provide an embedded system computing infrastructure (middleware) for µCrons that supports predictable dynamic reconfiguration of real-time application software that is distributed on µCrons. A µCron is the mechatronic assembly of mechanic, hydraulic, pneumatic, etc. parts with electric, computing, networking and software level components to an intelligent, reconfigurable, user-programmable device of fine granularity (similar to an automation object).

The µCrons project has been funded by the Austrian government within the FIT-IT embedded systems program under contract number FFG-808205/7126. The project lasted from May 2004 to November 2006. The project consortium consists of the companies Festo GmbH [19] and Fronius International AG [20], as well as the research institutes Profactor [45], ACIN [1], and University of Applied Science, Upper Austria [57]. More detailed information is available on the corresponding web page [36].

Model checking. Model checking is a technique for validation of the functionality of a system for finite state concurrent systems. Based on the model of a system, a model checking tool verifies whether the state space of the system fulfils a given specification or not. If the model violates some property of the specification, a counterexample as a path within the state space is given.

Mulitlevel Queue (MLQ). The MLQ scheduling policy is part of the eCos kernel component. It provides the use of several threads within the same priority level, with a maximum number of 32 priorities. All tasks within one priority level are held within a queue, and the execution time of a task may be limited by a timeslicing period. A detailed description of the MLQ scheduler is provided in Massa (2003, Section 5.1.3.1).

Net Condition/Event Systems (NCES). NCES are a modeling approach that is a combination of Condition/Event systems and Petri nets. There exist modules utilizing events and conditions as interfaces that can be interrelated to composite modules. Internals of a module are described Petri nets, whereas incoming conditions as well as events may be used for the firing condition of a transition. If a NCES has no inputs it is called Signal-Net System (SNS). A description of NCES and their use for verification of distributed control systems is given for instance in Vyatkin and Hanisch (2003b).

Object Management Group (OMG). The OMG is an international, open membership, not-for-profit computer industry consortium. The different task forces develop enterprise integration standards for a wide range of technologies, and an even wider range or industries. The most important specification developed by the OMG is UML. [38]

Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation (O^3neida). The O^3neida organization [28] operates as a network of networks focused on fostering distributed industrial automation based upon open standards. The basic idea of the O3neida organization is describe in Vyatkin *et al.* (2005).

OPC Unified Architecture (OPC UA). OPC UA has been defined by the OPC Foundation [41], which is a user driven organization that has released several specifications for the access of data from various devices in a unified manner. OPC stems from OLE for Process Control, which is derived from the technology Object Linking and Embedding used in Microsoft Windows operating systems. OPC UA defines a Client/Server architecture based on standardized web technologies (such as XML defined by W3C [63]).

Ordered Binary Decision Diagram (OBDD). OBDDs (Clarke *et al.*, 1999, Chapter 5) are a canonical form representation for Boolean formulas. They can be calculated from calculated from binary decision trees simply speaking by eliminating redundancy. A canonical form can be achieved by two restrictions: (i) a similar order of variables along each path and (ii) no isomorphic subtrees or redundant vertices. One challenge in the use of OBDDs is that the size of the OBDD depends critically on the variable order. Therefore techniques such as dynamic reordering have been developed.

PLCopen. PLCopen [44] was founded as an independent worldwide association for industrial suppliers and users resolving topics related to industrial control programming. PLCopen members have concentrated on technical specifications around IEC 61131-3, creating specifications and implementations in order to reduce cost in industrial engineering. Examples are the specifications for motion control and safety function blocks, and the XML schemes for an unified data format for control logic.

Predicate transformer. A predicate transformer is a function $\tau : Pred(S) \rightarrow Pred(S)$ with the attributes (i) τ is monotonic, (ii) τ is $\cup$-continuous and (iii) τ is $\cap$-continuous. *Pred(S)* is denoted as a lattice under the set inclusion ordering from the states *S* of a Kripke structure *M* (3). These properties are used for symbolic model checking to utilize fixpoint calculations. A detailed description is given for instance in Clarke *et al.* (1999, Chapter 6).

Priority inheritance protocol. A RTOS provides different means for synchronization of different tasks. The mutual exclusion method allows multiple tasks to share a resource serially. If a resource is occupied by a low priority task, and a high priority task wants to access the resource, different protocols can be used to resolve this situation. “The priority inheritance protocol allows a task that owns the mutual exclustion object to be raised to the priority level equal to the highest level of all threads waiting for the mutual exclusion object. The priority inheritance protocol is only used when a higher priority task is waiting for the mutual exclusion object." (Massa, 2003, Section 6.2.1)

Programmable Logic Controller (PLC). The term PLC is an umbrella term for control devices based on the IEC 61131 standard. Although the term is used more generally in literature, we will use this term for control devices that are programmable according to the concepts of (IEC 61131-3, 2003).

Program Organization Unit (POU). The standard IEC 61131-3 defines POU as an software element which may be either a program, function block or function. (IEC 61131-3, 2003)

Production planning and scheduling (PPS). A PPS system is used for computer based planning, ordering, and supervision of production flows. The main tasks are the coordination of the production program, quantities, schedules, and capacities. (Favre-Bulle, 2004, section 3.5.1)

Property Specification Pattern. "A property specification pattern is a generalized description of a commonly occurring requirement on the permissible state/event sequence in a finite-state model of a system. A property specification pattern describes the essential structure of some aspect of a system's behavior and provides expressions of this behavior in a range of common formalisms." (Dwyer *et al.*, 1998, Section 3, first paragraph)

Real-time computer system. "A real-time computer system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computation, but also on the physical instant at which these results are produced." (Kopetz, 1997, Section 1.1, first paragraph)

Real-time operating system (RTOS). A RTOS is an operating system, which provides support for concurrent programming via processes and/or threads, real-time scheduling services with predictable timing behavior, preemption, predictable synchronization mechanisms, mutual exclusion, and time management services. (Bouyssounouse and Sifakis, 2005, Section 21.1)

Reconfigurable Manufacturing System (RMS). "A reconfigurable manufacturing system is designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in the market or in regulatory requirements." (Koren *et al.*, 1999, Section 2)

Simulation. Simulation is a technique for validation of the functionality of a system by providing certain inputs and observing the outputs. The tests are performed on an abstraction or a model of the real product. Typical PLC engineering tools support the simulation on the engineering computer without the real PLC.

Sequential Function Chart (SFC). SFC is defined in (IEC 61131-3, 2003) as a means for partitioning POUs written in one of the languages defined by the standard, for the purpose of performing sequential control functions. It consists of steps and transitions interconnected by direct links. A step may have a set of actions associated, which are executed if the step is activated and according to the used action qualifier (for instantce time limited or rising edge). Each transition is associated with a transition condition. The initial situation of a SFC network is characterized by the initial state which is the active state. Evolution of the active states of steps shall take place along the directed links when caused by the clearing of one or more transitions.

Service Interface Function Block (SIFB). A SIFB is defined in (IEC 61499-1, 2005, Section 3.89) as "function block which provides one or more services to an application, based on a mapping of service primitives to the function block's event inputs, event outputs, data inputs, and data outputs".

Signal-Net Systems (SNSs). SNSs are autonomous systems of NCES modules, that are systems with no external inputs. For such systems analysis is possible. Starke and Roch (2002) describe analysis techniques for SNS in detail, such as dynamic properties, structural properties, Invariants and model checking. The appropriate model checker for SNS is called SESA and has been developed at Humboldt-University Berlin. It is available next to a graphical NCES tool in [61].

Software component: There exist several definitions for software components in literature. Within this work we use (Szyperski, 2002, Section 4.1.5): "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." Its main characteristics are that a software component is a unit of independent deployment, it may be used for composition by third-parties and it has no (externally) observable states.

Software maintenance. The term software maintenance is defined in (IEEE 14764, 2006, Section 3.10 "the totality of activities required to provide cost-effective support to a software system. Activities are performed during the pre-delivery stage as well as the post-delivery stage." The predecessor IEEE 1219 (1998) focuses more on the modification of a software product after delivery, whereas pre-delivery activities are necessary such as a the planning for post-delivery activities.

Software evolution. The term software evolution lacks for a concrete definition but became very famous due to the so called 'laws of software evolution' as for instance claimed by Lehmann and Ramil (2000) for component-based software. It is stated that a software product needs to be continuously adapted in order to kept satisfactory. There are different challenges of software evolution stated in Mens *et al.* (2005) including sufficient engineering support as well as runtime environments enabling adaptations at run-time.

Structured Text (ST). ST is a graphical programming language defined in (IEC 61131-3, 2003). ST consists of a list of statements, which may include assignments, FB/function calls, selections (IF construct), and iterations (FOR, WHILE, and REPEAT construct). Within the statements expressions are used, which are composed of operators and operands. The evaluation of expressions consists of applying the operators to the operands according to the operator precedence. In order equal precedence, evaluation shall be applied from left to right.

Subapplication. A subapplication type is defined in (IEC 61499-1, 2005, Section 3.95) as a "functional unit whose body consists of interconnected component function blocks or component subapplications". In the note to (IEC 61499-1, 2005, Section 3.94) it is stated that "a subapplication instance may be distributed among resources, i.e. its component function blocks or the content of its component subapplications may be assigned to different resources".

Symbolic Model Verifier (SMV). SMV was the first tool for checking finite-state systems by the use of symbolic model checking algorithms using OBDDs. It has been developed by Ken McMillan (1993) in his PhD thesis and provided important results to the state explosion problem. SMV uses its own input language, which is based on modules and their interaction. There are several tools available based on the original SMV, the most relevant one is the open source NuSMV [35].

Testing. Testing is a technique for validation of the functionality of a system by providing certain inputs and observing the outputs. The tests are performed on the real product.

Timed Net Condition/Event Systems (TNCES). TNCES is the extension of the strictly causual, untimed model of NCES to a timed model as introduces in Hanisch *et al.* (1997).

Tool Calling Interface (TCI). TCI specifies a concept that defines a calling interface for the device engineering tool and the automation system engineering tool. It provides parameterization and diagnostics over network boundaries, data storage, and reloading of set parameters when a device is replaced. TCI has been specified by the Profibus and Profinet International organization [48], where further information is available.

Total life cycle web-integrated control (TORERO). The TORERO project has focused on creating a total life cycle web-integrated control design architecture and methodology for distributed control systems in factory automation. It lasted from 1998 to 2002 and was funded by the European Union. Schwab *et al.* (2005) describe the TORERO approach in detail, further information is also available at [54].

Unified Modeling Language (UML). UML [56] is a visual language for the specification, visualization, construction and documentation of models for software systems. It has been specified by the OMG and represents the de facto standard used by software engineers. It includes for instance use case diagrams, class diagrams, and state charts.

Verification Environment for Distributed Applications (VEDA). The tool VEDA has been developed by Valeriy Vyatkin and provides a framework for the verification of distributed applications according to IEC 61499. The model of the IEC 61499 application is generated automatically, the validation is supported by visualization of the process along selected trajectories within the reachability graph. Details are given for instance in Vyatkin and Hanisch (2001a), actually VEDA is no more supported and developed. Related projects can be found in [14].

Visual Verifier (ViVe). ViVe is a verification tool for TNCES. The model checking is done via SESA and a simple built-in model checking algorithm. The models, reachability graph, and sequence diagrams of paths within the reachability graph are available in a graphical representation. ViVe is part of the visual framework for verification of function blocks [61] provided by Valeriy Vyaktin.

World Wide Web Consortium (W3C). The W3C [63] has been created as industry consortium dedicated to building consensus around web technologies in 1994. It develops interoperable technologies (specifications, guidelines, software, and tools) to lead the web to its full potential. XML is one of the most important standards developed by W3C.

Worst Case Active Task Set (WCATS): The WCATS is defined in (Zoitl, 2007) as “the task set that needs the most execution resources of all possible task sets”. It has to be considered for the proof of schedulability of real-time constrained control applications based on event chains.

Worst Case Execution Time (WCET): “The WCET of a task is an upper bound for the time between task activation and task termination. It must be valid for all possible input data and execution scenarios of the task, and should be a tight bound.” (Kopetz, 1997, Section 4.5)

XML Interface for Robots and Peripherals (XIRP). The XIRP specification (VDMA 66430-1, 2006) provides an XML based description for the interaction between robots and processor based peripherals (e.g. a vision system). The communication protocol defines the machine-to-machine exchange of commands. XIRP is an german initiative and has been specified by VDMA [59].

eXtended Markup Language (XML). XML [64] is a text format that consists of markup codes and raw data. By the use of the markup codes the raw data is structured and semantic information is added. XML has been developed by the *World Wide Web Consortium* (W3C) and is widely used in computer systems and for data exchange between computer systems. The structure of a XML file can be defined through a DTD file or a XML schema.

XML schema. The purpose of a XML schema is to define a class of XML documents. It can be viewed as a collection (vocabulary) of type definitions and element declarations whose names belong to a particular namespace. Different namespaces can be managed within an XML document. [65]

eXtreme Programming (XP). XP has been developed by Kent Beck as a summary of well known practices from computer science such as pair programming, testing, simplicity of design, short iterations, refactoring and so on. The design process becomes highly adaptive as changes are applied as soon as they have been identified, starting with a simple first version of the program. A detailed description of XP is given in Beck (2000).

Bibliography

In order to provide a better overview within the bibliography, we will use the following classification of references: “**” means that this reference is directly used for the new contribution of this work, and “*” means that this reference is interesting in the context of this work. Any other references regard to state-of-the-art.

* Alcaraz-Mejía, M., López-Mellado, E. (2006) Petri Net Model Reconfiguration of Discrete Manufacturing Systems. In: Proceedings of 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM’06), vol. 1, Saint-Etienne (France), May 2006, pp. 547-552

* Almeida, E. E., Luntz, J. E., Tilbury, D. M. (2007) Event-Condition-Action Systems For Reconfigurable Logic Control. IEEE Transactions on Automation Science and Engineering, vol. 4, nb. 2, ISSN 1545-5955, pp. 167-181

Alur, R. Courcoubetis, C., Dill, D. (1990) Model-checking for real-time systems. In: Proceedings of 5th Annual IEEE Symposium on Logic in Computer Science (LICS’90), Philadelphia (PA, USA), June 1990, pp. 414-425

Alur, R., Dill, D. (1992) A Theory of Timed Automata. Lecture Notes in Computer Science (LNCS 600), Real-Time: Theory in Practice, Springer Verlag Berlin Heidelberg, pp. 45-73

* Angelov, C., Sierszecki, K., Marian, N. (2005) Design Models for Reuseable and Reconfigureable State Machines. Lecture Notes in Computer Science (LNCS 3824): Proceedings of International Conference on Embedded and Ubiquitous Computing (EUC’05), Nagasaki (Japan), December 2005, pp. 152-163

Angelov, C., Ke, X., Sierszecki, K. (2006) A Component-Based Framework for Distributed Control Systems. In: Proceedings of 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06), Dubrovnik (Croatia), August/September 2006, pp. 20-27

* Appavoo, J., Hui, C., Soules, C. A. N., Wisniewski, R. W., Da Silva, D. M., Krieger, O., Auslander, M. A., Edelsohn, D. J., Gamsa, B., Ganger, G. R., McKenney, P., Ostrowski, M., Rosenburg, B., Stumm, M., Xenidis, J. (2003) Enabling automatic behavior in systems software with hot swapping. IBM Systems Journal, vol. 42, nb. 1, pp. 60-76, ISSN 0018-8670

** Baier, T., Fritsche, J., Keintzel, G., Loy, D., Schranz, R., Steininger, H., Strasser, T., Sünder, C. (2007) Future scenarios for application of downtimeless reconfiguration in industrial practice. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN’07), Vienna ‘(Austria), July 2007, pp. 1129-1134

* Bani Younis, M., Frey, G. (2003) Formalization of Existing PLC Programs: A Survey. In: Proceedings of IEEE/IMACS Multiconference on Computational Engineering in Systems Applications (CESA’03), Lille (France), July 2003, 6 pp.

Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., Stursberg, O. (2004) Verification of PLC Programs Given as Sequential Function Charts. Lecture Notes on Computer Science (LNCS 3147), Integration of Software Specification Techniques for Applications in Engineering, Springer Verlag Berlin Heidelberg, pp. 517-540, ISBN 3-540-23135-8

Bause, F., Kritzinger, P. S. (1996) Stochastic Petri Nets: An Introduction to the Theory. Advanced Studies in Computer Science, Verlag Vieweg, Wiesbaden (Germany), ISBN 3-528-05535-9

** Beck, K. (2000) Extreme Programming Explained: Embrace Change. Addison-Wesley, USA, ISBN 201-61641-6

Bender, K., Großmann, D., Danzer, B. (2007) FDT+EDD+OPC UA=FDD UA: Die Gleichung for eine einheitliche Gerätebeschreibung? atp—Automatisierungstechnische Praxis, Oldenbourg-Industrieverlag, Munich (Germany), vol. 49, nb. 2, pp. 48-54, ISSN 0178-2320

Bengtsson, J., Wang, Y. (2004) Timed Automata: Semantics, Algorithms and Tools. Lecture Notes in Computer Science (LNCS 3098), In: Proceedings of 4th Advanced Course on Petri Nets (ACPN'03), Eichstätt (Germany), September 2003, pp. 87-124

** Bennet, K. H., Rajlich, V. T. (2000) Software Maintenance and Evolution: a Roadmap. In: Finkelstein (Ed.) The Future of Software Engineering. ACM Press, ISBN 1-58113-253-0

* Bitsch, F. (2001) Saftey Patterns—The Key to Formal Specification of Safety Requirements. Lecture Notes in Computer Science (LNCS 2187), In: Proceedings of 20th International Conference on Computer Safety, Reliability and Security (SAFECOM'01), Budapest (Hungary), September 2001, pp. 176-189

* Bitsch, F. (2003) A Way For Applicable Formal Specification Of Safety Requirements By Tool-Support. In: Proceedings of Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03), Budapest (Hungary), May 2003, 11 pp.

* Blunden, B. (2003) Memory Management: Algorithms and Implementation in C/C++. Wordware Publishing, Inc., Plano (TX, USA), ISBN 1-55622-347-1

* Bonfe, M., Fantuzzi, C. (2003) Design and Verification of Mechatronic Object-Oriented Models for Industrial Control Systems. In: Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '03), Lisbon (Portugal), September 2003, pp. 253-260

* Bouyssounouse, B., Sifakis, J. (Eds.) (2005) Embedded Systems Design: The ARTIST Roadmap for Research and Development. Lecture Notes in Computer Science (LNCS 3436), Springer Verlag, Berlin/Heidelberg, ISBN 978-3-540-25107-1

** Brennen, R. W., Zhang, X., Xu, Y., Norrie, D. H. (2002a) A reconfigurable concurrent function block model and its implementation in real-time Java. Integrated Computer-Aided Engineering, IOS Press, vol. 9, nb. 3. ISSN 1069-2509, pp.263-279

* Brennan, R. W., Fletcher, M., Norrie, D. H. (2002b) An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems. IEEE Transactions on Robotics and Automation, vol. 18, nb. 2, ISSN 1070-9932, pp.444-451

** Brunnenkreef, J. (2006) Design and implementation of a demonstrator / testbed for IPMCS with respect to physical and logical reconfiguration of mechatronic parts. Technical report on external training at Profactor GmbH, University of Twente, Enschede (The Netherlands), December 2006

CiA DSP 311 (2007) CANopen device description—XML schema definition. Draft Standard Proposal, Version 1.0, CAN in Automation (CiA) e.V., 8th March 2007

Christensen, J. H. (1994) Holonic Manufacturing Systems: Initial architecture and standards directions. In: Proceedings of 1st European Conference on Holonic Manufacturing Systems, Hannover (Germany), December 1994, 20 pp.

* Čengić, G., Ljungkrantz, O., Akesson, K. (2006) Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime. In: Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 1269-1276

* Chaki, S., Clarke, E., Groce, A., Jha, S., Veith, H. (2004) Modular Verification of Software Components in C. IEEE Transactions on Software Engineering, vol. 30, nb. 6, pp. 388-402, ISSN 0098-5589

Clarke, E. M., Emerson, I. A. (1981) Design and synthesis of synchronization skeletons using branching time temporal logic. Lecture Notes in Computer Science (LNCS 131), In: Proceedings of Workshop on Logics of Programs, May 1981, Yorktown Heights (NY, USA), pp. 52-71

* Clarke, E. M., Grumberg, O., Peled, D. A. (1999) Model Checking. The MIT Press, Cambridge (MA, USA), ISBN 0-262-03270-8

* Cofer, D. D., Rangarajan, M. (2003) Event-triggered Environments for Verification of Real-time Systems. In: Proceedings of 35th Winter Simulation Conference (WSC'03), December 2003, New Orleans (LA, USA), pp. 915-922

* Corbett, J. C. (1996) Timing Analysis of Ada Tasking Programs. IEEE Transactions on Software Engineering, vol. 22, nb. 7, July 1996, ISSN 0098-5589, pp. 461-483

D'Aprile, D., Donatelli, S., Sproston, J. (2004) CSL Model Checking for the GreatSPN Tool. Lecture Notes in Computer Science (LNCS 3280), In: Proceedings of 19th International Symposium on Computer and Information Sciences (ISCIS'04), Kemer-Antalya (Turkey), October 2004, pp. 543-553

** Demmelmayr, F., Zafari, S. (2007) Evolution von Steuerungssystemen. (in German), Final documentation, university course "Leittechnik Vertiefung", Vienna University of Technology, 8th February 2007

* Douglas, D. P. (1999) Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison-Wesly, Boston (USA), ISBN 0-201-49937-5

* Dubinin, V., Vyatkin, V., Hanisch, H.-M. (2006) Modelling and Verification of IEC 61499 Applications using Prolog. In: Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 774-781

** Dwyer, M. B., Avrunin, G. S., Corbett, J. C. (1998) Property Specification Patterns for Finite-State Verification. In: Proceedings of the 2nd Workshop on Formal Methods in Software Practice (FMSP'98), Clearwater Beach (FL, USA), March 1998, pp. 7-15

** Dwyer, M. B., Avrunin, G. S., Corbett, J. C. (1999) Patterns in Property Specifications for Finite-State Verification. In: Proceedings of the 21st International Conference on Software Engineering (ICSE'99), Los Angeles (CA, USA), May 1999, pp. 411-420

* European Commission (2006) MANUFUTURE: Strategic Research Agenda, Assuring the future of Manufacturing in Europe. Report of the High-Level Group, September 2006, ISBN 92-79-01026-3

Favre-Bulle, B. (2004) Automatisierung komplexer Industrieprozesse: Systeme, Verfahren und Informationsmanagement. Springer Verlag, ISBN 3-211-21194-2

* Favre-Bulle, B. (2005) Zukunft der Forschung in den Produktionswissenschaften. Technical Report, Verein zur Förderung der Modernisierung der Produktionstechnologien in Österreich (VPTÖ), Steyr–Gleink

** FDCML.org (2002) FDCML 2.0 Specification, Version 1.0, 8th November 2002

** Ferhatbegovic, T. (2007) Echtzeitbetriebssysteme—Vergleich. (in German), Final documentation, university course "Leittechnik Vertiefung", Vienna University of Technology, March 2007

Ferrarini, L., Veber, C., Lorentz, K. (2003) A case study for modelling and design of distributed automation systems. In: Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM'03), Kobe (Japan), July 2003, pp. 1043-1048

Frey, G., Litz, L. (2000) Formal methods in PLC programming. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC'00), vol. 4, Nashville (TN, USA), October 2000, pp. 2431-2436

* Frey, G., Hussain, T. (2006) Modeling Techniques for Distributed Control Systems based on the IEC 61499 Standard—Current Approaches and Open Problems. In: Proceedings of 8th International Workshop on Discrete Event Systems, Ann Arbor (MI, USA), July 2006, pp. 176-181

** Gosetti, I. (2007) Formale Beschreibung einer IEC 61499-Laufzeitumgebung unter Berücksichtigung von Echtzeitanforderungen und dem unterlagerten Betriebssystem. (in German), Master thesis, Vienna University of Technology, Automation and Control Institute, September 2007

Greifender, J., Frey, G. (2007) Probabilistic Timed Automata for Modeling Networked Automation Systems. In: Proceedings of 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07), Cachan (France), June 2007, pp. 143-148

** Guler, M., Clements, S., Wills, L. M., Heck, B. S., Vachtsevanos, G. J. (2003) Transition Management for Reconfigurable Hybrid Control Systems. IEEE Control Systems Magazine, vol. 23, nb. 1, February 2003, pp. 36-49, ISSN 0272-1708

** Hall, K. H., Staron, R. J., Zoitl, A. (2007) Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 823-828

* Hanisch, H.-M., Thieme, J., Lüder, A., Wienhold, O. (1997) Modeling of PLC Behavior by Means of Timed Net Condition/Event Systems. In: Proceedings of 6th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'97), Los Angeles (CA, USA), September 1997, pp. 391-396

Hanisch, H.-M., Thieme, J., Lautenbach, K., Simon, C. (2000) A Modular Modeling Approach for Hybrid Systems Based on C/E-systems and Extended Timestamp Nets. In: Proceedings of 4th International Conference on Automation of Mixed Processes (ADPM'00), Dortmund (Germany), September 2000, pp. 363-368

** Hanisch, H.-M. (2004) Closed-Loop Modeling and Related Problems of Embedded Control Systems in Engineering. Lecture Notes on Computer Science (LNCS 3052): Proceedings of 11th Int. Workshop on Abstract State Machines (ASM'04), Lutherstadt Wittenberg (Germany), May 2004, Springer Verlag Berlin Heidelberg, pp. 6-19

* Hanisch, H.-M., Lobov, A., Martinez Lastra, J. L., Tuokko, R., Vyatkin, V. (2006) Formal validation of intelligent-automated production systems: towards industrial applications. International Journal on Manufacturing Technology and Management

(IJMTM), vol. 8, nb. 1/2/3, Inderscience Enterprise Ltd., pp. 75-106, ISSN 1368-2148

** Hanni, C. (2007) Baukasten für die Evolution von Regelkreisen im laufenden Betrieb auf Basis des εCEDAC Ansatzes (in German), Master thesis, Vienna University of Technology, Automation and Control Institute, September 2007

Hopcraft, J. E., Motwani, R., Ullman, J. D. (2001) Introduction to automata theory, languages, and computation. Second edition, Addison-Wesely (USA), ISBN 0-201-44124-1

** Hummer, O., Sünder, C., Zoitl, A., Strasser, T., Rooker, M. N., Ebenhofer, G. (2006) Towards Zero-downtime Evolution of Distributed Control Applications via Evolution Control based on IEC 61499. In: Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 1285-1292

** Hummer, O., Sünder, C., Strasser, T., Rooker, M. N., Kerbleder, G. (2007) Downtimeless System Evolution: Current State and Future Trends. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1123-1128

Huth, M., Ryan, M. (2004) Logic in Computer Science: Modelling and Reasoning about Systems. Second Edition, Cambridge University Press, New York, ISBN 0-521-54310-X

* Iacooca Institute (1991), 21st Century Manufacturing Enterprise Strategy: An Industry-Led View. Lehigh University Press, ISBN 0-9624866-3-9

IEC 61131-1 (2003) Programmable Controllers—Part 1: General information. International Standard, International Electrotechnical Commission, Second Edition, Geneva

* IEC 61131-3 (2003) Programmable Controllers—Part 3: Programming languages. International Standard, International Electrotechnical Commission, Second Edition, Geneva, ISBN 2-8318-6653-7

IEC 61131-5 (2000) Programmable Controllers—Communications. International Standard, International Electrotechnical Commission, First edition, Geneva

** IEC 61499-1 (2005) Function blocks—Part 1: Architecture. International Standard, International Electrotechnical Commission, First Edition, Geneva

** IEC 61499-2 (2005) Function blocks—Part 2: Software tools requirements. International Standard, International Electrotechnical Commission, First edition, Geneva

** IEC 61499-3 (2004) Function blocks for industrial-process measurement and control systems—Part 3: Tutorial information. Technical report, International Electrotechnical Commission, First edition, Geneva

** IEC 61499-4 (2005) Function blocks—Part 4: Rules for compliance profiles. International Standard, International Electrotechnical Commission, First edition

IEC 61804-3 (2006) Function blocks (FB) for process control—Part 3: Electronic Device Description Language (EDDL). International Standard, International Electrotechnical Commission, First edition, Geneva

IEC 62390 (2005) Common automation device—Profile guideline. Technical report, International Electrotechnical Commission, First edition, Geneva

* IEEE 1219 (1998) IEEE standard for software maintenance, IEEE—The Institute of Electrical and Electronics Engineergs, Inc., USA, ISBN 0-7381-0336-5

IEEE 1451.3 (2003) IEEE Standard for a Smart Transducer Interface for Sensors and Actuators—Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems. IEEE Instrumentation and Measurement

Society, IEEE—The Institute of Electrical and Electronics Engineers, Inc., USA, ISBN 0-7381-3823-1

** ISO 15745-1 (2003) Industrial automation systems and integration—Open systems application integration framework—Part 1: Generic reference description. International Organisation for Standardization, Geneva

ISO/IEC 14764-IEEE Std 14764 (2006) Software Engineering—Software Life Cycle Processes—Maintenance. International Standard, ISO/IEEE, USA, ISBN 0-1381-4961-6 SS95534

John, K.-H., Tiegelkamp, M. (2000) SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen. Third, revised edition, Springer Verlag, ISBN 3-540-66445-9

John, K.-H., Tiegelkamp, M. (1995) IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools. Springer Verlag, ISBN 3-540-67752-6

Kaiser, J., Piontek, H. (2005) Self-describing devices in COSMIC. In: Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05), vol. 1, Catania (Italy), September 2005, 4 pp.

* Kalita, D., Khargonekar, P. P. (2002) Formal Verification for Analysis and Design of Logic Controllers for Reconfigurable Manufacturing Systems. IEEE Transactions on Robotics and Automation, vol. 18, nb. 4, pp. 463-474

* Khalgui, M., Rebeuf, X., Simonot-Lion, F. (2004) A behavior model for IEC 61499 function blocks. In: Proceedings of 3rd Workshop on Modelling of Objects, Components, and Agents, Aarhus (Denmark), October 2004, pp. 71-88

* Khalgui, M., Rebeuf, X., Simonot-Lion, F. (2006) Component based deployment of industrial control systems: a hybrid scheduling approach. In: Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 1293-1300

** Kopetz, H. (1997) Real-Time Systems: Design Principles of Distributed Embedded Applications. Kluwer Academic Publisher, Boston, ISBN 0-7923-9894-7

* Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., Van Brussel, H. (1999) Reconfigurable Manufacturing Systems. Annals of the CIRP, vol. 48, nb. 2, pp. 527-540

** Kovácsházy, T., Péceli, G., Simon, G. (2001) Transients in Reconfigureable Signal Processing Channels. IEEE Transactions on Instrumentation and Measurement, vol. 50, nb. 4, pp. 936-940

* Krákora, J., Waszniowski, L., Píša. P., Hanzálek, Z. (2004) Timed Automata Approach to Real Time Distributed System Verification. In: Proceedings of 5th IEEE International Workshop on Factory Communication Systems (WFCS'04), Vienna (Austria), September 2004, pp. 407-410

** Kramer, J., Magee, J. (1985) Dynamic Configuration for Distributed Systems. IEEE Transactions on Software Engineering, vol. SE-11, nb. 4, April 1985, pp. 424-436

* Kramer, J., Magee, J. (1990) The Evolving Philosophers Problem: Dynamic Change Configuration. IEEE Transactions on Software Engineering, vol. 11, nb. 11, November 1990, pp. 1293-1306

Kropik, M. (2005) Distributed Automation in Automotive Manufacturing: Current Status and Strategies. 18th International Cooperation Symposium Industry—Research, September 13th, 2005, Vienna (Austria)

Lee, S.-M., Harrison, R., West, A. A. (2004) A Component-based Distributed Control System for Assembly Automation. In: Proceedings of 2nd IEEE International Conference on Industrial Informatics (INDIN'04), Berlin (Germany), June 2004, pp. 33-38

** Lehmann, M. M., Ramil, J. F. (2000) Software evolution in the age of component-based software engineering. IEE Proceedings Software, vol. 147, nb. 6, ISSN 1462-5970

Lewis, R. W. (1998) Programming industrial control systems using IEC 1131-3, Revised edition. IEE-The Institution of Electrical Engineers, London (United Kingdom), ISBN 0-85296-950-3

Lewis, R. W. (2001) Modelling control systems using IEC 61499: Applying function blocks to distributed systems. IEE-The Institution of Electrical Engineers, London (United Kingdom), ISBN 0-85296-796-9

* Li, J., Dai, X., Meng, Z. (2005) Dynamic Reconfiguration of Petri Net Logic Controllers Based on Modified Net Rewriting Systems. In: Proceedings of IEEE International Conference on Mechatronics and Automation, vol. 2, Niagara Falls (Canada), July 2005, pp. 592-567

* Lobov, A., Popescu, C., Martinez Lastra, J. L. (2006a) A Framework for Validation of Reconfigurable Manufacturing Systems. In: Proceedings of 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'06), vol. 1, Saint-Etienne (France), May 2006, pp. 529-534

Lobov, A., Popescu, C., Martinez Lastra, J. L. (2006b) An Algorithm for Siemens STL representation in TNCES. In: Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 641-647

** logi.cals Austria (2008) logi.CAD Help. Documention for logi.CAD 5.0,

Lopez Orozco, O. J., Martinez Lastra, J. L. (2007) Agent-Based Control Model for Reconfigurable Manufacturing Systems. In: Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Patras (Greece), September 2007, pp. 1233-1238

Lüder, A., Schwab, C., Tangermann, M., Peschke, J. (2005) Formal models for the verification of IEC 61499 function block based control applications. In: Proceedings of 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05), Catania (Italy), September 2005, pp. 105-112

** Mandl, R., Zhang, W. (2008) Charakterisierung von Embedded Software. (in German), Final documentation, university course "Leittechnik Vertiefung", Vienna University of Technology, May 2008

** Massa, A. J. (2003) Embedded Software Development with eCosTM. Prentice Hall-Professional Technical Reference, Pearson Education, Inc., Upper Saddle River (NJ, USA), ISBN 0-13-035473-2

McMillan, K. L. (1993) Symbolic Model Checking: An Approach to the State Explosion Problem. Kluwer Academics Publishers, ISBN 0-7923-9380-5

** Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., Jazayeri, M. (2005) Challenges in Software Evolution. In: Proceedings of 8th IEEE Int. Workshop on Principles of Software Evolution (IWPSE'05), September 2005, Lisbon (Portugal), pp. 13-22

* Meolic, R., Kapus, T., Brezočnik, Z. (2001) CTL and ACTL patterns. In: Proceedings of International Conference on Trends in Communications (EUROCON'2001), vol. 2. IEEE, Bratislava (Slovak Republic), July 2001, pp. 540-543

Niemann, K.-H. (2007) Stand der Integration intelligenter Systemkomponenten in der Prozessleittechnik. (in German), atp—Automatisierungstechnische Praxis, Oldenbourg-Industrieverlag, Munich (Germany), vol. 49, nb. 5, pp. 15-20, ISSN 0178-2320

* Pang, C., Vyatkin, V. (2007) Towards Formal Verification of IEC61499: modeling of Data and Algorithms in NCES. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 879-884

** Pang, C., Vyatkin, V. (2008) Automatic Model Generation of IEC 61499 Function Blocks Using Net Condition/Event Systems. In: Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN'08), Daejeon (Korea), July 2008, pp. 1133-1138

Park, E., Tilbury, D. M., Khargonekar, P. P. (2001) A Modeling and Analysis Methodology for Modular Logic Controllers of Machining Systems Using Petri Net Formalism. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews, vol. 31, nb. 2, pp. 168-188

Peschke, J., Lüder, A. (2005) The JAKOBI architecture—a distributed dynamic execution environment in Java. In: Proceedings of 3rd IEEE International Conference on Industrial (INDIN'05), Perth (Australia), August 2005, pp. 25-31

Peterson, J. L. (1981) Petri Net Theory and the Modeling of Systems. Prentice-Hall, Inc., Englewood Cliffs (NJ, USA), ISBN 0-13-661983-5

Petig, M. (2000) The way to distributed PLCs. Dedicated Systems Magazine, vol. 1, nb.2, April/May/June 2000, pp. 30-32

Petri, C. A. (1962) Kommunikation mit Automaten. (in german), Dissertation, Institut für Instrumentelle Mathematik der Universität Bonn.

** Phytec Messtechnik GmbH (2003) phyCORE-AT91M55800A, Hardware Manual, Edition February 2003

Priese, L., Wimmel, H. (2003) Theoretische Informatik: Petri-Netze. (in german), Springer Verlag, Berlin (Germany), ISBN 3-540-44289-8

Queille, J. P., Sifakis, J. (1981) Specification and verification of concurrent systems in CESAR. Lecture Notes in Computer Science (LNCS 137), In: Proceedings of 5th International Symposium on Programming, April 1981, Turin (Italy), pp. 337-351

* Rasche, A, Polze, A. (2005) Dynamic Reconfiguration of Component-based Real-time Software. In: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'05), Sedona (AZ, USA), February 2005, pp. 347-354

** Rausch, M., Hanisch, H.-M. (1995) Net Condition/Event Systems with Multiple Condition Outputs. In: Proceedings of INRA/IEEE Symposium on Emerging Technologies and Factory Automation, vol. 1, Magdeburg (Germany), October 1995, pp. 592-600

Recalde, L., Silva, M., Ezpeleta, J., Teruel, E. (2003) Petri Nets and Manufacturing Systems: An Examples-Driven Tour. Lecture Notes in Computer Science (LNCS 3098), Lectures on Concurrency and Petri Nets, Advances in Petri Nets, Tutorial volume of 4th Advanced Course on Petri Nets (ACPN'03), Eichstadt (Germany), September 2003, Springer Verlag Berlin Heidelberg, pp. 742-788

** Rooker, M. N., Sünder, C., Strasser, T., Zoitl, A., Hummer, O., Ebenhofer, G. (2007) Zero Downtime Reconfiguration of Distributed Automation Systems: The εCEDAC Approach. Lecture Notes in Computer Science (LNCS 4659), In: Proceedings of 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'07), Regensburg (Germany), September 2007, Springer Verlag Berlin Heidelberg, pp. 326-337

Schnakenbourg, C., Faure, J.-M., Lesage, J.-J. (2002) Towards IEC 61499 Function Blocks Diagrams Verification. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC'02), vol. 3, Yasmine Hammamet (Tunisia), October 2002, 6 pp.

Schwab, C., Tangermann, M., Ferrarini, L. (2005) Web based Methodology for Engineering and Maintenance of Distributed Control Systems: The TORERO Approach. In: Proceedings of 3rd IEEE International Conference on Industrial Informatics (INDIN'05), Perth (Australia), August 2005, pp. 32-37

* Setchi, R. M., Lagos, N. (2004) Reconfigurability and Reconfigurable Manufacturing Systems—State-of-the-art Review. In: Proceedings of the 2nd IEEE International Conference on Industrial Informatics (INDIN'04), Berlin (Germany), June 2004, pp. 529-535

* Simon, G., Kovácsházy, T., Péceli, G. (2000a) Transient Management in Reconfigurable Systems. Lecture Notes in Computer Science (LNCS 1936), In: Proceedings of First International Workshop on Self-Adaptive Software (IWSAS'00), Oxford (UK), April 2000, Springer Verlag Berlin Heidelberg, pp. 90-98

* Simon, G., Kovácsházy, T., Péceli, G. (2000b) Transients in Reconfigurable Control Loops. In: Proceedings of the 17th IEEE Instrumentation and Measurement Technology Conference (IMTC'00), Baltimore (MD, USA), May 2000, pp. 1333-1337

* Simon, G., Kovácsházy, T., Péceli, G. (2001) Transient Reduction in Control Loops in Case of Joint Plant-Controller Reconfiguration. In: Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference (IMTC'01), Budapest (Hungary), May 2001, pp. 1172-1176

Sproston, J. (2004) Model Checking for Probabilistic Times Systems. Lecture Notes on Computer Science (LNCS 2925), Validation of Stochastic Systems: A Guide to Current Research, Springer Verlag Berlin Heidelberg, pp. 189-229, ISBN 978-3-540-22265-1

* Sreenivas, R. S., Krogh, B. H. (1991) On condition/event systems with discrete state realizations. Discrete Event Dynamic Systems, vol. 1, nb. 2, September 1991, Springer Netherlands, pp. 209-236, ISSN 0924-6703

* Stanica, M.-P. (2005) Behavioral Modeling of IEC 61499 Control Applications. PhD thesis, Universite de Rennes, Institut D'Electronique et de Telecommunications de Rennes, Rennes (France)

** Starke, P. H., Roch, S. (2002) Analysing Signal-Net Systems. Technical report, Humboldt-Universität zu Berlin, Institut für Informatik, September 2002

* Steffen, T. (2005) Control Reconfiguration of Dynamical Systems. Lecture Notes in Control and Information Sciences (LNCIS 320), Springer Verlag Berlin Heidelberg, ISBN 3-540-25730-6

* Stewart, D. B., Volpe, R. A., Khosla, P. K. (1997) Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects. IEEE Transactions on Software Engineering, vol. 23, nb. 12, ISSN 0098-5589, pp. 759-776

** Strasser, T., Zoitl, A., Auinger, F., Sünder, C. (2005) Towards Engineering Methods for Reconfiguration of Distributed Real-Time Control Systems Based on the Reference Model of IEC 61499. Lecture Notes in Computer Science (LNCS 3593), In: Proceedings of 2nd International Conference on Industrial Applications of Holonic and

Multi-Agent Systems (HoloMAS'05), August 2005, Copenhagen (Denmark), Springer Verlag Berlin Heidelberg, pp. 165-175

** Strasser, T., Sünder, C., Rooker, M. N., Hummer, O., Zoitl, A., Müller, I. (2007) Enhanced IEC 61499 System Model for Evolution of Control Applications in Distributed Industrial-Process Measurement and Control Systems. In: Proceedings of the European Control Conference (ECC'07), Kos (Greece), July 2007, pp. 1356-1363

Sünder, C., Zoitl, A., Christensen, J.H., Vyatkin, V., Brennan, R.W., Valentini, A., Ferrarini, L., Strasser, T., Martinez-Lastra, J.L., Auinger, F. (2006a) Interoperability and Useablity of IEC 61499. In: Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN'06), Singapore (Singapore), July 2006, pp. 31-37

Sünder, C., Zoitl, A., Dutzler, C. (2006b) Functional structure-based modelling of automation systems. International Journal of Manufacturing Research (IJMR), vol. 1, no. 4, pp. 405-420

** Sünder, C., Zoitl, A., Favre-Bulle, B., Strasser, T., Steininger, H., Thomas, S. (2006c) Towards Reconfiguration Applications as basis for Control System Evolution in Zero-downtime Automation Systems. In: Innovative Production Machines and Systems, Proceedings of 2nd I*PROMS Virtual International Conference, July 2006, pp. 523-528

** Sünder, C., Rofner, H., Vyatkin, V., Favre-Bulle, B. (2007a) Formal description of an IEC 61499 runtime environment with real-time constraints. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 853-859

* Sünder, C., Zoitl, A., Christensen, J. H., Colla, M., Strasser, T. (2007b) Execution Models for the IEC 61499 elements Composite Function Block and Subapplication. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1169-1175

** Sünder, C., Hummer, O., Favre-Bulle, B. (2007c) Enhanced Engineering of Downtimeless System Evolution by use of Hardware Capability Descriptions within the εCEDAC Approach. In: Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Patras (Greece), September 2007, pp. 764-767

* Sünder, C., Zoitl, A., Rofner, H., Strasser, T., Brunnenkreef, J. (2007d) Benchmarking of IEC 61499 runtime environments. In: Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Patras (Greece), September 2007, pp. 474-481

** Sünder, C., Gosetti, I., Vyatkin, V., Favre-Bulle, B. (2008) Comprehensive Formal Description of IEC 61499 Control Devices. In: Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN'08), Daejeon (Korea), July 2008, pp. 1166-1172

* Szyperski, C. (2002) Component Software: Beyond Object-Oriented Programming. Second Edition, Addison-Wesley, ACM Press New York, ISBN 0-201-74572-0

* Tešanović, A., Nadjm-Tehrani, S., Hansson, J. (2005) Modular Verification of Reconfigurable Components. Lecture Notes in Computer Science (LNCS 3778), Component-Based Software Development for Embedded Systems, Springer Verlag Berlin Heidelberg, pp. 59-81, ISBN 978-3-540-30644-3

Thramboulidis, K., Prayati, A. (2001) Field Device Specification for the Development of Function Block Oriented Engineering Support Systems. In: Proceedings of 8th IEEE

International Conference on Emerging Technologies and Factor Automation (ETFA'01), Nice (France), September 2001, vol. 1, pp. 581-587

Thramboulidis, K. (2005) Model-Integrated Mechatronics—Toward a New Paradigm in the Development of Manufacturing Systems. IEEE Transactions on Industrial Informatics, vol. 1, no. 1, pp. 54-61

* Thramboulidis, K., Zoupas, A. (2005) Real-Time Java in Control and Automation: A Model Driven Development Approach. In: Proceedings of 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05), Catania (Italy), September 2005

* Thramboulidis, K., Papakonstantinou, N. (2006) An IEC61499 Execution Environment for an aJile-based Field Device. In: Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 989-992

VDMA 66430-1 (2006) XML-basiertes Kommunikationsprotokoll für Industrieroboter und prozessorgesteuerte Peripheriegeräte (XIRP), Teil 1: Allgemeine Vereinbarungen. VDMA Einheitsblatt, Verband Deutscher Maschinen- und Anlagenbauer e.V. (VDMA), Germany

* Vyatkin, V., Hanisch, H.-M. (1999) A Modeling approach for Verification of IEC1499 function blocks using Net Condition/Event Systems. In: Proceedings of the 7th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'99), Barcelona (Spain), September 1999, pp. 261-270

* Vyatkin, V., Hanisch, H.-M. (2001a) Formal Modeling and Verification in the Software Engineering Framework of IEC61499: a Way to Self-verifying Systems. In: Proceedings of 8th IEEE International Conference on Emerging Technologies and Factor Automation (ETFA'01), Nice (France), September 2001, vol. 2, pp. 113-118

* Vyatkin, V., Hanisch, H.-M. (2001b) Application of Visual Specifications for Verification of Distributed Controllers. In: Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC'01), vol. 1, Tucson (AZ, USA), October 2001, pp. 646-651

* Vyatkin, V. (2003) Intelligent Mechatronic Components: Control System Engineering using an Open Distributed Architecture. In: Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '03), Lisbon (Portugal), September 2003, pp. 277-284

Vyatkin, V., Hanisch, H.-M., Pfeiffer, T. (2003a) Object-oriented Molular Place/Transition Formalism for Systematic Modeling and Validation of Industrial Automation Systems. In: Proceedings of 1st IEEE Int. Conference on Industrial Informatics (INDIN'03), Calgary (Canada), August 2003, pp. 224-232

Vyatkin, V., Hanisch, H.-M. (2003b) Verification of distributed control systems in intelligent manufacturing. Journal of Intelligent Manufacturing, vol. 14, nb. 1, Springer Netherlands, pp. 123-136, ISSN 0956-5515

** Vyatkin, V., Christensen, J. H., Martinez Lastra, J. L. (2005) OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation. IEEE Transactions on Industrial Informatics, vol. 1, no. 1, pp. 4-17

* Vyatkin, V. (2006) Execution Semantic of Function Blocks based on the Model of Net Condition/Event Systems. In: Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN'06), Singapore (Singapore), July 2006, pp. 874-879

* Vyatkin, V. (2007a) IEC 61499 Function Blocks for Embedded and Distributed Control System Design. ISA—The Instrumentation, Systems, and Automation Society, ISBN 978-0-9792343-0-9

** Vyatkin, V. (2007b) Modelling and Verification of Discrete Control Systems with Net Condition/Event Systems and Visual Verification Framework, Working draft, Version 13/11/2007 (contained in [61])

* Vyatkin, V., Bouzon, G. (2008) Using Visual Specification in Verification of Industrial Automation Controllers. EURASIP Journal of Embedded Systems, Hindawi Publishing Coorporation, 9 pp., ISSN 1687-3955

* Walsh, J. D., Bordeleau, F., Selic, B. (2007a) A Constrained Executable Model of Dynamic System Reconfiguration. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07), Waikoloa (HI USA), January 2007, pp. 257c-257c

** Walsh, J. D., Bordeleau, F., Selic, B. (2007b) Domain analysis of dynamic system reconfiguration. Software and Systems Modeling, vol. 6, no. 4, Springer Verlag, pp. 355-380, ISSN 1619-1366

* Waszniowski, L., Hanzalek, Z. (2003) Analysis of Real Time Operating System Based Applications. Lecture Notes in Computer Science (LNCS 2791), In: Proceedings of First International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03), Marseille (France), September 2003, Springer Verlag Berlin Heidelberg, pp. 219-233

Wermelinger, M. A. (1999) Specification of Software Architecture Reconfiguration. PhD thesis, Universidade Nova de Lisboa, Departamento de Informatica, Lisbon (Portugal)

* Whisnant, K., Kalbarczyk, Z. T., Iyer, R. K. (2003) A system model for dynamically reconfigurable software. IBM Systems Journal, vol. 42, no. 1, pp. 45-59, ISSN 0018-8670

* Wind River Inc. (2007) Wind River Workbench 3.0, Technical Note, 17 pp.

Wollschlaeger, M., Wenzel, P. (2005) Common Model and Infrastructure for Application of XML within the Automation Domain. In: Proceedings of 3rd IEEE International Conference on Industrial Informatics (INDIN'05), Perth (Australia), August 2005, pp. 246-251

Wurmus, H., Wagner, B. (2000) IEC 61499 konforme Beschreibung verteilter Steuerungen mit Petri-Netzen. (in German), In: Proceedings of Fachtagung Verteilte Automation, Magdeburg (Germany), 8 pp.

* Yu, L., Shoja, G. C., Müller, H. A., Srinivasan. A. (2002) A Framework for Live Software Upgrade. In: Proceedings of 13th IEEE International Symposium on Software Reliability Engineering (ISSRE'02), Annapolis (MD USA)November 2002, pp. 149-158

Zeichen, G., Fürst, K. (2000) Automatisierte Industrieprozesse (in German). Springer Verlag, Vienna (Austria), ISBN 3-211-83560-1

Zhang, W., Diedrich, C., Halang, W. A. (2004) Module and Integration Verification for Function Block-based Safety-Related System Development. In: Proceedings of the 2nd IEEE International Conference on Industrial Informatics (INDIN'04), Berlin (Germany), June 2004, pp. 210-215

Zhang, W., Halang, W. A., Diedrich, C. (2005) Specification and Verification of Applications Based on Function Blocks. Lecture Notes in Computer Science (LNCS 3778), Component-Based Software Development for Embedded Systems, Springer Verlag Berlin Heidelberg, pp. 8-34, ISBN 978-3-540-30644-3

** Zoitl, A., Sünder, C., Terzic, I. (2006) Dynamic Reconfiguration of Distributed Control Applications with Reconfiguration Services based on IEC 61499. In: Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06), Prague (Czech Republic), June 2006, pp. 109-114

** Zoitl, A. (2007) Basic Real-Time Reconfiguration Services for Zero Down-Time Automation Systems. PhD thesis, Vienna University of Technology, Automation and Control Institute, Vienna (Austria)

Webiography

[1] ARTIST FP5, Online available: `http://www.artist-embedded.org/artist/ARTIST-FP5.html`, January 2008

[2] Automation and Control Institute. Vienna University of Technology, Online available: `http://acin.tuwien.ac.at`, November 2007

[3] Bachmann electronic. Online available: `http://www.bachmann.at`, November 2007

[4] BANDERA, SAnToS laboratory, Online available: `http://bandera.projects.cis.ksu.edu/`, March 2008

[5] BLAST: Berkeley Lazy Abstraction Software Verification Tool, Online available: `http://mtc.epfl.ch/software-tools/blast/`, March 2008

[6] CAN in Automation (CiA): Controller Area Network (CAN). Online available: `http://www.can-cia.org/`, February 2008

[7] DRIVECOM User Group e.V. Online available: `http://www.drivecom.org/`, Febrary 2008

[8] The εCEDAC project. Profactor Produktionsforschungs GmbH. Online available: `http://www.ecedac.org`, November 2007

[9] Ecos Homepage. Online available: `http://ecos.sourceware.org/ecos/`, May 2008

[10] FDCML.org. Online available: `http://www.fdcml.org/`, February 2008

[11] FDT Group, Online available: `http:/www.fdtgroup.org/en/home-en.html`, February 2008

[12] 4DIAC, Framework for Distributed Industrial Automation and Control. Profactor Produktionsforschungs GmbH, Online available: `http://www.fordiac.org`, March 2008

[13] Fuber, Function Block Execution Runtime. Online available: `http://sourceforge.net/projects/fuber`, January 2008

[14] Function Blocks – IEC 61499 Standard, Engineering Distributed Embedded Automation Systems with the New Generation Component Architecture. Valeriy Vyatkin, Block Design. Online available: `http://www.fb61499.com`, January 2008

[15] James H. Christensen. Holobloc Inc. Function Block Development Kit. Online available: `http://www.holobloc.com/doc/fbdk/index.htm`, January 2008

[16] James H. Christensen, Holobloc Inc. Online available: *http://www.holobloc.com*, November 2007

[17] James H. Christensen, IEC 61499 Compliance Profile for Feasibility Demonstration. Online available: *http://www.holobloc.com/doc/ita/index.htm*, November 2007

[18] ETMCC: Erlangen Twente Markov Chain Checker, Online available: *http://www7.informatik.uni-erlangen.de/etmcc/*, March 2008

[19] FESTO Gesellschaft m.b.H. Online available: *http://www.festo.at*, March 2008

[20] Fronius International AG. Online available: *http://www.fronius.com*, March 2008

[21] GreatSPN 2.0, Dipartimento di Informatica, Universita die Torino, Online available: *http://www.di.unito.it/~greatspn/index.html*, March 2008

[22] HMS Holonic Manufacturing Systems. Online available: *http://hms.ifw.uni-hannover.de*, January 2008

[23] Intelligent Manufacturing Systems. Online available: *http://www.ims.org*, January 2008

[24] INTERBUS Club. Online available: *http://www.interbusclub.com/*, February 2008

[25] International Electrotechnical Commission. Online available: *http://www.iec.ch*, December 2007

[26] ICS Triplex, ISaGRAF v.5, IEC 61131 and IEC 61499 Software. Online available: *http://www.isagraf.com*, January 2008

[27] Java PathFinder, Online available: *http://javapathfinder.sourceforge.net/*, March 2008

[28] Loytec electronics. Online available: *http://www.loytec.com*, November 2007

[29] kichner SOFT GmbH, also logi.cals Austria. Online available: *http://www.kirchnersoft.com*, *http://www.logicals.com*, May 2008

[30] KRONOS, Online available: *http://www-verimag.imag.fr/TEMPORISE/kronos/*, March 2008

[31] MAGIC: Modular Analysis of programs In C, Online available: *http://www.cs.cmu.edu/~chaki/magic/*, March 2008

[32] Martin-Luther-Universität Halle-Wittenberg, Fachbereich Mathematik und Informatik, Institut für Informatik, Lehrstuhl Automatisierungstechnik. Online available: *http://www.aut.informatik.uni-halle.de/*, July 2008

[33] The MOVIDA Tools Framework, Online available: *http://www.pe.tut.fi/movida3/tools*, March 2008

[34] MSDN, .NET Framework Developer Center. Online available: *http://msdn2.microsoft.com/en-us/netframework/default.aspx*, February 2008

[35] NuSMV: a new symbolic model checker, Online available: *http://nusmv.irst.itc.it/*, March 2008

[36] The μCrons project. Profactor Produktionsforschungs GmbH. Online available: *http://www.microns.org*, March 2008

[37] Scott Kim puzzlemaster. Online available: *http://www.scottkim.com*, January 2008

[38] Object Management Group. Online available: *http://www.omg.org/*, January 2008

[39] O^{3}neida—An Open, Object-Oriented Knowledge Economy for Intelligent Industrial Automation. Online available: *http://www.oooneida.info*, November 2007

[40] O^{3}neida Workgroup on Execution Models of IEC 61499 Function Block Applications. Online available: *http://www.oooneida.org/standards_development_Compliance_Profile.html*, February 2008

[41] OPC Foundation—Dedicated to interoperability in automation. Online available: *http://www.opcfoundation.org/*, February 2007

[42] Petri Nets World, University of Hamburg. Online available: *http://www.informatik.uni-hamburg.de/TGI/PetriNets/*. March 2008

[43] Phytec Messtechnik GmbH, Online available: *http://www.phytec.de*. June 2008

[44] PLCopen—for efficiency in automation. Online available: *http://www.plcopen.org*, January 2008

[45] PLCopen Newsletter December 2007. Online available: *http://www.plcopen.org/pages/promotion/publications/downloads/newsletter/ezine_dec_07.htm*, December 2007

[46] PRISM Model Checker, Online available: *http://www.prismmodelchecker.org/*, March 2008

[47] Profactor Produktionsforschungs GmbH. Online available: *http://www.profactor.at*, November 2007

[48] PROFIBUS & PROFINET International (PI), Online available: *www.profibus.com/pi/*, February 2008

[49] Siemens VAI. Online available: *http://www.siemens.com*, November 2007

[50] SLAM, Online available: *http://research.microsoft.com/slam/*, March 2008

[51] SPIN: On-the-fly, LTL model checking with SPIN, Online available: *http://spinroot.com/spin/whatispin.html*, March 2008

[52] The SMV System, Model Checking @CMU, Carnegie Mellon, Online available: *http://www.cs.cmu.edu/~modelcheck/smv.html*, March 2008

[53] Spec Patterns, SAnToS laboratory. Online available: *http://patterns.projects.cis.ksu.edu/*, March 2008

[54] TORERO, Total life cycle web-integrated control. Online available: *http://www.uni-magdeburg.de/iaf/cvs/torero/*, February 2008

[55] TSMV: TCTL Symbolic Model Checking of Simply-Timed Systems, Online available: *http://www.lsv.ens-cachan.fr/~markey/TSMV/*, March 2008

[56] UML Resource Page. Online available: `http://www.omg.org/uml/`, January 2008

[57] University of Applied Science, Upper Austria. Online available: `http://www.fh-ooe.at`, March 2008

[58] UPPAAL, Online available; `http://www.uppaal.com/`, March 2008

[59] VDMA - Verband Deutscher Maschinen- und Anlagenbauer e.V., Online available: `http://www.vdma.org`, February 2008

[60] VeriSoft, Bell Laboratories, Lucent Technologies, Online available: `http://cm.bell-labs.com/who/god/verisoft/`, March 2008

[61] Visual Framework for Verification of Function Blocks. Valeriy Vyatkin, Block Design. Online available: `http://www.fb61499.com/valid.html`, March 2008

[62] Wind River VxWorks. Online available: `http://www.windriver.com/products/vxworks/`, May 2008

[63] World Wide Web Consortium. Online available: `http://w3c.org`, January 2008

[64] Extensible Markup Language (XML) 1.0 (Fourth Edition). Online available: `http://www.w3c.org/TR/2006/REC-xml-20060816/`, January 2008

[65] W3C XML Schema. Online available: `http://www.w3.org/XML/Schema/`, February 2008

Index of Figures

Index of Tables

Curriculum Vitae

Personal data

Name:	Christoph Sünder
Date of birth:	February 2nd 1979
Address:	Korneuburgerstraße 2
	2003 Wiesen, Austria

Current Occupation (since August 2008)

Thales Rail Signalling Solutions GesmbH
Safety Manager

Education

July 2004 – June 2008	Research assistant at the Automation and Control Institute	
	Vienna University of Technology	
	Research fields:	Distributed Control
		Reconfiguration at run-time
		Verification of control programs
		Motion Control
October 1999 – June 2004	Vienna University of Technology	
	Master of Electrical Engineering	
	Major: Automation and Control Engineering	
	(graduation with distinction)	
September 1993 – June 1998	Technical High School for Electrical Engineering	
	TGM, 1200 Vienna	
	(graduation with distinction)	
1990 – 1998	Elementary and Middle School	

Publications

Sünder, C., Vyaktin V. (2008) Functional and temporal formal modelling of embedded controllers for intelligent mechatronic systems. Accepted for International Journal for Mechatronics and Manufacturing Systems, Inderscience Publishers, ISSN: 1753-1039

Sünder, C., Wenger, M., Hanni, C., Gosetti, I., Steininger, H., Fritsche, J. (2008) Transformation of existing IEC 61131-3 automation projects into control logic according to IEC 61499. Accepted for 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08), Hamburg (Germany), September 2008

Strasser, T., Rooker, M. N., Ebenhofer, G., Zoitl, A., ***Sünder, C.***, Valentini, A., Martel, A. (2008) Structuring of Large Scale Distributed Control Programs with IEC 61499 Subapplications and a Hierarchical Plant Structure Model. Accepted for 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08), Hamburg (Germany), September 2008

Strasser, T., Rooker, M. N., Ebenhofer, G., Hegny, I., Wenger, M., ***Sünder, C.***, Valentini, A., Martel, A. (2008) Multi-Domain Model-Driven Design of Industrial Automation and Control Systems. Accepted for 13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08), Hamburg (Germany), September 2008

Brennan, R., Vrba, P., Tichy, P., Zoitl, A., ***Sünder, C.***, Strasser, T., Marik, V. (2008) Developments in dynamic and intelligent reconfiguration of industrial automation. Computers in Industry, Elsevier, ISSN: 0166-3615 vol. 59, 15 pp.

Fritsche, J., Steininger, H., ***Sünder, C.***, Zoitl, A. (2008) Kein Entweder-Oder! (in German). Computer&Automation, Control Guide S2, WEKA Fachmedien GmbH, ISSN: 1615-8512, pp. 16-21

Sünder, C., Gosetti, I., Vyatkin, V., Favre-Bulle, B. (2008) Comprehensive Formal Description of IEC 61499 Control Devices. In: Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN'08), Daejeon (Korea), July 2008, pp. 1166-1172

Sünder, C., Zoitl, A., Christensen, J. H., Steininger, H., Fritsche, J. (2008) Considering IEC 61131-3 and IEC 61499 in the context of Component Frameworks. In: Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN'08), Daejeon (Korea), July 2008, pp. 277-282

Strasser, T., Rooker, M. N., Ebenhofer, G., Zoitl, A., ***Sünder, C.***, Valentini, A., Martel, A. (2008) Framework for Distributed Industrial Automation and Control (4DIAC). In: Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN'08), Daejeon (Korea), July 2008, pp. 283-288

Strasser, T., ***Sünder, C.***, Valentini, A. (2008) Model-Driven Embedded Systems Design Environment for the Industrial Automation Sector. In: Proceedings of the 6th IEEE International Conference on Industrial Informatics (INDIN'08), Daejeon (Korea), July 2008, pp. 1120-1125

Sünder, C., Zoitl, A., Steininger, H., Fritsche, J. (2008) Symbiose von IEC 61131-3 und IEC 61499: Integration von scheinbar sehr unterschiedlichen Welten (in German). In: Proceedings of 10. Fachtagung Entwurf komplexer Automatisierungssysteme (EKA'08), Magdeburg (Germany), April 2008, pp. 185-196

Sünder, C., Zoitl, A., Hummer, O., Strasser, T., Ebenhofer, G., Rooker, M. N., Kerbleder, G. (2008) 4DIAC—Framework for Distributed Industrial Automation Control. In: Proceedings of 10. Fachtagung Entwurf komplexer Automatisierungssysteme (EKA'08), Magdeburg (Germany), April 2008, pp. 163-174

Sünder, C., Fritsche, J., Steininger, H., Strasser, T. (2007) Evolution Control Engineering of Distributed Automation Components: Der εCEDAC Ansatz (in German). In: Proceedings of SPS/IPC/DRIVES Elektrische Automatisierung, Nürnberg (Germany), November 2007, pp. 297-306

Steininger, H., Strasser, T., ***Sünder, C.***, Zoitl, A. (2007) Die rekonfigurierbare Fertigung (in German). Computer&Automation, WEKA Fachmedien GmbH, 11/2007, ISSN 1615-8512, pp. 54-60

Sünder, C., Zoitl, A., Rofner, H., Strasser, T., Brunnenkreef, J. (2007) Benchmarking of IEC 61499 runtime environments. In: Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Patras (Greece), September 2007, pp. 474-481

Sünder, C., Hummer, O., Favre-Bulle, B. (2007) Enhanced Engineering of Downtimeless System Evolution by use of Hardware Capability Descriptions within the εCEDAC Approach. In: Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Patras (Greece), September 2007, pp. 764-767

Weehuizen, F., Brown, A., ***Sünder, C.***, Hummer, O. (2007) Implementing IEC 61499 Communication with the CIP Protocol. In: Proceedings of 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07), Patras (Greece), September 2007, pp. 498-501

Zoitl, A., Strasser, T., Hall, K., Staron, R., ***Sünder, C.***, Favre-Bulle, B. (2007) The Past, Present, and Future of IEC 61499. Lecture Notes in Computer Science (LNCS 4659), In: Proceedings of 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'07), Regensburg (Germany), September 2007, Springer Verlag Berlin Heidelberg, pp. 1-14

Rooker, M. N., ***Sünder, C.***, Strasser, T., Zoitl, A., Hummer, O., Ebenhofer, G. (2007) Zero Downtime Reconfiguration of Distributed Automation Systems: The εCEDAC Approach. Lecture Notes in Computer Science (LNCS 4659), In: Proceedings of 3rd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'07), Regensburg (Germany), September 2007, Springer Verlag Berlin Heidelberg, pp. 326-337

Sünder, C., Rofner, H., Vyatkin, V., Favre-Bulle, B. (2007) Formal description of an IEC 61499 runtime environment with real-time constraints. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 853-859

Sünder, C., Zoitl, A., Christensen, J. H., Colla, M., Strasser, T. (2007) Execution Models for the IEC 61499 elements Composite Function Block and Subapplication. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1169-1175

Hummer, O., ***Sünder, C.***, Strasser, T., Rooker, M. N., Kerbleder, G. (2007) Downtimeless System Evolution: Current State and Future Trends. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1123-1128

Baier, T., Fritsche, J., Keintzel, G., Loy, D., Schranz, R., Steininger, H., Strasser, T., ***Sünder, C.*** (2007) Future scenarios for application of downtimeless reconfiguration in indus-

trial practice. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1129-1134

Zoitl, A., ***Sünder, C.***, Strasser, T., Colla, M. (2007) A Device and Resource Execution Model for IEC 61499 Control Devices. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1143-1149

Strasser, T., ***Sünder, C.***, Zoitl, A., Rooker, M. N., Ebenhofer, G. (2007) Enhanced IEC 61499 Device Management Execution and Usage for Downtimeless Reconfiguration. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna (Austria), July 2007, pp. 1163-1168

Strasser, T., ***Sünder, C.***, Rooker, M. N., Hummer, O., Zoitl, A., Müller, I. (2007) Enhanced IEC 61499 System Model for Evolution of Control Applications in Distributed Industrial-Process Measurement and Control Systems. In: Proceedings of the European Control Conference (ECC'07), Kos (Greece), July 2007, pp. 1356-1363

Sünder, C., Zoitl, A., Dutzler, C. (2006) Functional structure-based modeling of automation systems. International Journal of Manufacturing Research (IJMR), Inderscience Publishers, vol. 1, nb. 4, ISSN: 1750-0591, pp. 405-420

Sünder, C., Favre-Bulle, B., Vyatkin, V. (2006) Towards an Approach for the Verification of Downtimeless System Evolution. In: Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 1133-1136

Hummer, O., ***Sünder, C.***, Zoitl, A., Strasser, T., Rooker, M. N., Ebenhofer, G. (2006) Towards Zero–downtime Evolution of Distributed Control Applications via Evolution Control based on IEC 61499. In: Proceedings of 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Prague (Czech Republic), September 2006, pp. 1285-1292

Sünder, C., Zoitl, A., Christensen, J.H., Vyatkin, V., Brennan, R.W., Valentini, A., Ferrarini, L., Strasser, T., Martinez-Lastra, J.L., Auinger, F. (2006) Interoperability and Useablity of IEC 61499. In: Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN'06), Singapore (Singapore), July 2006, pp. 31-37

Sünder, C., Zoitl, A., Rainbauer, M., Favre-Bulle, B. (2006) Hierarchical Control Modelling Architecture for Modular Distributed Automation Systems. In: Proceedings of the 4th IEEE International Conference on Industrial Informatics (INDIN'06), Singapore (Singapore), July 2006, pp. 12-17

Favre-Bulle, B., ***Sünder, C.***, Zoitl, A. (2006) Verteilt automatisieren (in German). Computer&Automation, WEKA Fachmedien GmbH, 3/2006, ISSN: 1615-8512, pp. 32-36

Strasser, T., Müller, I., Zoitl, A., ***Sünder, C.***, Grabmair, G. (2006) Verteilte, rekonfigurierbare Automatisierungssysteme im Produktionsbereich (in German). Industrie Magazin, Industriemagazin Verlag GmbH, 03/2006, pp. 33-36

Sünder, C., Zoitl, A., Strasser, T., Favre-Bulle, B. (2005) Intuitive Control Engineering for Mechatronic Components in Distributed Automation Systems based on the reference model of IEC 61499.

Sünder, C., Zoitl, A., Favre-Bulle, B., Strasser, T., Steininger, H., Thomas, S. (2006) Towards Reconfiguration Applications as basis for Control System Evolution in Zero-downtime Automation Systems. In: Innovative Production Machines and Systems, Proceedings of 2nd I*PROMS Virtual International Conference, July 2006, pp. 523-528

Strasser, T., Müller, I., Schüpany, M., Ebenhofer, G., Mungenast, R., ***Sünder, C.***, Zoitl, A., Hummer, O., Thomas, S., Steininger, H. (2006) An Advanced Engineering Environment for Distributed & Reconfigurable Industrial Automation & Control Systems based on IEC 61499. In: Proceedings of 2nd I*PROMS Virtual International Conference on Innovative Production Machines and Systems, July 2006, pp. 493-498

Sünder, C., Zoitl, A., Mehofer, F., Favre-Bulle, B. (2006) Advanced use of PLCopen Motion Control Library for autonomous Servo Drives in IEC 61499 based automation and control systems. Elektronik und Informa-tionstechnik (e&i), vol. 123, nb. 5, May 2006, Springer Verlag Wien New York, ISSN: 0932-383X, pp. 191-196

Zoitl, A., ***Sünder, C.***, Terzic, I. (2006) Dynamic Reconfiguration of Distributed Control Applications with Reconfiguration Services based on IEC 61499. In: Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06), Prague (Czech Republic), June 2006, pp. 109-114

Strasser, T., Müller, I., ***Sünder, C.***, Hummer, O., Uhrmann, H. (2006) Modeling of Reconfiguration Control Applications based on the IEC 61499 Reference Model for Industrial Process Measurement and Control Systems. In: Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06), Prague (Czech Republic), June 2006, pp. 127-132

Zoitl, A., Smodic, R., ***Sünder, C.***, Grabmair, G. (2006) Enhanced Real-Time Execution of Modular Control Soft-ware based on IEC 61499, In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA'06), Orlando (FL, USA), May 2006, pp. 327-332

Sünder, C., Zoitl, A., Strasser, T., Favre-Bulle, B. (2005) Intuitive Control Engineering for Mechatronic Components in Distributed Automation Systems based on the reference model of IEC 61499. In: Proceedings of 3rd International IEEE Conference on Industrial Informatics (INDIN'05), Perth (Australia), July 2005, pp. 50-55

Zoitl, A., Grabmair, G., Auinger, F., ***Sünder, C.*** (2005) Executing real-time constrained Control Applications modelled in IEC 61499 with respect to Dynamic reconfiguration. In: Proceedings of 3rd International IEEE Conference on Industrial Informatics (INDIN'05), Perth (Australia), July 2005, pp. 62-67

Strasser, T., Müller, I., Zoitl, A., ***Sünder, C.***, Grabmair, G. (2005) A Distributed Control Environment for Reconfigurable Manufacturing. In: Proceedings of 1st I*PROMS Virtual International Conference on Innovative Production Machines and Systems, July 2005, 6 pp.

Strasser, T., Zoitl, A., Auinger, F., ***Sünder, C.*** (2005) Towards Engineering Methods for Reconfiguration of Distributed Real-Time Control Systems Based on the Reference Model of IEC 61499. Lecture Notes in Computer Science (LNCS 3593), In: Proceedings of 2nd International Conference on Industrial Applications of Holonic and Multi-Agent Systems (HoloMAS'05), August 2005, Copenhagen (Denmark), Springer Verlag Berlin Heidelberg, pp. 165-175

Sünder, C. (2004) Integration of Motion Control in Distributed Automation Systems according to IEC 61499. Master thesis, Vienna University of Technology, Automation and Control Institute, June 2004

Oberhauser, K., Nemecek, A., ***Sünder, C.***, Zimmermann, H. (2004) Universal Integrated PIN Photodetector. In: Proceeding of 34th European Solid-State Device Research Conference (ESSDERC'04), Leuven (Belgium), September 2004, pp. 349-352