FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Kopplung von Simulation und Automationsinfrastruktur

Eine auf FMI und IEC 61499 basierende Analyse

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Michael H. Spiegel, BSc
Matrikelnummer 01125727

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr. techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Bernhard Heinzl, BSc

Wien, 20. Februar 2018

Michael H. Spiegel	Wolfgang Kastner

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Linking Simulation and Automation Infrastructure

A Study Based on the FMI and IEC 61499

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Michael H. Spiegel, BSc
Registration Number 01125727

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr. techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Bernhard Heinzl, BSc

Vienna, 20th February, 2018

Michael H. Spiegel

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Michael H. Spiegel, BSc
Wurzbachtalgasse 25, 1140 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2018

Michael H. Spiegel

Acknowledgements

I would like to thank the Austrian Institute of Technology (AIT) which funded this thesis and the Automation Systems Group at TU Wien represented by Prof. Wolfgang Kastner, Dipl.-Ing. Bernhard Heinzl and Dipl.-Ing. Andreas Fernbach who supervised the presented work, provided valuable scientific input and enabled an abundant collaboration. Furthermore, I would like to thank Dr. Edmund Widl who encouraged me to extend the research which was done for my Bachelor thesis and who kindly supervised my work at the AIT. I also want to express my kind gratitude to my other colleagues at the AIT and to the participants of the ERIGrid project for the numerous fruitful discussions and for providing me the initial models of the test cases. This thesis is partly supported by the European Union's Horizon 2020 research and innovation program (H2020/2014-2020) under project "ERIGrid" (Grant Agreement No. 654113).

Last but not least I want to thank my fiancé Stefanie Berger who spent hours on proof-reading this thesis as well as her and my family for continuously supporting me in my studies. Thank you Steffi, Heide, Thomas, Kathi and Georg!

Kurzfassung

Obwohl sie von einer breiten Öffentlichkeit oft nicht bemerkt werden, beeinflussen Automationssysteme weite Bereiche unseres täglichen Lebens. Die Herstellung vieler hochqualitativer Güter, eine kontinuierliche Energieversorgung oder moderne Gebäudetechnik beispielsweise basieren auf hochgradig automatisierten Prozessen. Mittels numerischer Simulation können diese Systeme mit reduziertem Prototypenaufwand entwickelt werden. Allerdings sind direkte Interaktionsmöglichkeiten mit Automationssystemen während deren Laufzeit bislang limitiert. Um die effiziente Entwicklung komplexer, zuverlässiger Cyber-physical Systems voranzutreiben, werden im Zuge dieser Arbeit Kopplungsmöglichkeiten zwischen ereignisbasierten Automationssystemen und numerischen Simulationen systematisch identifiziert. Einsatzbereiche dieser Kopplungspunkte reichen von hybriden Simulationen kontinuierlicher Anlagenmodelle mit diskreten Regelungen bis hin zur Emulation realer Anlagenteile, welche in einem Laborumfeld schwer realisierbar sind.

Die hier präsentierte Forschung wird auf Basis offener Standards, speziell der IEC 61499 für Automationssysteme und den Functional Mock-up Interface (FMI) Standards für numerische Simulationen, durchgeführt. Durch die systematische Diskussion können zahlreiche Kopplungspunkte und -strategien in einer umfassenden qualitativen Studie identifiziert und klassifiziert werden. Eine prototypische Implementierung eines Simulationsprogramms zur Erzeugung virtueller Komponenten ergänzt die eingangs durchgeführte Studie. Um die implementierte Methodik zu evaluieren, werden zwei Testfälle aus dem Bereich der intelligenten elektrischen Energienetze umgesetzt und ausgewertet. Die Testfälle demonstrieren, dass die Instanziierung von standardbasierten virtuellen Komponenten in Automationsinfrastruktur prinzipiell umsetzbar ist und dass der Einsatz einer neuartigen vorhersagebasierten Ereignissynchronisation die Einbindung von Modellen ermöglicht, die bei klassischer periodischer Synchronisation nicht akkurat gekoppelt werden können.

Die Diskussion zeigt, dass die Menge der identifizierten Kopplungspunkte nicht auf klassische Hardware-in-the-Loop (HIL) Szenarien beschränkt ist. Damit eröffnen sich auch neue Wege in der Testentwicklung. Die praktische Auswertung demonstriert die Machbarkeit standardbasierter virtueller Komponenten auf der Grundlage ereignisbasierter Steuerungen und unterstreicht die Notwendigkeit einer sorgfältigen Auswahl der eingesetzten Kopplungsansätze.

Abstract

Although the proper operation of automation systems is widely invisible to the general public, control infrastructure is extensively deployed and strongly affects our daily lives. For instance, manufacturing of high-quality mass goods, continuous energy supply infrastructure, and modern buildings highly depend on automation. Numerical simulation allows to efficiently engineer these complex systems and reduces the need of costly prototypes, but direct interaction with automation systems at runtime is still limited. This thesis addresses the efficient development of complex dependable Cyber-Physical Systems (CPS) by systematically pinpointing links between event-based automation systems and numerical simulation. Applications of such links range from hybrid system simulations, which couple continuous plant models and discrete control logic, to simulation-backed virtual components, which mimic parts of a system that cannot be easily included in a laboratory environment.

The research is based on open standards, in particular the IEC 61499 for event-based automation systems and the Functional Mock-up Interface (FMI) standards for model exchange and co-simulation. Systematic hierarchical categorization, which is applied in a comprehensive study, and a qualitative discussion based on a set of relevant features, such as real-time operation, show a broad range of viable links between IEC 61499 and the FMI. The theoretical study is amended by a prototypical implementation of a generic component simulator. Two test cases in the context of smart electrical grids are arranged to assess the implemented methodology. The instantiation of event-based virtual components in automation systems is successfully demonstrated. By applying a novel predictive event-based execution, it is shown that models which cannot be accurately coupled via conventional periodic synchronization, can successfully interact with assessed controllers.

The study reveals that the range of links between IEC 61499 and the FMI well exceeds the spectrum of traditional Hardware-in-the-Loop (HIL) setups and may guide test engineers in implementing novel assessment methodologies. Presented practical evaluations demonstrate the feasibility of standard-based virtual components and highlight the importance of a proper methodological selection.

Contents

CHAPTER 1

Introduction

Our daily life is strongly influenced by ubiquitous technologies. An electrical grid delivers power for a vast amount of appliances, a chain of complex industrial processes is maintained to deliver daily goods and a huge amount of systems is connected to provide mobility services. All these systems and systems of systems require careful engineering and impose major design challenges to gain appropriate dependability [59]. This thesis tackles some engineering challenges by studying the linkage of test and automation infrastructure with mathematical system models.

1.1 Background and Motivation

Manufacturing plays an important role in the economy of the European Union (EU) and many other economies worldwide. For instance, approximately 20 % of all jobs in the EU are provided by the manufacturing sector which achieves a turnover of about 7000 billion Euros [28, 56]. Today's manufacturing industries face many challenges, such as ever-increasing flexibility requirements, the need to reduce energy consumption, and the transition towards a more environmentally friendly production [27, 28, 45]. Many of these challenges are also subsumed in the vision of Industry 4.0 which includes harnessing and adopting existing advanced technologies from the Information and Communication Technology (ICT) domain for production [27]. The relevance of modernizing the manufacturing sector in the EU was identified years ago [56], but efforts are still ongoing [27, 28, 56, 80].

A large number of initiatives towards the modernization of the production sector have been launched, but still major obstacles for adopting advanced manufacturing techniques exist [56]. Data access issues from industry partners and other systems as well as the lack of advanced development tools were identified as major barriers. Several initiatives to combat the lack of standardized solutions have been launched, including initiatives from industry and politics, but many open questions regarding standardization remain [80].

Another domain facing major challenges is the energy sector, including electric and thermal energy systems [4, 80]. Our economy, which includes the manufacturing sector, as well as our daily lives vitally depend on energy supply systems as the basis for most technical processes. A high penetration of Renewable Energy Sources (RES) into the electric grids is required to reduce the amount of greenhouse gases which are emitted. Nonetheless, many RES, such as solar resources and wind energy, induce a considerable amount of power variability which potentially thwarts system stability [93, 94]. Furthermore, a transition of public and private transport towards a low-emission electrically powered fleet additionally demands electric energy networks. Flexibility options such as demand-side response, flexible generation and grid infrastructure, as well as energy storages may be deployed to foster the integration of RES and electric vehicle charging, but still many hurdles for a widespread adaption exist. Such open research questions in creating a flexible, smart electrical grid concern for instance control and scheduling strategies as well as efficient development, test, and verification infrastructure.

Major challenges in manufacturing as well as in the energy domain are often faced by the use of advanced automation and ICT [45, 59, 80]. Conventional distribution networks, for instance, are statically configured and operated. However, a high share of volatile RES often exceeds the limits given by the static configuration and the renewable energy that is fed into the grid has to be throttled. Automated grid components such as smart transformers may provide required flexibility and adapt to volatile renewable power generation. Since both technical domains, energy and manufacturing automation, are highly interconnected and often use similar technologies, methodologies from one domain may be applied to the other domain as well. It was, for instance, shown that particular controllers can be used in factory and energy system automation [4, 89, 102, 103].

Some years ago, the concept of event-based control in automation was introduced to supersede and enhance traditional cyclic execution of control algorithms and to increase the flexibility of traditional automation systems [50, 102, 103]. The International Electrotechnical Commission (IEC) standard 61499 was created which specifies a system-level design language of potentially distributed automation systems. Although the standard is currently not widely used in industry [109], it receives a lot of attention in academia and is successfully used to implement novel control and engineering concepts [45, 90, 102]. In particular, the event-based execution model allows to implement a conventional cyclic operation and novel action-based designs.

Often, additional value is gained by interconnecting automation equipment and by a coordinated operation. Although the detailed potential of interconnected machines is still subject to intense research effort, possible use cases include advanced non-local control and optimization strategies, predictive maintenance, as well as the automation of manually operated business processes [3, 12, 28, 80]. Interconnected machines, which incorporate an interaction between an ICT system and a physical process, are often called Cyber-Physical Systems (CPS) [59]. Following the widespread use of ICT in many critical processes, dependability requirements, which are fundamentally different from general-purpose computing, are increasingly important. To efficiently fulfill these

requirements, novel development, testing, validation and verification methodologies are still required [28, 80].

In order to successfully use novel methodologies, the whole engineering workflow has to be taken into account. Often, it is necessary to adapt the workflow in order to fulfill particular dependability requirements. For instance, static timing analysis of hard real-time systems requires adapted coding practice to reach tight execution-time bounds. A methodology which integrates well into existing workflows may gain broader acceptance than a methodology which requires to broadly alter well-proven practices. Numerical models and their simulation are integral parts of many state-of-the-art engineering workflows. Simulation enables an engineer to assess and optimize major properties of a system and to reduce the amount of needed prototypes. A vast number of general-purpose and domain-specific modeling and simulation tools in many scientific and engineering domains exist [33]. Many of these tools focus on a very limited specialized domain, such as communication networks or fluids, but also for general-purpose modeling tools, the range of available models and capabilities varies. To gain a comprehensive view of complex systems, it is often beneficial to couple simulation tools or to combine models provided by multiple tools.

Additional need for model exchange and tool coupling arises in domains where multiple vendors and institutions which provide simulation models of subsystems are involved [13]. For instance, in the automotive domain, component manufacturers may provide models of their components which then can be used to assess the overall functionality of a system. In order to protect the intellectual properties of a component manufacturer, internals of the model must not be accessible and interactions with other models must be restricted to component boundaries. Some standards that address the combination of multiple simulation tools into a co-simulation and the exchange of models exist [38, 39, 40, 51], e.g. the Functional Mock-up Interface (FMI) standards for model exchange and co-simulation. Several tools from various domains already support one of the FMI standards, but interaction with the automation domain is still limited [33].

1.2 Problem Statement

One of the main motivations for conducting the work is the need of enhanced testing and validation techniques in the context of smart grid and automation infrastructure. One way of testing subsystems in the context of the whole system is to represent other relevant parts of the setup as virtual components which are interlinked with the physically available subsystems. Often, the targeted system or the laboratory components are backed by industrial automation infrastructure. To integrate virtual components into test setups, the selected point of interaction needs to be capable of emulating models in real time. Although it may also be constructive to integrate physical components into a simulation environment, the reversed mindset of integrating virtual components into a physically available infrastructure may be more intuitive. In the latter case, the models from previous design and engineering phases may be directly used in the testing phase.

Additionally, expected real-time properties strongly point towards an approach in which models or simulation tools are integrated into automation infrastructure.

Hardware-in-the-Loop (HIL) tests, which couple physically available hardware and numerical models, have a long tradition and are widely adapted in many industrial areas such as automotive and power electronics [11, 101]. Subsystem prototypes, for instance, may be efficiently tested in a HIL setup without the need to build related subsystems. The Hardware under Test (HuT) can be assessed in well-defined ways such as in rare events and border conditions which may not be easily triggered within a comprehensive prototype. The accuracy of HIL tests is strongly influenced by the HIL setup and timing properties thereof [62, 63, 101]. Insufficient synchronization, for instance, may easily lead to inaccuracies and instabilities.

Many commercial HIL simulators specifically focusing on solving system models in real time are available [34, 100]. Such specialized simulators can achieve update frequencies in the megahertz range and often communicate via dedicated digital and analog IO lines. To gain such an update performance, the modeling capabilities and the range of supported models are limited and often focused on a particular application domain. For many applications, such as static models of power systems and some thermal process models, the performance of dedicated HIL equipment is not essentially needed [3, 4, 75] and introduces a considerable cost overhead. In these cases, general-purpose computing hardware and tool-specific links are often used to reduce costs and to enable new use cases. Standard automation equipment such as network IO interfaces may be deployed to access hardware which does not provide a high-level communication interface. Often, simulation tools and automation infrastructure are coupled via tool-specific interfaces which increases the development effort. In order to seamlessly create virtual components in existing laboratory environments such as smart grid test stands [4], a standard-based interface which covers a broad variety of simulation models and tools is favorable.

Nevertheless, IEC 61499, which is also used in laboratory environments [4], focuses on control system architecture and run-time aspects and does not cover a detailed model-based engineering process. Similarly, communication protocol specifications, such as Modbus and the Abstract Syntax Notation One (ASN.1)-based protocol of IEC 61499, mostly focus on data exchange of process data only and do not include considerations on coupling virtual components [50, 65]. For instance, many industrial communication protocols do not timestamp messages and silently assume a real-time operation. Contrary, synchronization of simulation time is a crucial part in many co-simulation scenarios. Associating a message with an instant of time eases synchronization and the implementation of virtual components.

Some authors have already demonstrated the high potential of coupling simulation models and automation infrastructure [45, 88, 90, 91, 110] which well exceeds purely academic use cases. Coupling virtual components and physically available automation infrastructure, for instance, is used to build a training platform which allows to train students and personal on using the automation system without the risk of damaging expensive equipment. Nonetheless, standards for automation systems and system models

are most often independently conceived and not jointly envisioned. In particular, a comprehensive discussion on coupling the FMI and IEC 61499 on a general basis is still missing. It is believed that such a joint discussion enriches future coupling efforts and eases the selection of an appropriate methodology.

Today, first simulation and automation tools implement basic coupling approaches [26, 66, 86], but a widespread adaption of standard-based simulation tool and automation infrastructure coupling is not seen. In particular, very little support in linking FMI-based models and tools to event-based control infrastructure such as IEC 61499-compliant controllers is available. It was demonstrated that coupling event-based systems and continuous or hybrid models requires special attention to gain accurate results [8, 68, 108]. Although the initial studies were not applied to real-time systems, the fundamental principles of synchronization also hold on automation infrastructure. In particular, applying event prediction as a form of extrapolation may also reduce unwanted delay effects in real-time systems.

1.3 Main Goals

In order to bridge gaps between IEC 61499 and the FMI, the thesis first presents a comprehensive theoretical study on linking both standards in a single setup. Thereby, the study addresses the following research questions:

> In which ways can FMI-based models or tools and IEC 61499-based controllers be coupled? What are the expected implications of each strategy on important aspects such as real-time operation and automatic model transformation?

The state of the art in linking the FMI and IEC 61499 shall be further advanced by the implementation and evaluation of one principal way of interaction. In particular, the implementation tackles a predictive and a periodic synchronization approach. Special attention is to be put on a broad applicability and re-usability of developed software components beyond any particular test cases. By design, future testing and validation efforts should be actively reduced when using the provided tool-independent interface.

The primary aim of the evaluation is to demonstrate findings of the theoretical study in a proof-of-concept implementation and to further advance the knowledge of major influence factors on the implementation performance. The evaluation specifically focuses on HIL-related aspects such as the real-time performance of the implementation. Special attention has to be put on carving out implications of both synchronization approaches and on comparing their results to the theoretical expectations derived from the interaction study.

1.4 Methodology

The theoretical interaction study is conducted by first defining principal ways of interaction for linking IEC 61499 and the FMI. A set of important features is defined which is then used to structure the discussion. To achieve the applicability goals, important features specifically focus on testing, validation and HIL-related aspects. For each principal way of interaction, one or more coupling strategies are defined and discussed. To ease generalization and application, the discussion in the theoretical part is entirely based on the IEC 61499 and FMI standards and does not tackle specific implementations. Since the implementation and quantitative discussion of every principal way of interaction is beyond the scope of this thesis, a qualitative comparison between listed linkage strategies is presented.

Based on the interaction study and the targeted virtual component use cases, an interface software is designed and implemented. The software design covers a detailed specification of software objectives and the resulting decomposition into major design entities. A software development process and quality assurance measures guide the implementation in order to increase the confidence of the experimental results and to meet the expectations regarding a general applicability beyond the specified test cases.

For evaluation purposes, a set of test cases as well as a detailed evaluation methodology are defined. The test cases specifically focus on border cases and may not reflect an average-case scenario. In particular, the test cases are used to assess the limits of the implemented coupling approaches and to highlight differences between the approaches. A monolithic reference simulation which does not use the described standards is conducted to assess effects and implications of the coupling approaches. The reference simulation itself is also used to define any control logic and to export relevant models, but will not directly use the FMI. Hence, effects which occur due to a re-implementation of existing models can be reduced to a minimum and the coupling strategies can be specifically studied.

All implemented test cases include external hardware to demonstrate the feasibility of HIL setups. Nevertheless, pure software implementations are used to study effects which cannot be easily observed by including external hardware. Such effects include detailed delay observations which can be more accurately traced without including external monolithic hardware and network connections.

1.5 Structure

This thesis is structured as follows: Chapter 2 reviews related work and defines the contribution of the conducted efforts. Various ways of linking IEC 61499 and the FMI are described in the comprehensive interaction study in Chapter 3. The prototypical implementation of one principal way of interaction is documented in Chapter 4 and the evaluation is presented in Chapter 5. Chapter 6 concludes the studies and sketches future research topics.

CHAPTER 2

Related Work

2.1 IEC 61499

The IEC standard 61499 provides a system-level design language for distributed automation systems [50, 103]. The following sections will give an overview of the IEC 61499 itself, summarize the discussions regarding IEC 61499, its application in industry, and available software tools. Section 2.1.1, which describes the foundations of the IEC 61499, is partly based on the author's previous work [85].

2.1.1 IEC 61499 Component and Execution Model

The IEC standard 61499 targets the design, execution and management of Industrial Process Measurement and Control Systems (IPMCSs) [50]. It defines a generic architecture which may be represented in graphical and textual form. It aims at an implementation-independent specification such that exchange of information among diverse software tools is enabled. An IPMCS which is specified and designed by means of IEC 61499 may couple devices from various vendors. The standard itself is split into three documents. The first document, IEC 61499-1, defines the reference model which may be used to specify systems and their components, some predefined instances of components, application management facilities, and an informal communication protocol. The second document, IEC 61499-2, defines requirements for software tool support for the specification, analysis and validation of IPMCSs. The last document, IEC 61499-4, lists rules for the development of compliance profiles which promote inter-vendor interoperability of devices and tools.

The architecture, which is defined in the IEC standard document 16499-1, consists of various reference models which cover the entire system down to a detailed modeling of sub-applications by means of a concept called Function Blocks (FBs). A system consists of a set of interconnected devices which interface a controlled process. The process

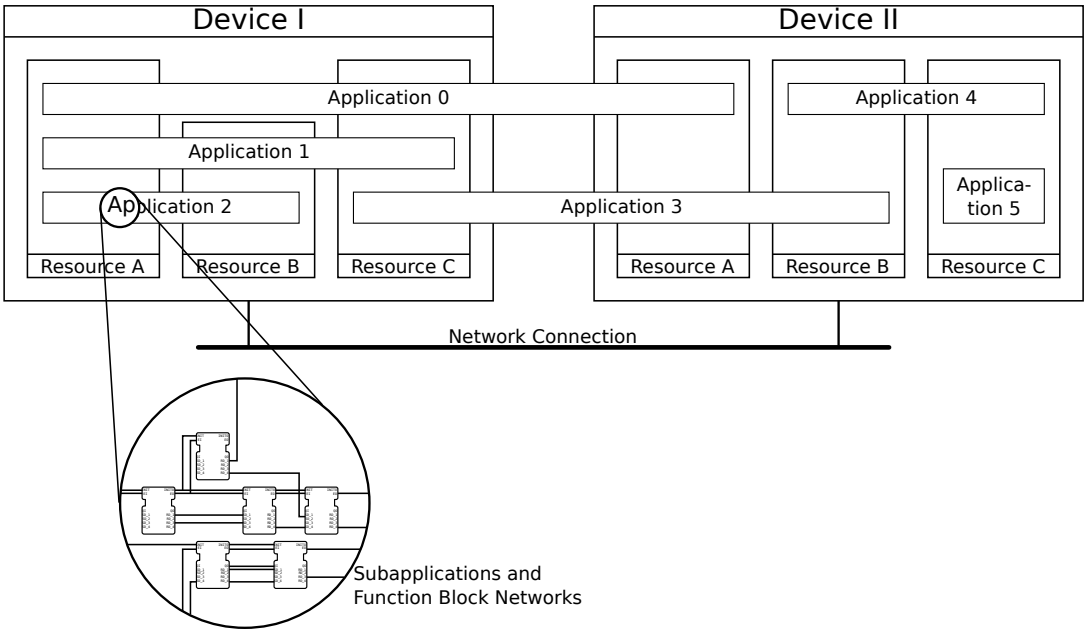


Figure 2.1: IEC 61499 device model example [85]

itself is not part of the architecture but communication links are explicitly modeled. Figure 2.1 shows an example of a system which consists of a communication link and two devices. Each device may host multiple resources which define an abstract functional unit with independent control of its execution. A resource may be managed without directly affecting other resources and defines a scheduling function which controls the execution of all associated functionalities.

The actual control logic is encapsulated in independent applications which may be distributed across multiple resources. Applications are not restricted to reside on a single device and may even be distributed across multiple devices. One key concept of IEC 61499-based controls is the encapsulation of certain functionalities into reusable FB types which may be instantiated as FBs. FBs expose their functionality via well-defined interfaces. Multiple FBs may be connected to an FB network in order to model more complex behavior. Applications may consist of such an FB network. In addition, IEC 61499 allows to structure applications by means of subapplication types and subapplication instances. Like in the case of applications, the functionality of subapplications may be modeled by a network of subapplications and FBs, but subapplications may also expose their functionality via an FB-like interface. FBs are the basic unit of distribution, i.e. a single FB instance may only reside on one device. In contrast, subapplications may be split across multiple devices.

The IEC 61499 introduces the notion of events as an occurrence, which is significant to trigger the execution of an algorithm. In contrast to the mode of operation defined by

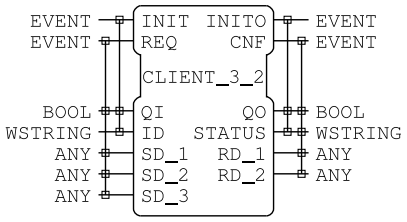


Figure 2.2: IEC 61499 function block interface example

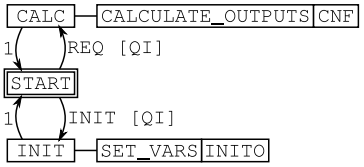


Figure 2.3: IEC 61499 ECC example

IEC 61131, algorithms and control logic are not executed periodically per se but on the occurrence of an associated event. Therefore, subapplication and FB types may specify event in- and outputs which are used to receive and transmit events. As soon as an event from a connected FB or subapplication is received, the execution of the encapsulated control logic starts. Events in the context of IEC 61499 are not directly associated with data, i.e. an event may be used solely for scheduling and may not be associated with any variable value. FB types may additionally define data in- and outputs. The type system as defined in IEC 61131 is used to specify the type of internal variables, in-, and outputs [50, p.68]. One can use FB data in- and outputs to pass values between FBs and subapplications. In case of FB types, data in- and outputs may be associated with one or more event in- and outputs respectively. As soon as an incoming event is triggered, all associated data inputs will be sampled and the control logic of the FB can access updated input data. Figure 2.2 shows an exemplary graphical FB interface specification. By definition, inputs are always drawn at the left, outputs at the right side of the FB. The top part of the FB representation specifies event in- and outputs and the bottom part lists available data in- and outputs. In the example, `QI` and `ID` are sampled with the `INIT` event as indicated by rectangular markings in front of the input names.

The IEC 61499 defines three kinds of FBs: Basic Function Block (BFB), Composite Function Block (CFB), and Service Interface Function Block (SIFB). All three kinds differ in the way the behavior of FB instances is specified. BFBs use algorithms and a mechanism which is called Execution Control Chart (ECC) to define the execution semantics. An ECC is a structure inside each FB instance which consists of a finite amount of states and transitions between two states. Figure 2.3 shows a typical graphical representation of a simple ECC. Algorithms are associated with states and will be executed as soon as the FB enters the associated state. Transitions may be guarded by an input event and a guard condition on some variables of the FB. In case the input event is triggered and the guard condition is fulfilled, the transition is taken and the

following state is entered. The execution of the FB stops if no more transitions can be taken. The specification of the algorithms itself is beyond the scope of the standard. Nevertheless, the languages defined in IEC 61131-3 in general and Structured Text (ST) in particular are explicitly referenced as specification mechanisms for algorithms. As soon as an algorithm terminates, an associated output event may be triggered which schedules the execution of connected FB instances.

The behavior of CFB instances is defined by the component FBs which reside in the CFB as well as the interconnections between component and CFB in- and outputs. A CFB input, for example, may be connected to a CFB output or to inputs of hosted FBs. Incoming events will be relayed to all connected FBs and trigger nested functionality. Likewise, output events of the CFB are triggered if a connected instance triggers an event. Consequently, the principal operation of CFBs is similar to the execution of subapplications. Nevertheless, a CFB cannot be split among multiple resources and is therefore subject to a single scheduling function only.

SIFBs provide interfaces to resources such as network connections and IO-ports. The behavior of SIFB types is specified in an abstract manner by defining the causal dependency between in- and output events. Service sequence diagrams are used to formally represent feasible sequences of events. For instance, an event at the initialization input `INIT` will eventually trigger an event at the output `INITO`. The exact behavior in terms of value transformations and side effects is not formally specified. SIFBs are the only kind of FBs which can actively trigger an event without previously receiving one. Hence, interactions may be initiated by the resource and not by the application. SIFBs which provide application-initiated interactions or a mixture of resource- and application-initiated interactions are feasible as well. IEC 61499 defines, in addition to the general means for specifying SIFB types, some generic SIFB types such as communication FBs.

A mechanism for encapsulating event and data flows, which is called adapter interface is also defined in IEC 61499. Adapter interface types specify a set of in- and output events and variables one can use to provide and request a certain service. For instance, controllers which need to pass a work piece from one station to another may encapsulate a two-way handshake into a single adapter. In case an FB or subapplication provides the interface specified by the adapter, it exposes a generic plug of the adapter interface type. All plugs are part of the interface of the particular entity. Similarly, a so-called socket is exposed in case an FB or subapplication may use the interface which is defined by the adapter. Plugs and sockets may be connected in CFBs and subapplications to match provider and acceptor instances. Hence, the adapter mechanism may be used to reduce the number of connections and to define abstract services between FB instances.

The platform specific program, which interprets the IEC 61499-compliant configuration and executes the control flow, is often called Run-time Infrastructure (RTI) [37]. Although some features of the RTI may be accessed via SIFBs, the definition of an RTI itself is beyond the scope of the IEC 61499.

2.1.2 Discussion of the IEC 61499

IEC 61499 is sometimes referred to as successor of the industry-leading IEC 61131 standard. Although the standard is currently not widely adapted in industry [109], its capabilities have been examined in various publications [92, 102, 103], showcasing its benefits in industrial applications [4, 69, 88, 90, 91, 110].

In 2009, Vyatkin described the capabilities of the IEC 61499 standard and motivated its extensions in comparison to the widely adopted IEC 61131 standard [103]. He noted that IEC 61499 features a vendor-neutral Extensible Markup Language (XML)-based exchange format which tackles the limited portability of IEC 61131-based implementations. On the basis of the first version of IEC 61499, Vyatkin identified some execution issues and ambiguities. For instance, in some cases, the order of executed events was not defined unambiguously. Strasser, too, worked out some execution issues which are mostly related to run-time aspects [92]. He presented several standard-compliant execution and scheduling models which may lead to different results. A CFB, for instance, may be executed atomically or transparently via its sub-function blocks. In 2011, Vyatkin concluded that the second version of the IEC standard eliminates most of the ambiguities [102].

Chia-Han et al. surveyed model-driven development concepts of control software and listed several causes of the slow adaption of IEC 61499 in industry [109]. For instance, IEC 61499-based applications require additional design decisions such as distribution concepts, which increase the switching costs. A steep learning curve and missing support in many well-established tools additionally hinders the usage. Existing IEC 61499 tools are not as mature as conventional software which is used to engineer Programmable Logic Controllers (PLCs) and portability of program code between different IEC 61499-based tools is not always guaranteed.

Some effort regarding real-time analysis of IEC 61499-based controls has already been conducted [61, 92, 102]. For instance, Lindgren et al. noted that it is currently not possible to express real-time semantics in the execution model defined by IEC 61499 [61]. All timing semantics directly emerge from a specific implementation. The authors proposed an extension which adds real-time semantics to the IEC 61499. The extension uses nondeterminism to abstract the actually deployed scheduling policy and to maintain backwards compatibility. As a proof of concept, timing semantics of IEC 61499 were implemented in the Real Time For the Masses framework core language.

2.1.3 Application of IEC 61499-based Controls

The applicability of IEC 61499 for control applications was demonstrated in many research and industrial projects showcasing its capabilities [102]. Strasser studied the application of IEC 61499 in closed-loop control [89]. He noted the significant difference between the execution models of IEC 61499 and IEC 61131 which plays an important role for control applications. While IEC 61131 uses a strictly cyclic execution model, IEC 61499 features an event-based execution which schedules tasks according to explicitly modeled processing

dependencies [50, 89]. Strasser implemented the cyclic operation of closed-loop controls via cyclic event execution and demonstrated the control capabilities at a seesaw balancing experiment [89]. It was noted that special attention has to be put on synchronous delivery of data and events in case of distributed control algorithms. Similarly, Hametner applied IEC 61499-based closed-loop control to a helicopter model [44]. He estimated the Worst Case Execution Time (WCET) of some FBs by measuring the execution time in various configurations. He pointed out that if the control loop output is not properly synchronized, the behavior of the system cannot be accurately described in the z-domain.

Hegny et al. used IEC 61499-based applications to execute plant models and to test IEC 61499-based controllers [45, 46]. The models are represented as timed state charts which are a subset of Unified Modeling Language (UML) state chart diagrams. Model to model transformation is used to convert the UML models into IEC 61499 applications. The concept was tested on the model of a sorting machine. The integration of simulation in the development process was further discussed in more detail by showing several coupling concepts [47]. Various levels of integration were identified, ranging from the direct integration within the controlled plant, models which replace SIFBs, to the interaction with external simulation tools. Later, the state of the art in model-driven development of control software was summarized by Chia-Han et al. [109]. Several software components and frameworks which support modeling and simulation of control applications are already available. Often, hybrid modeling which supports both continuous and discrete-time modeling is required. For IEC 61499-based controllers, a design pattern which targets model-driven design of control applications was developed [47, 58, 109]. The design pattern is called layered model view controller pattern and separates the plant model from the control logic such that the control logic can be easily deployed in the production plants. Further work on integrating IEC 61499-based controls in co-simulation is discussed in Section 2.2.3.

Andrén et al. proposed a semantic-driven framework for smart grid applications which combines several domain models to specify key aspects of smart grid applications [5]. IEC 61499 was chosen as a domain model for control-related aspects. The applicability of IEC 61499-based controls in smart grid applications was demonstrated in various co-simulation [69, 88, 90, 91, 110], test, and production setups. Andrén et al., for instance, also presented a HIL test stand which is specifically tailored to smart grid applications [4]. An IEC 61499-based controller and a Supervisory Control and Data Acquisition (SCADA) system were deployed which control the test stand and provide the required flexibility.

2.1.4 Tool-Support for IEC 61499-based Applications

Several implementations of the IEC standard 61499 are available, ranging from academic projects to industrial products [109]. One of the first implementations was the Function Block Development Kit (FBDK) [29, 30]. FBDK consists of an IEC 61499-compliant editor as well as a run-time environment which executes the control logic. Similarly, Eclipse 4diac provides an Eclipse IDE-based development environment and a run-time environment which executes the control applications [22]. Both components are released

under an open-source license. The run-time environment of Eclipse 4diac is, in contrast to FBDK, implemented in C++ and specifically targets small embedded control devices [37]. Nevertheless, also Windows and Linux workstations are supported.

Also, some commercial IEC 61499-compliant packages are available. NXT Control, for instance, offers an engineering environment and corresponding controllers which target IEC 61499 as well as IEC 61131-based control applications [71]. The commercial engineering environment does not only focus on control applications but also targets Human Machine Interfaces (HMIs) and SCADA systems.

2.2 Co-Simulation and Model Exchange

Co-simulation corresponds to the process of coupling several domain-specific tools and models. It is widely applied beyond the automation domain and enables an engineer or scientist to utilize various domain-specific tools in inter-domain problems [8, 88]. Consequently, tools, which model a specific aspect (e.g. energy consumption) each, can be jointly executed in a co-simulation. Section 2.2.1, which is partly based on the author's previous work [85], describes the concepts of the FMI standards for co-simulation and model exchange. Sections 2.2.2 and 2.2.3 describe work related to co-simulation standards, including the FMI, and the use of IEC 61499 in co-simulation, respectively.

2.2.1 Functional Mock-up Interface Standard

The FMI standard defines facilities to couple one or more models and/or co-simulation tools [13, 38, 39, 40]. At the time of writing, two versions of FMI are available, FMI 1.0 and FMI 2.0. The second version introduced several improvements and optional features. FMI 2.0 is not fully compatible with the first version of the standard. Nevertheless, the main concepts, such as the mathematical representation of models and tools, differ only slightly. Hence, the following section covers both versions and highlights important differences.

FMI 1.0 is split into two separate standard documents which cover co-simulation and model exchange separately. FMI for model exchange requires an external component which numerically solves the exposed model equations. When using FMI for co-simulation, each coupled tool contains an independent solver. Data is exchanged at discrete points in simulation time only. The second version of FMI includes both variants in a single standard document but still differentiates between model exchange and co-simulation.

Simulation models and co-simulation slaves are exported as so called Functional Mock-up Units (FMUs). An FMU encapsulates all technical information which is needed to include the model or tool in a comprehensive simulation run. The interface is divided into two main parts: a static model description which includes metadata and an executable implementation which runs the model equations or co-simulation tool connection. The static model description is encoded in an XML format and C functions are defined to interface the executable implementation. FMI supports both, platform-dependent binary

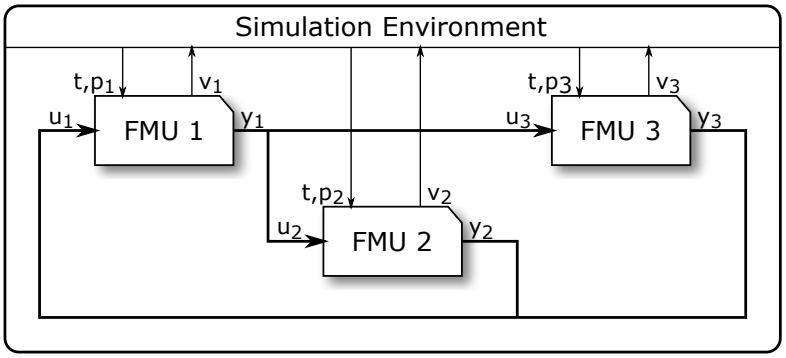


Figure 2.4: General FMI architecture

shared objects and source code representation of executable implementations. Static model descriptions are separated from dynamic C code in order to keep the C functions as lightweight as possible and to allow an embedded operation on constrained devices. All files which are necessary to describe the model and tool connection respectively are encapsulated into a single zip-compressed archive file.

To access an FMU, the model description needs to be parsed and the executable must be interfaced. Before executing a contained model or simulation, an FMU needs to be instantiated. Per default, multiple instances of a single FMU may be used simultaneously. In order to differentiate between the independent FMU instances, a data structure is dynamically generated which contains or references the internal variables of an instance. The details of that data structure are not exposed to the environment which uses the FMU. Only a pointer is maintained to address FMU instances.

Simulation or model data is accessed via model variables. Figure 2.4 shows an exemplary data exchange of FMI-based models or tools. FMUs do not directly exchange data with each other. The simulation environment actively performs data exchange and sets input model variables u_i according to the connected output variables y_j. Additionally, it manages a synchronized notion of time t, sets model parameters p_i, and records all exposed variables v_i. Again, static data such as the name and type of each variable is listed in the model description file. At run-time, variables are accessed via an integer typed value reference passed on to getter and setter functions. These functions enable an FMU to cache previously calculated results and to evaluate equations on demand. Four basic variable data types (real, integer, boolean, and strings) are defined. Additionally, the model description may define enumerations which are mapped to integer variables at run-time. FMI does not directly support structural data types such as C structs and arrays. Nevertheless, an optional variable naming convention which maps structured data types to multiple variables is included. An FMU may not expose the whole state via model variables which limits resetting an FMU to a previous state. FMI 2.0 introduces optional features which allow to save and reset the entire state of an FMU. In addition, state serialization and de-serialization functions may be provided. Nevertheless, the format of serialization is beyond the scope of the standard.

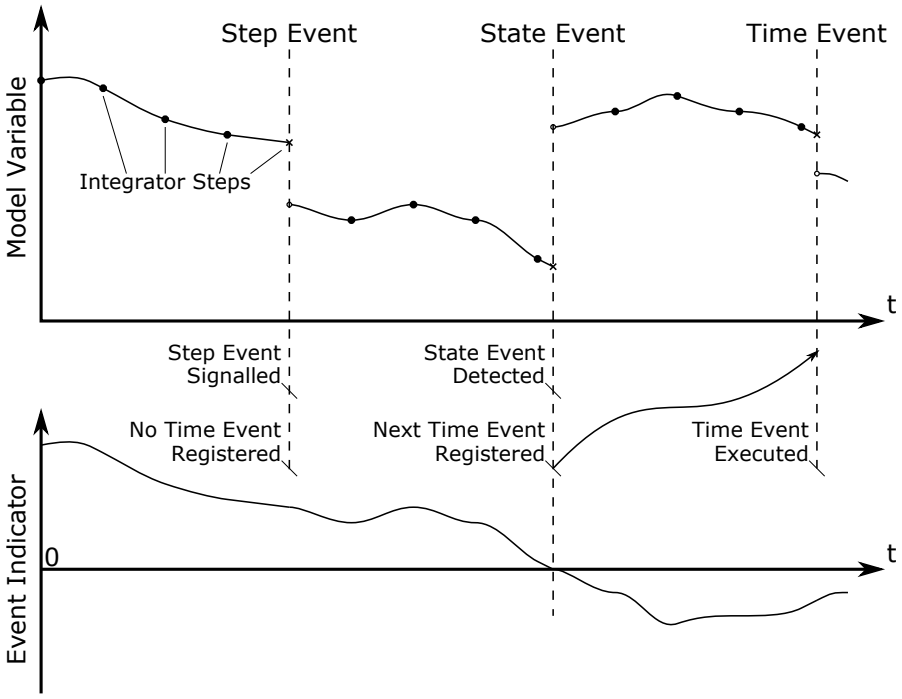


Figure 2.5: Event indication mechanisms

An exposed variable may be amended by a more detailed type and unit definition. For instance, a dedicated velocity type which uses $\frac{m}{s}$ as base unit may be defined. Additionally, several display units can be stated in the model description. FMI 2.0 replaces the simple identifier-based unit definition system of FMI 1.0 by an exponent-based system. In FMI 2.0, each unit is defined in terms of the seven SI units and an additional factor for angles. FMI 2.0 additionally allows to specify the structure of the model in more details. In particular, direct dependencies of each model variable in terms of its unknowns may be specified. The dependency information may be utilized to deploy an optimized handling of algebraic loops. An algebraic loop is formed whenever a direct circular dependency between connected FMU instances occurs. Since the model in- and output variables directly influence each other, convergence has to be established in order to obtain correct outputs. The model structure may also be taken into account to efficiently calculate sparse Jacobians. In general, FMI 2.0 also introduces an optional function which allows to retrieve partial derivatives of some model variables at a particular instant in time or at a particular communication point. The information may be utilized in various ways including advanced integration methods and linearization of a model.

Functional Mock-up Interface for Model Exchange

A model in FMI for model exchange is represented as a set of Ordinary Differential Equations (ODEs) in state space form with discrete events [39, 40]. All model variables

are functions with respect to an independent time variable. An event in terms of FMI is a discontinuity of an exposed variable which is assumed to be continuous between two event instances. In addition to all exposed model variables, an FMU for model exchange also provides functions to access the vector of states and their derivatives. Hence, an ODE solver may numerically solve the model. The ODE solver itself is not part of an FMU. An FMU may indicate events via three distinct facilities. A time event may be triggered by directly specifying the time instant of the next event while handling the current one. Hence, an FMU needs to be able to predict the next time event instance for a future point in time without necessarily knowing the solution at that instant of time. Dedicated event indicators may be used to signal state events while solving the continuous ODEs. An event is triggered as soon as the sign of an event indicator changes. More precisely, it is triggered when an event indicator changes from a value in $(\infty, 0]$ to a value in $(0, \infty)$ or vice versa. The ODE solver has to check all event indicators and must interrupt the process whenever an event is detected. An iterative procedure may be deployed to precisely determine the time of the current state event. Since the event indicators may directly depend on the state of the FMU, state event instances cannot be predicted. The third kind of events is called step events and may be triggered whenever an integration step finishes and can therefore be evaluated more efficiently than state events. Figure 2.5 illustrates all three event indication facilities.

Both versions, FMI 1.0 and 2.0, define a state machine which specifies the admissible sequence of function calls. In principle, modes which handle events and modes which forward continuous time by integration are supported. In event modes, it is not allowed to forward the time of the model. Several iterations may be necessary until the solution at an event instance converges. FMI 2.0 introduces several functions which explicitly mark mode transitions. These functions have to be called in order to switch the mode of the FMU. An FMU which follows FMI 1.0 implicitly switches modes by calling event or time update functions. In addition, an initialization mode was introduced in FMI 2.0 which explicitly allows to deploy a dedicated set of equations which may be used to evaluate the initial conditions.

Although a solver may freely set the time within the current integration step, FMI, in general, only allows increasing time. Hence, special attention has to be put on resetting to a particular instant of simulation time. Each event may potentially change the internal, discrete state of an FMU which may not be exposed. Since an FMU is not required to keep track of its state history, resetting the time before the last event may yield undesired results. FMI 1.0 uses the standard representation of dense time which is encoded in a floating point variable. FMI 2.0 extends that notion of time by a superdense representation. Formally, every time instant t follows the superdense representation $t = (t_\mathrm{R}, t_\mathrm{I})$ where $t_\mathrm{R} \in \mathbb{R}$ and $t_\mathrm{I} \in \mathbb{N}$ [40, p.69ff]. Intuitively, t_R corresponds to the instant of dense time and t_I enumerates the chain of causal events. Strict order in superdense time is defined by (2.1) and equality by (2.2):

$$(t_{\mathrm{R},1}, t_{\mathrm{I},1}) < (t_{\mathrm{R},2}, t_{\mathrm{I},2}) \quad :\Leftrightarrow \quad t_{\mathrm{R},1} < t_{\mathrm{R},2} \vee (t_{\mathrm{R},1} = t_{\mathrm{R},2} \wedge t_{\mathrm{I},1} < t_{\mathrm{I},2}) \tag{2.1}$$

$$(t_{\mathrm{R},1}, t_{\mathrm{I},1}) = (t_{\mathrm{R},2}, t_{\mathrm{I},2}) \quad :\Leftrightarrow \quad t_{\mathrm{R},1} = t_{\mathrm{R},2} \wedge t_{\mathrm{I},1} = t_{\mathrm{I},2} \tag{2.2}$$

The C interface of FMI 2.0 only implicitly uses superdense time by providing functions which access the dense time part, t_{R}, of the tuple and a function which increases the integer part, t_{I}, on signaling a new, valid discrete state. Nevertheless, superdense time is used to specify the exact semantic of the functions.

Functional Mock-up Interface for Co-Simulation

In contrast to model exchange FMUs, the FMI for co-simulation expects the FMUs to maintain their own solver [38, p.5f]. Communication between different FMUs takes place at discrete communication points only. Between the communication points, the FMUs have to solve their respective models independently. FMI for co-simulation specifies a master-slave architecture in which FMUs are directed by a simulation master. The communication between FMUs is handled by the simulation master implementing a master algorithm. The master algorithm itself is not specified by the FMI but it is advised that the master adapts the algorithm based on the capabilities of coordinated slaves. Optional capabilities include the support of variable communication step sizes or higher order signal extrapolation.

The FMI for co-simulation tries to support all stages of a simulation process including design, deployment, simulation and post-processing [38, p.8ff]. Although the FMI itself only specifies the interface functions, several different scenarios of model distribution are stated. These scenarios range from a one process scenario where the FMU directly executes the simulation code to a fully distributed simulation executed on different machines. Several communication mechanisms including shared memory or TCP/IP connections are feasible because the FMI does not specify any particular communication protocol beyond the C function-based interface.

Within the simulation phase, involved simulation tools regularly exchange information via the FMI [38, p.16]. It is assumed that the next communication step size is known a priori and that it is possible to interrupt the current simulation in order to exchange information at a given instant of time. Depending on the capabilities of the FMU, the simulation master may also discard and reject communication steps by setting a flag indicating that the last communication step was not accepted [38, p.29]. The second version of FMI adds an optional mechanism for directly saving and restoring the state of an FMU [40, p.24]. A communication step using this version may be rejected by restoring the previously saved state.

Like the FMI for model exchange, the FMI for co-simulation also relies on a static description of the FMU and a set of C functions accessing the co-simulation values [38, p.22ff]. One function, `fmiDoStep`, is defined to perform the next simulation

step. An asynchronous mode is optionally supported. In this mode, the `fmiDoStep` function returns immediately and a callback function may be called on finishing the step. Additionally, the FMI specifies functions to poll the current status of the FMU. If no simulation step is in progress, the FMU variables will be accessed using the getter and setter functions for model variables.

2.2.2 Work Related to Co-Simulation Standards

In order to reduce tool-specific coupling efforts, co-simulation and model exchange standards, such as the FMI, have been created [8, 13, 40, 51]. Many tools which focus on simulation already support co-simulation or model exchange standards and the FMI in particular [33]. Some academic research regarding the FMI standard was already conducted [8, 9, 10, 13, 68, 108]. Müller et al. and Widl et al. studied the FMI-based simulation of discrete-event and continuous systems [68, 108]. They proposed a prediction-based approach to use FMI-based models in discrete-event simulations. In order to detect and schedule upcoming events, the imported model is solved beyond the current instant of simulation time. In case an event is detected during the look-ahead phase, it can be scheduled by the discrete-event simulator. If another, external event is scheduled before the predicted event, the model is reset to a previous state and the event can be applied without introducing any artificial delay. The approach was implemented into an open-source library called FMI++ and tested in Ptolemy II and GridLAB-D.

The FMI++ library itself provides various utilities for two major use cases: importing FMUs into a simulation tool and exporting functionality via the FMI [95]. The import utilities define a set of wrappers which encapsulate the low-level functionality of an FMU and provide high-level functionalities, such as solving the model equations and synchronizing the operation of an FMU. For instance, one wrapper, which targets fixed-step-size simulations, and one, which focuses on discrete-event-based coupling, are available. Additionally, FMI++ contains several utility classes for loading and instantiating FMUs without the need of directly accessing low-level binary or XML files. Similarly, the export utilities of FMI++ provide high-level infrastructure to access a simulation tool via the FMI. An FMI++ back-end service may be incorporated into existing tools and can then be used to expose functionality via a front-end service and the FMI, without the need of directly implementing communication facilities and low-level C functions.

Another co-simulation architecture is defined in the Institute of Electrical and Electronics Engineers (IEEE) standard 1516 which is also called High Level Architecture (HLA) [8, 9, 51, 67]. The HLA defines a federation as set of applications which interact with each other. Object models which define the information produced or required by an application have to be defined in order to enable seamless interaction. In addition, the HLA defines programming language-independent interfaces as well as Web Services Description Language (WSDL), Java and C++ bindings.

Müller et al. introduced an HLA-based co-simulation in the context of smart grids

[67]. They noted that smart grid applications often require data exchange and therefore one needs to include ICT systems in a comprehensive simulation. PowerFactory and OPNET, a network simulator, were successfully integrated. Awais et al. presented several approaches which interlink HLA and FMI [8, 9]. The methods cover fixed time-stepped as well as discrete-event-based HLA components which wrap FMI-based models. The ordinary fixed time step approach involves a time span where HLA components do not generate an event. A variable step size approach may, in general, generate fewer events and increases the simulation performance. The algorithms were evaluated by a simple dynamic electricity market simulation. The authors concluded that the presented approaches are suitable for hybrid simulations but the fixed step size algorithm may suffer from performance issues.

Some frameworks which couple heterogeneous models and tools already exist. Mosaic, for instance, specifically tackles large-scale scenarios in future energy systems [79] via co-simulation. Mosaic allows to efficiently describe large-scale scenarios with a rule-based scenario description language. It implements a time-discrete approach for simulator coupling which specifically targets stationary simulations. Various API and FMI-support are provided to couple external models and tools.

The Ptolemy II framework was created to simulate models which do not follow a single model of computation in a comprehensive simulation [23]. Hierarchical decomposition is applied to create locally homogeneous models which can be jointly analyzed. Some implemented computational domains include discrete event, continuous time, and communicating sequential process models. FMI-support for Ptolemy II was added by Widl et al. who applied their predictive approach [108] and Cremona et al. who built an integrated development environment for FMI-based co-simulations called FIDE on top of Ptolemy II [18]. In contrast to the predictive coupling, FIDE uses code generation instead of the Ptolemy II execution engine to generate a binary which co-simulates the experiment. Although the authors mention the simulation of model exchange FMUs by wrapping them for co-simulation, model exchange specific feature are not considered in detail.

Another framework for joint simulation is the Building Controls Virtual Test Bed (BCVTB) which specifically focuses on applications in the building domain [15, 107]. BCVTB is based on Ptolemy II and uses the software as a middleware for coupling various simulators in a co-simulation. Notably, BCVTB already supports the BACnet protocol and some IO devices for accessing building control systems and external hardware. Although many modeling domains in Ptolemy II and consequently in BCVTB support a soft real-time approach [77], no comprehensive evaluation of the real-time operation is known to the author.

2.2.3 Use of IEC 61499 in Co-Simulation

Several projects which use IEC 61499-based controllers to model control-specific aspects in a co-simulation exist. The deployed coupling strategies range from simple tool integration

setups which neglect the execution time of an IEC 61499-based controller to complex interactions which try to preserve the timing of controllers as accurately as possible. Yang et al. listed several synchronization issues for including a controller in a closed-loop co-simulation [110]. Control output, which is read from the controller, needs to be applied at the correct instance of time. Race conditions and spurious delays may result from insufficient synchronization. The authors proposed and implemented a proxy-based algorithm which delays control output by a fixed amount of simulation time to reflect the execution time of the controller. A circuit protection scheme which was implemented as IEC 61499-based control application was used to evaluate the synchronization approach.

Strasser et al. demonstrated the use of co-simulation in smart grid applications by implementing and simulating an On-Load Tap Changer (OLTC) controller [90]. GNU Octave with the PSAT toolbox was used to simulate the power system. An IEC 61499-based controller which was coupled via a TCP/IP connection implemented the actual control logic. In a training platform for smart grid applications, co-simulation was used to create a comprehensive platform [91]. The PowerFactory simulation tool acts as a simulation master and implements the model of a power distribution network. The automation system is implemented in 4diac and a SCADA system. The IEC 61499-based control logic was accessed via a TCP/IP connection.

Stifter et al. presented a power system co-simulation setup which entirely relies on open-source software [88]. GridLAB-D is used to coordinate the co-simulation and to synchronize all individual tools. Therefore, new GridLAB-D plugins which interface other tools were developed. The acausal modeling concept of OpenModelica is used to model and include a battery. The battery model was exported and included via the FMI standard. In addition, GNU Octave and the PSAT toolbox were accessed via a newly developed wrapper component. As in the training platform, Eclipse 4diac was accessed via a TCP/IP connection and implemented control-specific logic. The whole setup was used to implement a simple controller which avoids voltage violations while charging an electric vehicle. Nikula presented a broker-based co-simulation approach to model heat trade [69]. The process simulation software Apros is used to simulate the heat network and an IEC 61499-based controller implements the trade algorithm. In order to couple both tools, a broker was used. For synchronizing both tools, the coupling software halts the process simulation while the controller executes. A case study on a simulated heat network was performed which assesses the trade algorithm as well as the simulation environment.

A tool which exports the results of FMI-based models to IEC 61499-based controllers was also developed [85, 86]. A prediction-based approach is used to forecast upcoming events which should be sent to the controller. Control actions are triggered in a best-effort real-time approach. Hence, simple co-simulation setups can be realized via the existing tool. Nevertheless, a closed-loop operation was not implemented.

2.3 Hardware in the Loop and Real-time Simulation

HIL simulations denote setups where physically available HuT interacts with simulated models in a closed-loop fashion. Since hardware inherently operates in real time only, the simulation must be executed according to the progress of real time. Other use cases of real-time simulation include co-simulation approaches which implicitly synchronize simulation time via the progress of real time.

2.3.1 Overview

During development and test cycles, HIL simulations extend pure virtual simulation by the ability of testing real hardware. Numerous text books which describe the foundation of closed-loop control and operation of HIL setups are available [20, 21, 55, 62, 63]. In particular, Viehweider et al. studied stability issues in power HIL simulations of electric energy systems [101]. They noted that stability of these simulations is a necessity, accuracy a sufficient condition. In the studied scenario, a power system model was coupled with a real load, forming a closed-loop. A virtual current source mimics the real load in the model and a real voltage source mirrors the output voltage of the model. Three methods which improve the system stability were proposed. The first one adds a hardware inductance in series to the real load. The inductance increases the phase reserve but suffers from poor accuracy. The second method deploys feedback current filtering, and the third one uses multi-rate partitioning to increase the sampling frequency of the hardware coupling subsystem. It turned out that multi-rate partition works best with respect to stability and accuracy.

Guo et al. conducted a comprehensive smart grid simulation of a large number of switching devices [43]. The simulation was executed in real time but no HuT was included. A case study was performed which includes a small community microgrid which is controlled via a control center. The control center communicates with a circuit breaker via an IEEE 802.11 network. In particular, the transformation from grid to islanding mode was simulated. Four distinct real-time simulators are used to execute the electrical model and a desktop computer runs the network simulator. The authors concluded that communication networks enable new protection schemes but network simulation is necessary to predict the behavior of the system accurately.

Andrén et al. described the development and validation process of a coordinated voltage controller which maintains adequate voltages in a medium voltage grid [3]. The presented coordinated voltage controller governs the production of active and reactive power at multiple Distributed Energy Resources (DERs) as well as the tap position of a transformer in order to maintain admissible voltage levels all over the grid. Hence, the coordinated voltage controller introduces much more complexity than a local voltage controller which operates directly at a single DER. Controller HIL tests were presented as a vital step towards a successful deployment of the newly developed controller. An industrial communication protocol was used to couple the controller implementation and the electricity grid simulation. The authors concluded that the controller HIL simulations

allowed to test the stability of the system and the influence of process dynamics, cycle times, and communication delay.

2.3.2 Standard-based Hardware in the Loop Simulation

A first step towards standard-based HIL simulation was presented by Pang et al. who coupled a building energy simulation and a real building to compare projected and measured energy consumption [75]. A Ptolemy II-based software and a communication tool were used to couple building simulation tools via FMI and the building control hardware. Weather data was taken as an input to predict the actual energy consumption. Some notable differences between the calculated output and the measured energy consumption were detected.

Some commercial FMI-enabled tools which target co-simulation and HIL simulations are already available [16, 17, 26, 33, 34, 42, 57, 66, 84, 98, 112]. Many tools focus on the automotive domain but also an industrial automation tool which allows HIL simulations was presented. Zehetner et al. developed a methodology for bridging hard real-time systems and non-real-time simulations in a combined co-simulation [66, 112]. Delays between components were identified as one of the main issues in co-simulation which strongly influence stability and accuracy in a closed-loop operation. The whole system is partitioned into a real-time capable part and the non-real-time simulations. In order to reduce occurring delays, coupling elements, which perform low-order polynomial extrapolation and disrupt the closed loop, were inserted. The methodology was tested on a combustion engine test stand and a test stand for automotive electronic control units. Less overshoot of the controlled signals was observed when using the latency compensation compared to standard coupling.

Several commercial co-simulation environments use well-established industrial communication interfaces to connect external hardware [57, 84, 98]. TISC Suite, for instance, interfaces metering systems and test stands via CAN or LIN protocol gateways [98]. Additional interfaces, such as LabView, Simatic S7 PLCSIM and FMI, are also listed to be available. Also SIMulation Workbench, a model environment for real-time HIL simulations, lists several available hardware interfaces, such as CAN, FlexRay, EtherCAT, serial-, and IO-Lines [84]. Some co-simulation tools, such as CarMaker, which targets virtual test driving of cars, aim for integration in dedicated real-time simulation hardware for HIL setups [16, 34, 98].

First FMI support in commercial automation-related products is already available [17, 26, 33, 42]. Dassault Systèmes offers an automation platform which specifically targets safety-critical systems [17]. The platform, which is called ControlBuild, can be used to model, simulate, test, and validate IEC 61131-based control applications. It is listed to support most variants of the FMI interface [33]. B&R Automation Studio, which allows to develop ANSI C and IEC 61131-based control applications, was also reported to support FMI for co-simulation [26, 33, 42]. Gunnarsson and Erwall et al. evaluated the FMI integration based on a reaction wheel pendulum [26, 42]. A controller was

exported via Automation Studio Target for Simulink and the model of the pendulum was exported via the FMI. The model was created via Dymola and MapleSim. Several solver configurations and their impact on the simulation result were studied. A co-simulation in Automation Studio was performed first. A controller HIL simulation on two dedicated PLCs, which were connected via EtherCAT, was also conducted. One PLC executed the controller while the other one simulated the exported plant model. It turned out that all HIL simulations using Automation Studio are feasible and chosen cycle times strongly influence the simulation outcome.

Although several HIL tools which support the FMI are already available, event-based systems and IEC 61499-based infrastructure received only little attention. As described above, first attempts were made by exporting results to IEC 61499-based controllers via the FMI [85, 86]. Nevertheless, to the author's knowledge, comprehensive studies of IEC 61499 and event-based integration in HIL simulations have not yet been conducted.

2.4 Contribution

This thesis first presents a comprehensive study on linking event-based automation infrastructure following IEC 61499 with FMI-based models or tools. While first considerations on using FMI for model exchange in IEC 61499-based controllers are already included in previous work [85], this thesis additionally covers the FMI for co-simulation as well as the encapsulation of IEC 61499-based facilities into FMUs. The basic operation of both, predictive and periodic synchronization in coupling FMI for model exchange and IEC 61499 is also presented in [85]. Preliminary work of the author which presents a generalized system architecture, considerations on event handling and expected real-time properties on including FMI-based models in IEC 61499-based applications was published [86]. In order to be able to present a comprehensive interaction study, some findings of [86] are included in the extended study as well. This thesis substantially extends preliminary results by including aspects such as automatic data model transformation and by providing a qualitative comparison of all discussed coupling approaches.

In the course of this thesis, a preliminary prototype which exports simulation results of FMI-based models via predictive event synchronization was extended towards a full-fledged closed-loop coupling tool. In particular, a proper handling of border cases such as late and concurrent events, refined configuration options such as configurable numerical integration, as well as support for periodic synchronization were added. The extended tool including its documentation is released as open-source software [35]. In contrast to existing tools which couple hardware, automation systems and FMI-based models or simulators, the extended interface tool specifically focuses on event-based execution and consequently allows to extensively exploit the capabilities of IEC 61499.

Previous evaluation of the described interface software does include hardware and pure software setups, but none of them demonstrated a closed-loop operation. This thesis extends previous results by using the predictive coupling approach in a closed-loop HIL simulation. A detailed analysis, which also covers deviation sources on a quantitative

and qualitative level, is presented and results of predictive synchronization are compared to results of conventional periodic synchronization. It is demonstrated that periodic synchronization shows systematic deficits on systems which require immediate action on executing events and that predictive synchronization can be successfully used to overcome these limitations.

CHAPTER 3

Interaction Study

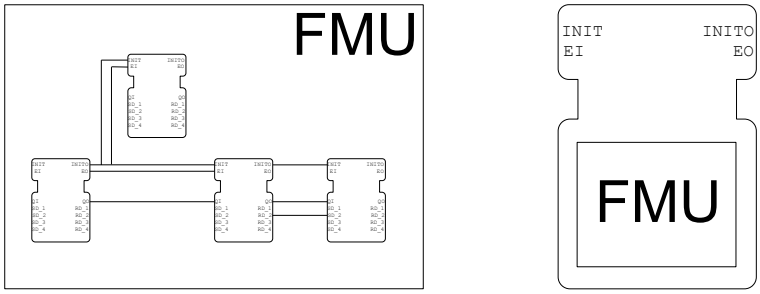
3.1 Preliminary Discussion

Before the various coupling approaches are discussed in detail, an abstract structure is introduced which guides the following considerations. Section 3.1.1 presents a coarse but complete decomposition of possible links between IEC standard 61499 and the FMI standards. Section 3.1.3 refines the initial decomposition such that each identified feature of Section 3.1.2 can be studied in detail.

3.1.1 Principal Ways of Interaction

Based on the design of the FMI described in Section 2.2.1, the FMI can either be used actively by instantiating and including existing FMUs within an application or passively by providing functions used to include the application as an FMU. To differentiate both scenarios, the first variant is called active use and the second one passive use of the FMI. Both cases, the active and passive use are illustrated in Figure 3.1. Note that each scenario may require additional interface components such as ODE solvers which are omitted in Figure 3.1. A manifold set of applications may utilize the FMI actively or passively, not restricted to control applications and HIL setups [33]. An application may even implement passive and active use simultaneously [39, 40]. Nevertheless, each implementation can be clearly separated and will therefore be discussed separately. Although the FMI may be deployed in a broad range of applications, the following sections focus on IEC 61499-based controls and applications in the automation domain in general.

By active use, an application is able to determine major control flow and timing parameters such as simulation time progress and FMU invocation. The component actively including the FMUs also has to implement certain features required by the FMI, including an ODE solver for model exchange and a master algorithm for co-simulation, respectively



(a) Encapsulating an IEC 61499 application into an FMU

(b) Using an FMU in an IEC 61499 application

Figure 3.1: Principal ways of coupling IEC61499 and the FMI

[38, 39, 40]. Many different algorithms solving ODEs or coordinating co-simulations already exist, but depending on the use case some of them might be inappropriate [21, 38]. As a consequence, a component actively using an FMU might have to implement a set of different algorithms or may suffer from poor performance. The implementation of multiple algorithms increases the implementation effort and the costs of the solution. On the other hand, the implemented ODE solver or master algorithm can be easily tailored to the needs of the application.

Passive use generally eases the use of existing code by outsourcing relevant parts to the co-simulation or model exchange environment. Although this scenario enables another set of use cases, especially in complex simulations, an FMU generally cannot take any assumptions about the operation of the master, except those defined in the FMI.

3.1.2 Important Features in Combining IEC 61499 and the FMI

Depending on the actual use case, different parameters and aspects in combining the IEC 61499 and the FMI become relevant. When using a set of IEC 61499-based controllers within a non-real-time grid simulation, it is relevant to model time progress in terms of current simulation time [110]. A delay introduced by the interface is not noticed as long as the current simulation time is not affected. Within an HIL simulation setup, real-time parameters like delays or the sample rates are important factors to gain stability and accuracy [101]. The optimal solution, relevant parameters and features highly depend on the specific use case. To address different needs, a set of possibly relevant features and aspects is defined. The following sections explain and justify the criteria used to discuss different ways of combining the FMI and the IEC 61499.

System Architecture and Model

Depending on the way the IEC 61499 interacts with the FMI components, different system architectures and views can be used to describe the resulting system. This includes

possible component models dividing the system into adequate subsystems as well as different mathematical techniques applicable to describe the system. A specific point of interaction may allow various representations and views. For example, using an IEC 61499 controller in a co-simulation of some other components can be seen as coupling multiple different, equally stated components or as hierarchically integrating the IEC 61499 controller into the co-simulation framework. When building a hierarchy, the controller is part of the joint simulation which is governed by the framework. Although both views will result in a similar implementation, they will affect the degree of understanding the behavior of the coupled system [55, p.29ff]. It is also shown that the structure of the model does not only affect the degree of understanding, but also influences the emergence of spurious simulation artifacts [23].

Additionally, the system architecture affects the level of integration and the envisioned coupling workflow. Loose coupling of independent components features a low level of integration and a more flexible workflow. Contrary, a tight integration into a specific co-simulation tool may enable a higher degree of automatic information exchange and a reduced coupling effort for that specific tool.

Provided system architectures may not only include the components which are directly interacting with the FMI and the IEC 61499 controllers but also additional hard- and software. For example, in an HIL setup, the HuT affects the overall system behavior and has to be considered in the system architecture as well. Depending on the specific use case, the system architecture may have to be adapted. In the context of this thesis, it is not possible to state every feasible system model and view, but by explicitly describing used models, the discussion of other features should be eased.

Event Handling

The IEC standard 61499 and the FMI use different definitions of the term event. On the one hand, an event in the context of the IEC 61499 describes an instantaneous occurrence primarily used to schedule the execution of algorithms [50, p.14]. On the other hand, FMI events indicate possible discontinuities between otherwise continuous signals [39, p.6]. It is necessary to map FMI events to events triggered within an IEC 61499 application to enable proper interaction. In case of continuous FMI variables, the FMU does not trigger any FMI events. To communicate continuous results from an FMU, additional measures will be necessary which eventually trigger the execution of IEC 61499 components.

Depending on the concept of interaction, events may be mapped differently and triggered at different instants of time. For example, on the one hand, a strictly periodic design may be applied which communicates values at fixed instants of time only. On the other hand, it is also conceivable to employ a variable step size that triggers IEC 61499 events only on demand. It may also be beneficial to delay certain events in order to meet other requirements like determinism or real-time operation. As a key feature, the provided concepts will define and discuss the event handling mechanism based on the questions of

mapping the event types, mapping continuous changes within the FMI to discrete events used in IEC 61499 and delaying events.

Data Model Transformation

Both standards define their own representation of data and metadata. FMI defines five scalar data types (including enumerations) [38, 39, 40] and IEC 61499 implicitly refers to the type system of IEC 61131 to define variable types [50, p.68]. Types from both standards need to be mapped in order to establish coherent data exchange. Furthermore, an FMU may encapsulate static variable descriptions such as unit and derived type definitions which need further investigation. Similarly, structural descriptions of IEC 61499-based systems and FMI-based models and tools may also be transformed.

Ideally, a point of interaction maps as much information as reasonably possible. Automatic transformation is preferred over manual intervention. Aspects regarding data model transformation also cover ways and strategies to transform both data representations. Significant data losses and required manual configuration of each transformation are highlighted in the discussion. Additionally, tool integration and user assistance in configuring a point of interaction are briefly discussed. Due to the limited scope of the thesis, no comprehensive formal description, e.g. in terms of Extensible Stylesheet Language Transformations (XSLTs), can be given. Instead, the discussion focuses on the main aspects of data model transformation only.

Real-time Operation

As stated before, real-time parameters like delays or sample rates have a significant impact on simulations running in real time, especially for HIL setups [101]. The definition of real-time systems used in this thesis is based on the definition provided by Kopetz:

> A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced. By system behavior we mean the sequence of outputs in time of a system. [55, p.2]

Real-time systems can be divided into soft and hard real-time systems [55, p.13ff]. In a hard real-time system, missing a deadline may have fatal consequences such as harming humans or losing expensive equipment. Hard real-time systems can be found in fly-by-wire or electrical energy systems mostly relying on a timely operation. In contrast, soft real-time systems may tolerate deadline misses without major consequences but with a degraded functionality. Within a common multimedia system, for example, deadline misses appear as interruptions and noisy sounds but do not have any major consequences.

Design principles applied to soft and hard real-time systems are fundamentally different [55, p.13ff]. Performance in hard real-time systems, on the one hand, has to be guaranteed even in peak load scenarios. In this case, sound assumptions have to be made and the

peak load scenario has to be well-defined in order to design a system that can function even under rare circumstances. The design of a soft real-time system, on the other hand, mostly focuses on the average performance without taking rare scenarios into account. Also, error detection and recovery may be user-assisted and not fully automated.

In order to study real-time parameters, we will at first discuss whether a specific point of interaction can provide response-time guarantees and which assumptions have to be taken to guarantee a response time. Response time in this context is defined as the maximum difference of the two instants in real time t_{ei} and t_{eo} where t_{ei} corresponds to the incoming and t_{eo} to the resulting outgoing event according to the system borders. Additionally, timely accuracies and the possibility of preserving the event order will be discussed.

Model and Tool Coupling

The FMI was designed to allow coupling of multiple tools or models in a modeling framework [38, 39]. Also the IEC 61499 arose from the need of configuring distributed automation systems covering several units [50]. Unlike the FMI, the IEC 61499 defines not only the interfaces of components but also a way to couple components. Within a framework that couples IEC 61499-based controllers and FMUs, different ways of coupling multiple FMUs and multiple IEC 61499 applications are conceivable. For example, two model exchange FMUs may be coupled within a single solver or by connecting different FBs wrapping the FMUs.

Each point of interaction will be evaluated based on the possibility of coupling different FMUs and applications. For each point of interaction, various ways may be stated and the limitations will be discussed. The potential of handling algebraic loops, described in Section 2.2.1, as well as further information needed to couple applications and FMUs will also be discussed.

Software Interfaces

The IEC 61499 interface considerations include various software implementation and configuration aspects related to IEC 61499-based systems. To exchange information with connected FMUs or with simulation tools, proper interfaces within IEC 61499 configurations have to be defined. Interfaces may strongly vary by their implementation effort and by assumptions concerning the execution platform. Some software components like ODE solvers or communication protocol implementation may already be available and may ease implementation.

Also, when passively providing or actively using the FMI, available software may be taken into consideration. Relevant software and software components include XML parsing and writing libraries, available FMI libraries as well as ODE solvers and co-simulation frameworks. In interfacing FMUs, the sequence of allowed function calls has to be checked against the interfacing algorithm in order to avoid illegal operation of FMUs. Each FMI interface specifies a state machine and enabled function calls in each state. For example,

increasing the time within a model exchange FMI event state would violate the FMI specification [39, p.22].

Use Cases

The type of interaction as well as other parameters will strongly affect the use cases of the interaction. Real-time operation, for example, usually cannot tolerate long simulation periods which would consume too much time, but it enables the integration of components which operate in real time only. Also, the way of implementing the FMI restricts certain use cases. The FMI can either be implemented as a passive component encapsulated within an FMU or as a tool actively using FMUs, depending on the specific use case.

Within the following sections, some possible use cases of different integration points will be discussed. However, it has to be noted that the integration points are not restricted to the presented use cases. They are intended to inspire different applications and to describe various benefits and drawbacks. Within the theoretical discussion, the use cases will focus on smart-grid-related scenarios, mostly in the context of testing grid components and simulating certain aspects of a smart grid.

3.1.3 Structure of the Discussion

The inherent differences between model exchange and co-simulation FMI do not allow a structured discussion of most aspects without differentiating between the interfaces. For example, the passive provision of a model exchange FMI requires state derivatives and event indicators which are not communicated by the FMI for co-simulation [38, 39]. Although the second version of the FMI merges both standards, it also distinguishes between model exchange and co-simulation applications [40]. This thesis, too, will separate the two FMI standards and the ways of using them, resulting in four principal ways of interaction to be discussed. Sections 3.2 and 3.3 describe the active use of model exchange and co-simulation FMUs within IEC 61499 controllers, respectively. Similarly, Section 3.4 and 3.5 describe the passive use of the FMI by encapsulating IEC 61499 applications in FMUs. Finally, Section 3.6 compares the different approaches with each other.

3.2 Using Model Exchange FMUs in IEC 61499 Applications

3.2.1 System Architecture and Model

Model exchange FMUs require the including tool to provide a numerical ODE solver [86]. The IEC 61499 standard does not speak about solving any equations directly [50, 86] but addresses computational issues on an algorithmic level. For instance, high-level subjects such as solving ODEs are not directly addressed. Instead, the standard defines ways to schedule generic program parts, called algorithms. An ODE solver has to be provided in

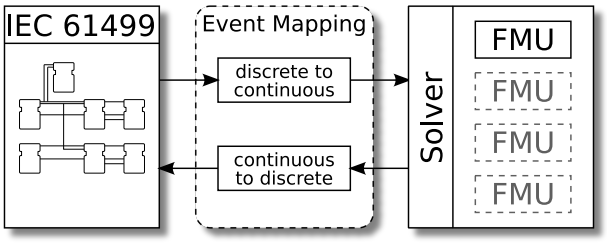


Figure 3.2: Basic abstract model exchange system model [86]

order to combine both levels of abstraction. Since model exchange FMI and IEC 61499 also use different event semantics, it is necessary to convert between both interpretations.

Within the presented setup, the simulation time of an FMU has to be controlled by explicitly setting the current instant of time [86]. In case the implemented solver increases the simulation time without any synchronization, results will be pretty meaningless, because the outputs from an IEC 61499 application and an FMU generally depend on the current instant of time. As a consequence, time synchronization has to be considered in the system design and either an IEC 61499 application or the interface component has to actively coordinate simulation time progress.

Figure 3.2 shows a first abstract system model which includes one or more FMUs, the solver, the time and event conversion logic, and the actual IEC 61499 application. In the abstract system model, the application is represented as a single component without using any IEC 61499-specific points of integration. Hence, the system can be represented without making excessive restrictions and a first impression of the architecture is given. An actual implementation will have to refine the abstract model according to the chosen IEC 61499-specific points of integration. Depending on the actual components' implementation, mathematical modeling of the system may vary, but in general the FMUs depend on the application output and vice versa, resulting in a closed-loop system.

It is important to notice that the solver which solves the ODEs is not part of the FMUs but resides in the interface logic. Consequently, the operation of a solver can be tailored to specific needs and it is not necessary to assume a generic solver such as the one in FMI fo co-simulation. For some implementations, it is beneficial to conceive the interface logic as non-standard part of an IEC 61499-based controller, but to enable a differentiated and generalized discussion, the presented architecture explicitly states various interface components which are needed to include FMUs into an IEC 61499-based application. Depending on the actual implementation, the interface components may, for example, be directly part of an IEC 61499-based application or a middleware which bridges both standards. Section 3.4.6 discusses implementation-related aspects in more detail.

An IEC 61499 application may not only interface FMI-based components but may also connect to other components using a continuous time domain. For example, by connecting a HuT to an IEC 61499-based controller, two different subsystems generally acting in

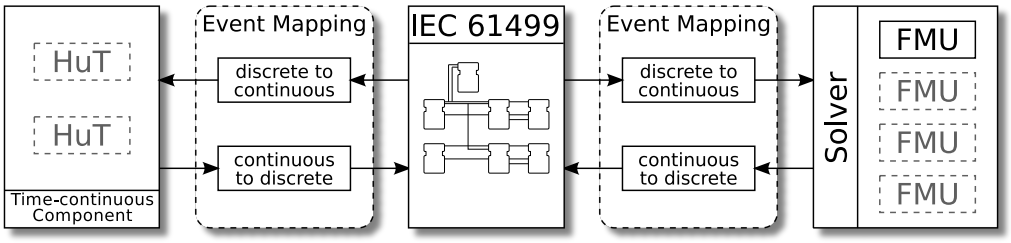


Figure 3.3: Extended abstract model exchange system model

a continuous time domain interact by means of the discrete-event-based controller. Figure 3.3 shows a possible system setup including the FMI and another abstract component interfacing with the IEC 61499 application. Even if the IEC 61499 application does not change any value and directly passes incoming events on, the execution time of the application [101] as well as the synchronization of different clocks [55] have to be considered. Depending on timing parameters and implementation details of the IEC 61499 application, the model of it may be simplified to an identity function directly passing incoming events on.

3.2.2 Event Handling

Since IEC 61499 events can be associated with in- and output variables, the event handling mechanism has to coordinate data and event exchange [86]. In particular, it has to set the input data of the FMUs and has to pass IEC 61499 events on to the FMUs. Additionally, it has to process the output data generated by simulation to trigger IEC 61499 events for further execution. In the following discussions, it is assumed that values associated with IEC 61499 events do not change between two consecutive events. Consequently, the semantics of IEC 61499 events are adapted to FMI event semantics, where events mark potential discontinuities. From an FMI perspective, an IEC 61499-based controller, which changes values between events, must either change the values continuously or an intermediate FMI event has to be triggered. Within a system operating with time- and value-discrete entities only, continuous changes are not possible and intermediate FMI events do not allow a one-to-one mapping of FMI events and IEC 61499 events. Consequently, the constant value assumption enables a one-to-one event mapping and eases event handling. Note, that FBs may still change the event data during their execution, but values will not be communicated unless a corresponding event is triggered. Since an outside entity wont notice intermediate changes, it can further be assumed that values only exist at instants of time when an IEC 61499 event is triggered.

According to Lunze, an interpolation function has to be used to map the discrete-time domain of the IEC 61499 to the continuous-time domain featured by the FMI [62, p.418]. Zero-Order Hold (ZOH) is one of the most widely deployed interpolation functions. The function holds values associated with one discrete-event instance constant until the next event instance occurs. Other interpolation functions, which may also be applicable,

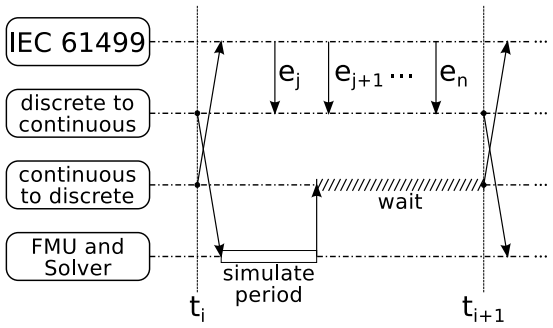


Figure 3.4: Periodic event mapping [86]

include higher-order holding elements. Nevertheless, this thesis will silently assume a ZOH interpolation unless another function is explicitly stated.

Periodic Event Mapping

First, a very simple periodic approach for synchronizing the operation of an FMU and an IEC 61499 application [86] is presented. Communication between the IEC 61499 application and the solver is restricted to discrete communication points $t_i = i \cdot T_\mathrm{a}$ only, where $T_\mathrm{a} \in \mathbb{R}, T_\mathrm{a} > 0$ corresponds to the constant step size and $i \in \mathbb{N}$ to the number of the current sample. Note that although no direct communication of the IEC 61499 application and the FMUs is permitted at $t \neq i \cdot T_\mathrm{a}, i \in \mathbb{N}$, a solver may access the FMUs at arbitrary instances of time to get accurate numerical solutions. The sequence of events and the processes of periodic event mapping is drawn in Figure 3.4. The graphic follows an adapted syntax which is loosely based on UML 2.3 sequence diagrams [81, p.209ff]. The current progress in time is visualized by the horizontal axis and individual components are arranged at the vertical axis. Interactions between different components are represented by one arrow, each.

The strictly periodic operation with fixed communication points is very similar to the operation specified by the FMI for co-simulation. An IEC 61499 event will be generated by the continuous-to-discrete event mapping component as soon as a synchronization point t_i is reached. In contrast to FMI for co-simulation, communication between the solver and the FMUs via the FMI is not restricted to predefined communication points. Hence, arbitrary intermediate steps may be taken. In particular, the solver will locally process any FMI event between two consecutive communication points t_i and t_{i+1} and no IEC 61499 event will be generated.

In the other direction, IEC 61499 events $e_j, j \in \mathbb{N}$ which occur between two consecutive synchronization points t_i and t_{i+1} and address the FMUs are delayed until t_{i+1}. According to the abstract system model which is presented in Figure 3.2, it is assumed that one single discrete-to-continuous event mapping component receives IEC 61499 events and associated data atomically and in a well-defined order, i.e. event data is sampled before

any other event arrives and can be clearly associated with a corresponding event. In IEC 61499-based applications, variable values may depend on the execution order of connected FBs. The atomicity assumption relaxes the discussion and allows to focus on coupling-related aspects. Any behavior which emerges from a varying schedule is considered as part of the IEC 61499 application and will not be discussed in detail.

According to the syntax described by Kopetz [55], $t(e_j)$ returns the instant of global time when event e_j happens. Additionally, $\mathbb{V}(e_j)$ corresponds to the set of variables associated with event e_j at the instant of time $t(e_j)$. To accurately reflect the system state at t_{i+1}, it is necessary in general to delay events $e_j, t_i \leq t(e_j) < t_{i+1}$ according to their time of occurrence $t(e_j)$. Note, that it is assumed that an event e_j cannot be simultaneously processed with a communication point and therefore, an event with $t(e_j) = t_i$ is, by definition, delayed to t_{i+1}. Assume both events e_j and e_{j+1} are issued between t_i and t_{i+1}. By the atomicity assumption of incoming events, the variable image after applying both events depends on the event order if and only if e_j and e_{j+1} share common variables $\mathbb{V}_{\text{common}} = \mathbb{V}(e_j) \cap \mathbb{V}(e_{j+1}) \neq \emptyset$. More precisely, the notion of variables which are communicated by both events, $\mathbb{V}_{\text{common}}$, will depend on the event order. When applying e_{j+1} before e_j, $\mathbb{V}_{\text{common}}$ will contain the system state at $t(e_j)$ instead of t_{i+1}. Hence, the discrete-to-continuous event mapping component has to preserve the event order, or use latest information only, to avoid faulty communication.

Within a single process application, the order of events may be preserved by enumerating incoming events. By the single process assumption, it is not possible that another incoming event is passed on to the discrete-to-continuous mapping component unless the operation processing the previous one is completed. Still, the atomicity assumption is necessary to guarantee that variables can be clearly associated with a certain event. However, the IEC 61499 defines a highly distributed architecture separating different scheduling domains [50] where the assumption of strictly sequential scheduling does not hold. Within a distributed environment, for example, different network and transport delays have to be considered. It is easily possible that an event generated after another event may arrive prior to the earlier one due to different communication times. Since a process, when mapping discrete events to a continuous state, sorts events based on the time of arrival, the system state may not be reconstructed properly.

Even by extending IEC 61499 events by time stamps, a proper reordering of events is not possible in general [55]. According to the model described by Kopetz, a global discrete reference clock z is introduced, generating periodic events, called ticks. Additionally, each device c^k maintains a discrete clock running with a granularity g^k expressed in number of nominal ticks of the reference clock z. Each tick will be enumerated, constructing a temporal order between ticks of a single clock. The function $z(microtick_i^k)$ corresponds to the i-th tick of clock c^k, called microtick. The precision Π_i of a system is defined in (3.1) as maximum difference of any two clocks in the ensemble of n clocks at a given microtick i.

$$\Pi_i = \max_{\forall j,k:1 \leq j,k \leq n} (|z(microtick_i^k) - z(microtick_i^j)|) \tag{3.1}$$

The maximum of Π_i within a defined interval of interest is called precision Π of an ensemble of clocks. A global time maintained by different clocks can be obtained by selecting a subset of microticks from each clock called macroticks [55]. A global time base is called reasonable if its granularity g exceeds the precision: $g > \Pi$. If the global time base is reasonable, the macroticks recorded by two devices observing the same event differ by at most one. Events observed by different devices may be reordered if and only if their time stamps differ by at least two macroticks. Any difference below two macroticks can be traced back to unavoidable deviations of the controller clocks. As a consequence, whenever two events are triggered too soon after each other, it is not possible to reconstruct their order.

The IEC 61499 itself does not specify any means to control the time when events are triggered [50]. For instance, IEC 61499 events do not have any associated time stamp which controls the time at which an event is triggered. Furthermore, physical systems may be connected to the IEC 61499 application, making it impossible to restrict the time, an event is fired. For example, two independent mechanical switches may be activated shortly one after another and the resulting events may be time-stamped equally. As a result, the temporal order of any two events occurring within the IEC 61499 application generally cannot be reconstructed, even if the event time is recorded. Even worse, ordering may be performed differently by any two independent processes [55]. If the system output relies on a consistent view of the state, an agreement protocol has to be used. This is the case if multiple entities, two unsynchronized models for example, use the same input variables to calculate an output. Although the agreement protocol cannot re-establish the temporal order of the events, it guarantees consistent data.

The restrictions on ordering events recorded at different clock domains is inherent in distributed systems and also IEC 61499 applications not using the FMI suffer from it. As stated before, a possible way of avoiding these issues within an FMI interfacing component is to restrict the discrete-to-continuous event mapping part to a single process. In this setup, it is still possible that IEC 61499 components trigger unordered events but the connected FMUs will gain a consistent view. Some applications like non-real-time co-simulations may additionally provide a common time base allowing to reorder the events accurately.

A major drawback of the periodic event mapping approach is the need for event aggregation between different synchronization points t_i [86]. The accuracy of the approach depends highly on the condition of included models and the time difference between two consecutive sample points. In case the synchronization step size T_a is chosen too coarse, synchronization may lead to significant errors. In particular, if a strong dependency of triggered events is encountered, the sampling period needs to be carefully chosen. For instance, an FMU output will be delayed by one entire synchronization period T_a, if the result directly depends on its inputs and an incoming event changes at least one of the inputs.

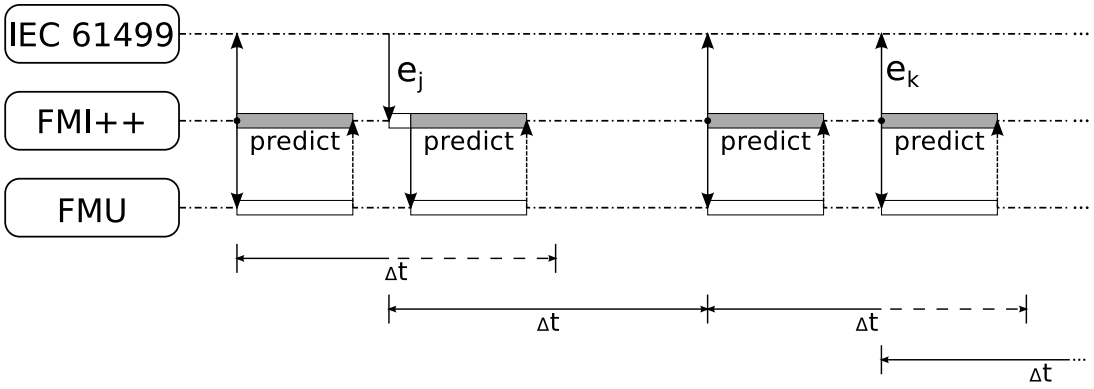


Figure 3.5: Prediction-based event mapping using the FMI for model exchange [86]: External event arrival (left), uninterrupted operation (middle) and FMU-internal event occurrence (right)

Prediction-based Event Mapping

Müller et al. and Widl et al. proposed a state prediction-based approach that approximates future states and events [68, 108]. The approach was implemented in an open-source library, called FMI++, which allows to couple discrete-event-based systems and FMUs for model exchange. The main idea of the approach is to predict FMI events in advance, such that these events can be accurately scheduled by the discrete-event-based system. For event prediction, they solve the FMU in advance and locally store intermediate states at equidistant nodes until the end of the prediction period, called lookahead horizon Δt, is reached or an event is detected. In both cases, an event is passed on to the simulation framework in advance, which then can coordinate the execution accordingly. On receiving a scheduled event by the discrete-event simulation, future predicted results may become invalid, because the event may change one of the FMU inputs. In case of an incoming event, the time of the FMU is set to the time of the event and the state of the FMU is interpolated from the stored nodes. Subsequently, prediction is repeated with the newly set inputs.

Although upcoming events triggered by FMI++ will be known in advance, there is no mechanism in IEC 61499 that directly time-stamps events and registers future events, which should be scheduled at a particular point in time [50, 86]. Consequently, events registered in advance have to be delayed by the event mapping component until the simulation or real time proceeds to the predicted event time. The sequence of interactions for external event arrival (left), uninterrupted operation (middle) and FMU-internal event occurrence (right) is illustrated in Figure 3.5. According to the general predictive event mapping approach, an event which is triggered by the IEC 61499 application, causes the FMI++ interface to re-calculate future states. Such a prediction update may invalidate previous predictions and requires the event mapping component to remove previously scheduled events. To synchronize the operation in case no FMI event was encountered

in the lookahead horizon Δt, an artificial event is triggered at the end of the lookahead horizon which itself triggers the next prediction and outputs continuous FMI variables.

Some algorithms implemented in the IEC 61499 domain need to be scheduled immediately if an input value changes. In addition to the proposed approach of triggering IEC 61499 events at the end of the lookahead horizon, an output deviation-based approach may be used. In this case, the lookahead horizon depends on the output values of the FMU. If the deviation between the predicted output value and the output value communicated last exceeds a certain threshold, an additional IEC 61499 event will be triggered. As stated by Müller et al., a maximum lookahead period should still be used to limit performance losses based on rejected values [68]. First experiments show the best performance in choosing a lookahead horizon equal to the predicted duration between two consecutive events which are triggered by the event-based framework.

Whereas events need to be delayed to the next synchronization point in case of a periodic approach, prediction-based event mapping allows to process events immediately [86]. The reduction of errors, which are introduced by delayed processing of events, comes with the requirement of rejecting results beyond the time of IEC 61499 events issued to the FMU [68, 108]. Such a rejection implies that the state of the FMU has to be reset to a previous point in time. The FMI 1.0 for model exchange exposes continuous states which are manipulated by the solver but does not define function for accessing discrete states [39, 68]. According to the specification, discrete state may not change between two consecutive FMI events. If an FMI event occurs during prediction, the discrete state will not be able to change to previous values. Especially when issuing step events during the prediction phase, the discrete state may change regularly. Since the FMI supports variable step size integration, step events may be delayed to the end of the actually taken time period. Every intermediate integration step calculated during the prediction may be seen as a sub-step not requiring a step event. Depending on the implementation of the FMU, the reduction of step events may avoid non-resettable state changes but may introduce inaccuracies due to less frequent step events.

In FMI 2.0, optional functions for retrieving and setting the entire model state, which includes discrete and internal states, are defined [40]. It will be possible to fully support the prediction-based approach if an FMU implements these state access functions [86]. In case the optional feature is not supported or on using the first version of the FMI, the application cannot predict values beyond the first FMI event.

3.2.3 Data Model Transformation

In order to discuss data model transformations, it is assumed that the included FMU is represented via a single FB. The FB hides implementation details and may be an SIFB, a BFB which implements the C functions and the solver, or a CFB which combines a variety of FMU-related FBs. In any case, it is assumed that the FMU FB type including boilerplate code may be automatically generated by the interface facilities and represents

a virtual instance of the modeled entity. A detailed description of feasible software interfaces follows in Section 3.2.6 which also justifies the assumption.

FMI model variables which possibly include parameters, in- and outputs may be accessed via the input and output variables of the FB. FMI specifies a data type system which differs from the type system of IEC 61499/IEC 61131 [39, 40, 50]. IEC 61499 references IEC 61131 for defining data types. The later standard defines a hierarchical representation of elementary data types and facilities to express user-defined data types [19, p.37ff]. For instance, an elementary data type `REAL` is also a general data type of `ANY_REAL`, `ANY_NUM`, `ANY_MAGNITUDE`, `ANY_ELEMENTARY`, and lastly `ANY`. The type hierarchy allows to define an FB type without specifying the elementary data type in detail. Furthermore, user-defined data types may be expressed. Such data types may restrict the range of an elementary data type or may define enumerations, references, arrays, and structures. FMI also defines means for specifying user-defined data types. In contrast to IEC 61499/IEC 61131, user-defined data types are non-structured only and are mapped to four orthogonal C types [39, 40]. For the types real, integer, and enumeration, the value range of derived variables may be restricted. An optional variable naming convention maps structured data types to a set of elementary model variables. In contrast to IEC 61499, no reference type model variables are defined in the FMI. Similarly, IEC 61499 does not define any means to directly handle unit definitions.

A model transformation scheme has to map FMI model variable types to IEC 61499/ IEC 61131 types and vice versa. Since several elementary types, such as a two byte integer from the automation domain, are not available in the FMI, elementary types may have to be casted. An FMI FB may either restrict automation data types for its in- and outputs or perform type casting itself. In case in- and output types are restricted, type conversion needs to take place in the IEC 61499 domain. In order to utilize user-defined FMI types, a component which includes FMI models in IEC 61499-based applications may automatically create a user-defined IEC 61131 type for each user-defined FMI type. Since IEC 61131 also defines range restrictions, minimum and maximum values may also be transformed without data loss. The FMI unit system which includes base and display units cannot be directly transformed to the automation domain. Although it may be possible to make the information available, e.g. via structures or type naming conventions, an automation type may not be directly amended by its unit. Hence, no guarantees regarding units can be given.

Special attention has to be put on reference and general IEC 61131 data types. Both types are not directly supported by the FMI. An interface component must (de-)reference or disallow any reference type. Furthermore, dynamic type casting or static type inference must be applied to handle general types from automation components. In case the FMI FB is automatically generated from the model description, it simply may not use any types which are not known by the FMI and type conversion is up to the IEC 61499-based application. In case the FMI model description uses the defined naming convention, a mapping component may also define and transform structured and array data types. Each element of the structured data type will have to be mapped to a single model

variable but the structural information from the FMI may be preserved.

In order to indicate updated values, an FMI FB needs to provide event in- and outputs which are associated with a certain set of variables. One basic scheme is to provide one initialization event in- and output each which handles model parameters and static data. Another pair of event in- and outputs may be used to indicate variable changes during run-time. In case one event variable is associated with all dynamic FMI model variables, all variables need to be processed and caching of variables may not be fully exploited. The system performance may be optimized by manually introducing groups of variables which share a single event indicator. Since an FMU does not expose information regarding caching mechanisms and logical semantic grouping of variables, an automatic mapping from variables to event in- and outputs is limited to trivial cases.

Although a direct transformation of model equations into algorithms may be feasible, the transformation highly depends on the capabilities of the RTI in interpreting C code. An FMU vendor may only provide FMUs in binary form or may reference external libraries and source files. Hence, the RTI needs to be able to interface these entities which is beyond the scope of IEC 61499. When abstracting the interface via SIFBs, an RTI still needs to support these SIFB types but interface requirements can be reduced to the SIFB description which include formal Service Sequence Diagrams (SSDs). In case an SIFB interface which follows the virtual component scheme is used, direct dependencies of model variables as encoded by the FMI 2.0 may be transformed to SSDs. As soon as an event is triggered which updates a variable, all events associated with depending variables are issued. Nevertheless, the SSDs highly depend on the deployed interface method.

A transformation may automatically encapsulate the SIFBs which interface FMUs and generate boilerplate code to gain a virtual component view of the FMUs. All model variables may be mapped to FB in- and outputs to be accessible. Often, an FMU exposes various internal variables which should not be used as in- and outputs and instead are used to record results and debug FMUs. When automatically exposing these variables, it is not possible to restrict their use within the IEC 61499. A local variable may be fed to another (virtual) component which violates the FMI 2.0 specification [40, p.46]. An FMU may also expose numerous variables of which only a few are needed in the IEC 61499 application. Automatically exposing all model variables may therefore clutter the FB interfaces. A manual selection may be necessary to improve usability and maintainability of the system.

3.2.4 Real-time Operation

Synchronization of Simulation Time

The strictly periodic approach described in Section 3.2.2 may maintain its own simulation time different to real time and update it based on the state of the event queue and the time of the next event. This assumes that the next event time can be calculated by the IEC 61499 application as well as by the solver which wraps the FMI. Due to the strictly

periodic nature, the IEC 61499 application has to predict triggered events only until the next synchronization point t_{i+1}. Since IEC 61499 events initially can only be generated by resource initiated SIFBs such as timer or interrupt-driven SIFBs [92], it is sufficient to predict the next time instant at which these FBs will fire an event. Each event will have to be processed until no more event is left unprocessed or t_{i+1} is reached.

The solver which directs the FMUs may act in a similar way until t_{i+1} is reached. At each instant in simulation time t_i, the input and output information has to be exchanged between the IEC 61499 application and the FMUs. In contrast to the FMI, IEC 61499 does not provide any functionality to control the current progress of time like setting the actual time instant [39, 50]. Time control functionality would have to be implemented by the RTI executing IEC 61499 applications. To guarantee a synchronized and deterministic operation, time synchronization in the IEC 61499 domain has to be done on a level which covers every deployed entity. If just one resource is used, time has to be synchronized on resource level. If multiple devices are involved, the devices have to synchronized their notion of simulation time.

When using the prediction-based approach, a more fine-grained synchronization has to be used to synchronize simulation time between the IEC 61499 application and the FMUs. Not only IEC 61499-internal events but also events generated by the FMUs have to be considered in predicting the next IEC 61499 event. Due to the prediction mechanism, the time of the next FMI event will be known beforehand and the FMUs can be treated as native resource-initiated SIFBs. Although some co-simulations show that it is not impossible to synchronize simulation time [38, 110], it is unlikely that IEC 61499 RTIs aiming for a real-time-like operation will implement time adjustment features. An RTI which provides time adjustment features would at least have to implement external interfaces accessing the controller clock and enhancing the notion of time, if no event is processed. The additional interfaces introduce a considerable overhead which is not needed for industrial RTIs.

An alternative to synchronizing time between different simulation components is running these components in real time, eliminating the need of synchronizing the simulation time [86]. Additionally, real-time operation of FMUs also enables the use of components such as HuT or standard IEC 61499 controllers which can only be used in real time. On the other hand, a real-time operation requires the FMI and each FMU to provide real-time guarantees. The applicability of such a real-time operation strongly depends on the targeted time range and the implementation of each FMU. In case the execution time of an FMU is too long or one FMU is not able to provide any real-time guarantees at all, a guaranteed real-time operation is not feasible. In addition to the FMUs, also the interfacing algorithms, such as the solver or event mapping algorithm, must comply to the real-time requirements.

In order to discuss real-time operation in more detail, the function $t_c^k(t)$ is introduced, which maps the current real-time instant $t \in \mathbb{R}, t \geq 0$ to the progress in simulation time of the k-th component. For the sake of simplicity, it is assumed that both time scales, the real-time scale and the simulation-time scale can be directly mapped without

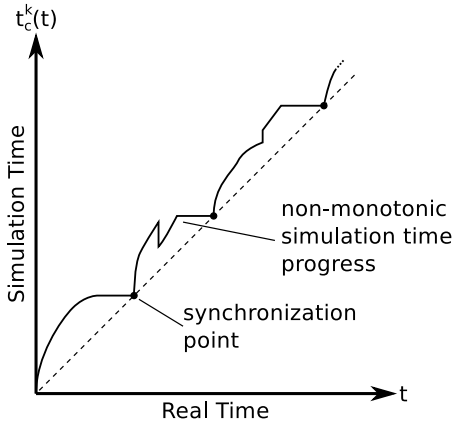


Figure 3.6: Exemplary simulation time function

any transformation, i.e. each instant of simulation time is also a valid instant of real time and vice versa. The exact shape of the function strongly depends on model- and tool-specific aspects as well as stochastic effects. Figure 3.6 shows one possible simulation time progress. One may note that, in general, monotonicity is not required. A simulation tool may reset any calculation, for instance if the desired accuracy is not achieved.

Real-time Periodic Event Mapping

Assuming strictly periodic event mapping as stated in Section 3.2.2, only synchronization at the time instants t_i has to be done in real time. Since involved FMUs and the IEC 61499 application do not interact between any two instants in time $t_i = i \cdot T_a$, the time $t_c^k(t)$ of the k-th component at the global time t may be scaled individually between two consecutive synchronization points $t_i \leq t \leq t_{i+1}$ as long as $t_i \leq t_c^k(t) \leq t_{i+1}$ [86]. At each synchronization point t_i, results from the previous period have to be available in order to be exchanged and the component time has to converge to

$$t_i = \lim_{t \to t_i^-} t_c^k(t). \tag{3.2}$$

Note that time-discrete components at t_i may have discontinuities in their simulation time function $t_c^k(t)$, which rises the need for a left-hand limit in (3.2). If and only if (3.2) is met for all components k, it simply follows by (3.3) that if all components converge to t_i in time, every component will be synchronized at t_i:

$$\forall i : \forall k : t_i = \lim_{t \to t_i^-} t_c^k(t) \Rightarrow \forall i : \forall k_1, k_2 : \lim_{t \to t_i^-} t_c^{k_1}(t) = \lim_{t \to t_i^-} t_c^{k_2}(t) \tag{3.3}$$

In case of synchronized components, each model maintains the same simulation time at each synchronization point t_i in real time.

This approach assumes exact time synchronization at each synchronization point t_i. In practice, measures like using hardware synchronization or minimizing event jitter have

to be taken to approximate the ideal behavior. Synchronization means including every event mapping component such as continuous-to-discrete and discrete-to-continuous event mapping. If an event mapping is distributed over several clock domains, additional synchronization effort is needed and the maximum accuracy of synchronization is limited by the precision of the ensemble of synchronized clocks [55].

To ensure hard real-time guarantees, each individual component time $t_c^k(t)$ must converge to the synchronization point in time as stated by (3.2). Physical systems use real time per definition and their time function $t_c^{phy}(t) = t$ always converges to the synchronization point in time. Depending on the purpose of the coupling scenario, IEC 61499 components may be seen as physical components which also run in real time. When targeting realistic device clocks, any deviation from the IEC 61499 device clock is part of the system behavior and does not need to be considered separately. In this case, the FMI interface needs to be externally synchronized to account for timing issues introduced by IEC 61499 applications.

When running FMUs and the solvers in a real-time mode, the FMI component will also have to provide the real-time guarantee of timely finishing the current simulation step. Assuming that the communication of data between the event mapping component and the solver is temporarily bound to the WCET, $WCET_{\text{em}}$, the solver has to execute one or more simulation steps within its WCET $WCET_{\text{sol}}$ and (3.4) has to be fulfilled:

$$WCET_{\text{sol}} + WCET_{\text{em}} < T_{\text{a}} \tag{3.4}$$

The solver's WCET highly depends on the WCETs of the FMI functions and the solving algorithm. Algorithms utilizing a variable step size or an iterative approach may not even provide a bounded WCET.

If, for example, the step size of an algorithm is determined based on the next upcoming event e_j, each integration step will be performed until $\max(t_{i+1}, t(e_j))$ is reached. The number of integration steps m_i is $m_i = n_i + 1$, where n_i corresponds to the number of FMI events triggered within two consecutive synchronization points $t_i < t(e_j) < t_{i+1}$. Assuming that the event update function and the numerical integration step have constant WCETs, $WCET_{\text{upd}}$ and $WCET_{\text{step}}$ respectively, and that manipulating the input and output values of the FMU is bound to $WCET_{\text{io}}$, the solver WCET calculates as

$$\begin{aligned} WCET_{\text{sol}} &= m_i \cdot WCET_{\text{step}} + (m_i + 1) \cdot WCET_{\text{upd}} + WCET_{\text{io}} \\ &= (n_i + 1) \cdot WCET_{\text{step}} + (n_i + 2) \cdot WCET_{\text{upd}} + WCET_{\text{io}}. \end{aligned} \tag{3.5}$$

The solver will have to call the event update function after every event instance and after an integration step is finished [39]. Since setting newly gathered input values at each synchronization point possibly generates an FMI event, the event update function also has to be called at the beginning of each period.

It follows from (3.5) that, if the number of triggered events n_i is unbounded, the solver WCET will also be unbounded. As a result, the implemented model does not only have

to provide WCET guarantees for executing provided functions but also has to limit the number of events triggered between any two consecutive synchronization points. These two features highly depend on the implementation of the model and are not covered by the FMI specification [39].

One naive approach of bounding the number of FMI events triggered is to delay any event request until the end of the fixed-size integration step is reached. Although some models may tolerate this approach, it violates the specification claiming that "the simulation shall integrate at most until `time = nextEventTime`, and shall call `fmiEventUpdate` at this time instant" [39, p.19]. Even if the integration algorithm does not adapt the step size based on any accuracy considerations, it generally has to support variable step sizes to handle FMI events.

Real-time synchronization is not necessarily restricted to the FMI, it is also possible to synchronize the process in- and outputs of the controller- only. Interfacing with FMUs can be done asynchronously and their results can be used to calculate the control output. Since the FMU inputs, which are valid at t_i, are provided after parts of the IEC 61499 application are executed, timing deviations introduced by the controllers cannot be considered. When executing IEC 61499 applications transparently without taking any timing deviations into account, the FMI will have to use IEC 61499 events to trigger its operation. In HIL setups, it might be useful to minimize the impact of IEC 61499 applications altogether. The functionality of the application within the proposed HIL scenario is not part of the actual simulation, instead the IEC 61499 application has to provide proper interfaces to access the hardware.

To study the impact of such a scenario, an adapted version of the periodic event mapping scenario is used. Instead of synchronizing the interface between the IEC 61499 application and the included FMUs, the interface between the HuT and the IEC 61499 application is synchronized. The IEC 61499 controllers pre-process measured data and redirect it to the solver. As soon as results are available, the data may be adapted by the IEC 61499 application to output them at the next synchronization point.

In contrast to the strictly periodic event mapping approach described in Section 3.2.2, the included models are part of the control operation in calculating the output which is issued to the HuT. Since the process interfaces of the IEC 61499 application are sampled periodically and synchronously, it can be safely assumed that the operation of the FMU is triggered after all input data is available and that no further events have to be considered during a simulation step. Such an assumption eases interfacing FMUs by reducing the need of storing previous events.

By synchronizing the in- and outputs of the application only, the system $WCET_{\text{sys}}$ is extended by adding the WCET of the controller parts $WCET_{\text{ctrl}}$, the event mapping $WCET_{\text{em}}$ and the $WCET_{\text{sol}}$ of the solver:

$$WCET_{\text{sys}} = WCET_{\text{ctrl}} + WCET_{\text{em}} + WCET_{\text{sol}}. \tag{3.6}$$

As described above, the solver can be directly called and the event mapping procedure at the FMI is reduced to solving the system of FMUs. The small communication overhead of passing on system states to the solver is included in $WCET_{\text{ctrl}}$. Although event mapping on the FMI side is reduced, it is still necessary to synchronize any external components. In order to guarantee the synchronization condition stated by (3.2), each subsystem listed by (3.6) needs to provide a bound WCET and $WCET_{\text{sys}}$ needs to be smaller that the sample period T_{a}.

Real-time Prediction-based Event Mapping

In order to operate the IEC 61499 controller which accesses the model in real time, each event e_k which is issued to the IEC 61499 application needs to be triggered at the corresponding instance of real time $t(e_k)$ [86]. One may naively deploy an event queue which buffers previously detected FMI events e_k and continues the simulation. To meet the real-time constraints, the simulation time of the FMU needs to be at $t(e_k)$ before real-time approaches, i.e. (3.7) has to hold. Again, $t_{\text{c}}^{\text{fmu}}(t)$ corresponds to the notion of FMU simulation time at the global instant of real time $t \in \mathbb{R}, t \geq 0$:

$$\exists t_{\text{detect}} : 0 \leq t_{\text{detect}} \leq t(e_k) \wedge \lim_{t \to t_{\text{detect}}^-} t_{\text{c}}^{\text{fmu}}(t) = t(e_k) \tag{3.7}$$

It turns out that (3.7) is a necessary, but not a sufficient timing condition for a general FMI-compliant operation. In particular, some issues arise when an event e_k is enqueued and the simulation time approaches beyond the FMI event and real time. As stated in Section 3.2.2, an FMU may not be able to properly reset the state before the last event time [39]. Assume that event e_k is correctly detected and enqueued at time t_{detect}, $0 \leq t_{\text{detect}} < t(e_k)$ and that the simulation of the FMU continues to detect further events, i.e. simulation time is beyond the last event time and assumption (3.8) holds:

$$\exists t_{\text{cont}} : t_{\text{detect}} \leq t_{\text{cont}} < t(e_k) \wedge \lim_{t \to t_{\text{cont}}^-} t_{\text{c}}^{\text{fmu}}(t) > t(e_k) \tag{3.8}$$

Since at t_{cont}, $t_{\text{cont}} < t(e_k)$, the FMI event e_k is still not triggered, an external IEC 61499 event e_j may be triggered at the real-time instant $t(e_j) \in \mathbb{R}$, $t_{\text{cont}} < t(e_j) < t(e_k)$, i.e. e_j is triggered before e_k is predicted but after the simulation continued at t_{cont}. Consequently, the event handling functions, which process e_k, are already called at $t(e_j)$ and in general it may not be possible to reset the (discrete) FMU state to $t(e_j)$. As a consequence, queuing of multiple FMI events is not feasible with the restricted first version of FMI and the real-time constraint for FMI events restricts to (3.9):

$$t(e_k) = \lim_{t \to t(e_k)^-} t_{\text{c}}^{\text{fmu}}(t) \tag{3.9}$$

When distributing an FMI event, the simulation time $t_{\text{c}}^{\text{fmu}}(t)$ of the FMU must be aligned to the current real-time instance [86]. Nevertheless, for FMI 2.0 FMUs which support state retrieval, (3.8) also provides a sufficient condition, if applied to all FMI events.

Events transferred in the reverse direction, i.e. from the IEC 61499 application to the FMUs, are not directly restricted by any real-time constraints [86]. Since multiple incoming IEC 61499 events may be stored in a queue and $t_c^{fmu}(t)$ can be set accordingly, each incoming event can be directly applied without considering real time. It is to note that the simplistic model of instantaneous triggering of events and simulation time progress function $t_c^{fmu}(t)$ hides several implementation challenges and limitations. In particular, resetting an FMU, processing events, and calculating model outputs requires a certain amount of time which limits the processing capabilities of other events. For instance, an FMI event needs to be handled and outputs need to be calculated before the event can be actually triggered. In case the event update function is not called before the outputs are fetched, the outputs will be based on the state before the event was triggered and hence they will be outdated. The processing time opens a time window where other events may not be processed properly. To analyze the real-time constraints, the event processing start time of any event e_j, $t'(e_j)$ is introduced. It corresponds to the instant of real time when the event e_j is removed from the queue and processing of the element starts.

The solver will have to update and re-predict states and outputs, if an event is triggered by the IEC 61499 application or by an FMU [86]. In the following discussion, $WCET_{\text{pred}}$ denotes the WCET of the state prediction and e_j corresponds to the last triggered event. Assume that $WCET_{\text{pred}}$ exists and that e_j was timely triggered and processed at $t'(e_j)$. Any outgoing FMI event e_k, which has to be triggered before the prediction operation finishes and new states are calculated, will violate the real-time condition expressed in (3.9). To discuss real-time properties in more detail, it is assumed that the notion of simulation time $t_c^{fmu}(t)$ remains constant until the prediction is fully performed, i.e. in the worst case (3.10) holds:

$$\forall t \in [t'(e_j), t'(e_j) + WCET_{\text{pred}}) : t_c^{fmu}(t) \coloneqq t(e_j) \tag{3.10}$$

Only after the prediction is completed, the simulation time $t_c^{fmu}(t)$ is eventually advanced. In case the event e_k, $t'(e_j) < t(e_k) < t'(e_j) + WCET_{\text{pred}}$ is triggered, the real-time condition (3.9) evaluates to

$$t(e_k) = \lim_{t \to t(e_k)^-} t_c^{fmu}(t) = \lim_{t \to t(e_k)^-} t(e_j) = t(e_j) \leq t'(e_j). \tag{3.11}$$

One can note that (3.11) directly violates the assumption that $t'(e_j) < t(e_k)$, which was stated before, and consequently, the event e_k will not be triggered in real time. Similarly, assume that updating a state and calculating associated outputs takes at most $WCET_{\text{upd}}$. Since the update functions have to be performed before an FMI event e_k is triggered, no other event can be processed at $(t(e_k) - WCET_{\text{upd}}, t(e_k)]$ either. Since the state update function in the time window has to be called before $t(e_k)$, especially no IEC 61499 event can be processed in that time window.

Form the observations above, it follows that an event e_j, which is triggered by an IEC 61499 application, must not closely occur before FMI events and that FMI events

e_k must not be triggered before the last event is fully processed [86]. The actual time window, in which the next event may safely occur, highly depends on the other events of the system. In particular, the processing schedule of the solver and even events triggered by the IEC 61499 application influence the permitted timing of the next FMI event. To break the complex inter-event dependency and to ease analysis and implementation, an emulation mode is introduced. In the emulation mode, it is assumed that all events, which are exchanged between FMUs and an IEC 61499 application are issued in time. Consequently, also events which are passed on to the FMUs must be delivered in time and cannot be delayed without violating the emulation mode assumption. Since storing an IEC 61499 event e_j into a queue would delay the execution, queuing in the emulation mode is not possible at all and every event has to be processed in time. Hence, the event and execution time requirements of events e_j restrict to $t'(e_j) = t(e_j)$ and the time between any two events must not fall below $WCET_{\text{pred}} + WCET_{\text{upd}}$. Additionally, the application must be able to tolerate event deviations of either FMI events or IEC 61499 events of up to $WCET_{\text{upd}}$.

Although the conditions which follow from the emulation mode are much more restrictive, the IEC 61499 application and the FMUs can now be analyzed separately [86]. In contrast to the periodic approach described above, the prediction-based event mapping still requires timing guarantees between any two events and not just between any two synchronization points. Whether such timing guarantees can be given, highly depends on the implementation of the application and FMUs.

The engineering effort of calculating a tight WCET is often very high and the execution time of some algorithms is not even bounded [86]. Then again, most soft real-time applications do not need any hard real-time guarantees and late results can be dropped or can be applied late without fatal consequences. Hence, the implementation effort can be drastically reduced by a best-effort approach but inaccuracies which result from deadline misses still have to be considered.

3.2.5 Model Coupling

Basically, two opposite approaches exist in coupling two or more FMUs within an IEC 61499 framework. The first one uses an FMI wrapper able to couple multiple FMUs and solve them in the time-continuous domain while the second approach uses means of IEC 61499 and bounds them via event and data connections. The first approach makes some demands to the solver and the event mapping component wrapping the FMUs which exceed single FMU operation. It first has to read additional user input stating the connections between different FMUs. Connections may not be directly defined by means of IEC 61499 because IEC 61499 does not provide any connection mechanism transferring data at any instant in simulation time.

Secondly, a solver handling multiple FMUs without restricting their direct dependencies has to handle systems of algebraic equations and ODEs [39, 40]. Especially in real-time operation such a solver has to be analyzed separately in order to gain appropriate real-time

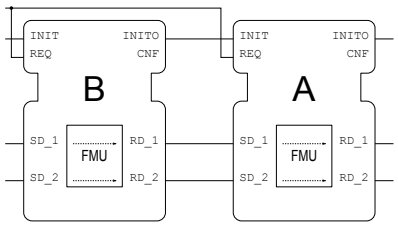


Figure 3.7: Naive approach for connecting two FMUs

guarantees. Solving each single FMU may not necessarily be performed synchronously. Instead, the whole subsystem including each FMU and the solver has to fulfill certain timing requirements. Analysing the overall timing might become much more complicated compared to a single FMU operation. Some algorithms solving non-linear algebraic equation systems, like Regula Falsi, use an iterative approach to approximate a solution [21]. Although it is possible to deduce an order of convergence, it is hard or even impossible to gain hard real-time constraints for the algorithm.

If it is not necessary to meet any real-time constraints, a dedicated solver capable of handling algebraic loops will produce more accurate results than coupling components via time-discrete connections. Time-continuous virtual connections can be modeled transparently while using sample-and-hold components, which are necessary to couple the FMUs within a time-discrete network, will introduce bandwidth limitations and discretisation errors [63]. On the other hand, coupling different FMUs using the event interface provided by IEC 61499 reduces the need for maintaining separate connection information. Each FMU may be encapsulated into a separate SIFB and different SIFBs may be connected by means of IEC 61499.

Depending on the event mapping approach and the timely operation, events issued by one FMU have to be redirected to connected FMUs in order to process new inputs. When using a strictly periodic event mapping approach, events may also be issued by a centralized timing component. In both cases, the sample time of the values has to be considered. If, for example, component A depends on the output of component B, the scheduling order of both components determines the output. Figure 3.7 illustrates the naive connection setup using undetermined scheduling. If component B is scheduled before A, A will use the outputs of B virtually sampled at the next synchronization point. If both FMUs execute concurrently, input values of A may even be randomly chosen. One approach of synchronizing the operation is to use separately synchronized latches. After every FMU has calculated its output, every component input will be latched and each FMU uses inputs from the same instant of simulation time. Latching may either be implemented by the FBs containing the FMUs themselves or by separate D (data latch) bistable FBs defined in IEC 61499 [50, p.64].

When using the prediction-based event mapping approach to couple multiple FMUs by means of IEC 61499, algebraic loops have to be taken into account. If two components outputs mutually depend on each other, an output event may be triggered directly after

the input changes. If both FMUs immediately trigger output events, an event loop will occur. Convergence of such an event loop would have to be shown separately and the resulting system may not be stable [63, p.507ff].

When using FMUs coupled via synchronized IEC 61499 connections, real-time analysis is simplified. Periodic synchronization points allow to calculate the WCET components separately instead of covering the whole system. The ability to decouple the timing of components is based on the restriction on sending events. Only at synchronization points these events will be triggered and data exchange will be issued synchronously. On using the prediction-based approach, events may depend on each other and these dependencies will have to be considered when calculating the system WCET.

3.2.6 Software Interfaces

The software interface accessing included FMUs has to provide the above described components, at least an event mapping component, a solver and several helper functions which are used to access the information of an FMU. One possible way of including FMUs is to provide very basic SIFBs to access the C functions which are provided by an FMU and to implement other parts by means of IEC 61499 [86]. One drawback of such a solution is the overhead introduced by IEC 61499. IEC 61499 focuses on industrial control tasks and not on numerical computation [50]. Although it might be possible to implement a solver by means of IEC 61499, it is most likely much more efficient to implement it separately. The following section will assume that the solver is implemented by means other than directly provided by IEC 61499.

The IEC 61499 defines the concept of SIFBs to access external services such as those provided by an FMU [50, p.44]. Except for few predefined SIFB types, the IEC 61499 does not specify the services itself and the actual implementation always depends on the executing RTI [86]. In order to implement an FMI service natively, the used RTI has to be extended such that it can access deployed FMUs. Although such an extension reduces additional configuration effort, the portability of an FMI-enabled IEC 61499 application is restricted to devices which actually implement the FMI service. Annex E of the IEC 61499 informatively describes an ASN.1-based protocol which covers uni- as well as bidirectional data transfer [50, p.95ff]. The protocol can be used to exchange data within a distributed application and may be used to access third-party tools as well. Since some RTIs already implement the ASN.1-based protocol, new services can be coupled via a network connection without modifications of the RTIs.

To manage the FMI integration, several functionalities for loading and instantiating FMUs have to be provided. Data transferred between the IEC 61499 application and FMUs depend on the output capabilities of the FMUs. Management functions have to parse the static information contained in the FMU to conduct the redirection of the FMU in- and outputs. The actual SIFB has to provide either a static interface covering all possible FMU types or a dynamic interface adapting its in- and outputs to the capabilities of the FMU. A static interface has to provide additional information

identifying the FMU in- and output of the provided information. A (de-)multiplexing component has to be implemented by the IEC 61499 application to separate different value semantics. Although it eliminates the need of adapting the service interface, for each FMU type a dedicated (de-)multiplexing component has to be created. Additionally, it is not easily possible to transfer multiple values in parallel unless the SIFB provides a variable number of in- and output variables. A simple protocol might be needed to synchronize sequentially arriving values to a common event time.

Sequential data transmission and a separate (de-)multiplexing component can be avoided by adapting the SIFBs to the capabilities of the FMUs. An adapted SIFB reflects the capabilities within the IEC 61499 framework but requires the corresponding service to provide adaptive interfaces. The manual effort for creating an adequate SIFB may be reduced by integrating proper tool support into the used Integrated Development Environment (IDE) or by providing a standalone tool. The descriptive XML file included in the FMU may be parsed to generate an SIFB definition. The XML representation of the IEC 61499 system might be used to generate the SIFB type without restricting support to single IDE vendors only.

Each integrated FMU wrapper may either control time by itself or issue results immediately and rely on the IEC 61499 application to delay results appropriately. The first approach may be useful if the timely behavior of the IEC 61499 application is to be taken into account. In this case, a different clock maintained by the FMI service has to be installed and used. The FMI service records IEC 61499 event times based on its own clock and delays output events accordingly. If the timely behavior of the IEC 61499 application shall be handled transparently, the application has to perform time coordination. The FMI service calculates the results as fast as possible and returns them to the application. The application now has to delay results until they get valid or has to adapt its notion of time based on the results returned.

Communication following the strictly periodic event mapping approach can be mapped to a bidirectional data transfer of communication SIFBs [50]. In the first scenario, the time-controlling FMI service acts like a client device and issues event requests to the server located at the IEC 61499 application. On receiving the request, the server snapshots its state and transfers it back to the client. The discrete-to-continuous event mapping is done partly by the IEC 61499 application by implementing a mechanism for retrieving previously set variable values. If controlling the time is up to the IEC 61499 application, the FMI service may act as a server providing the result of the next step on request. Although the FMI service may obtain the current simulation time by counting periodic requests, it might be useful to include a time stamp mechanism communicating the current simulation time. Time stamps may be used to initialize the simulation time individually and to detect communication and timing errors. Unfortunately, the IEC 61499 does not provide any mechanism for directly reading the device time [50]. Depending on the capability of the RTI, an up-counting approach has to be used by the IEC 61499 application too.

By using the prediction-based approach, events may be exchanged at any instant of time.

If the current simulation time is controlled by the IEC 61499 application, the controller has to properly time stamp events to allow the FMI service a temporal reconstruction. In this scenario, the FMI service acts like a server only, returning the next event time on request. Since controlling the event time is up to the IEC 61499 application, a logic has to be implemented which delays or rejects events returned by the server properly. Additionally, the IEC 61499 application has to settle the current prediction steps or promote time-stamped events to discard future states. The IEC 61499 neither provides any mechanism to directly retrieve the time stamp of an event nor provides any means to control the system time [50]. Implementing a time-controlling IEC 61499 application may therefore highly depend on the used RTI.

On the other hand, the FMI service implementing the prediction-based approach may include time management. The service will delay outgoing events according to their time and will time-stamp incoming events to select the proper state prediction interval. Additionally, a side channel may be used to adjust the time of the controller in a non-real-time mode, but this approach requires the IEC 61499 application to properly predict the next event time. Event prediction is not specified by the IEC 61499 standard either and therefore also highly depends on the RTI capabilities and implementation. In a real-time mode, the time of the application does not need to be artificially adjusted and no RTI-specific functionality, except the FMI service, is needed.

In contrast to the periodic approach, the prediction-based approach timely decouples in- and outgoing events. An event may either be initiated by the FMI service or by the IEC 61499 application, resulting in a mixed operation of the FMI service. The operation may be decoupled by using two different SIFBs, one resource-initiated and one service-initiated FB, to access the FMI service. Both SIFBs only feature unidirectional data transfer and may be implemented using a publish-subscribe communication connection. Alternatively, it is possible to combine service- and application-initiated interactions into a single SIFB [50, p.44], leading to a more compact FMI service interface.

Several libraries assisting the management of FMUs already exist [14, 32, 36, 48, 68, 108]. Some libraries implementing different algorithms for solving ODEs also exist [2, 41, 49]. By using appropriate libraries instead of starting from scratch, it is believed to speed up the development process and increase maintainability. However, before using external libraries, licensing issues and the planned license of the FMI service have to be investigated carefully. It is also important to check the functionality of the library carefully against needed features to avoid library lock-in scenarios.

3.2.7 Use Cases

Through the use of FMUs for model exchange in IEC 61499 applications, it becomes possible to simulate or emulate external components interacting with the IEC 61499 application. The components may be created and exported by any tool that supports the FMI and the implementation effort which is necessary to include models can be reduced drastically. Possible applications of virtual components which are integrated

via the FMI include model-based design and validation approaches. In these scenarios, the IEC 61499 application can be tested without the need for providing a test setup that includes the plant hardware. Depending on timing accuracy requirements of the simulation, a modified RTI has to be used. A modified RTI has to provide the ability of scaling time accurately and of simulating long-lasting periods without waiting until the next event is actually triggered.

During testing and validation, parts of the real hardware may also be included and tested using the real control algorithm. In contrast to a fully virtual simulation, FMUs and the controller have to be executed in real time. Scaling the simulation time according to the next event occurrence is not an option in general. If parts of the HuT directly interact with simulated system parts, an additional hardware interface and a side channel may have to be developed. It is either possible to use additional IEC 61499 components not needed during productive operation or to implement the side channel directly in the solver or event mapping component.

Strasser et al. proposed a co-simulation-based training platform used to educate laboratory personnel [91]. FMI could be used to include arbitrary FMUs for model exchange emulating real hardware. For training, a real-time approach should be used to demonstrate the system timing behavior. The SCADA system used to control the real plant may also be used to control the training setup, thereby avoiding confusion that might arise from a changed user interface.

The application of FMUs included in IEC 61499-based controllers is not limited to testing, training and validation but may also include productive use. An advanced control scheme may use predicted values to estimate the system state and to optimize the control output [6, 63]. Although each FMU will have to provide results without further delaying them, it might be necessary to provide hard real-time guarantees to calculate the overall real-time parameters of the system.

3.3 Using Co-Simulation FMUs in IEC 61499 Applications

3.3.1 System Architecture and Model

In contrast to the FMI for model exchange, the FMI for co-simulation does not require the controller to provide a solver on its own [38, 40]. Equations are solved by the FMUs themselves and data is only exchanged at well-defined sampling points. The simple system architecture illustrated in Figure 3.8 does not implement any particular solver but provides a component which implements the master algorithm. The master algorithm is split into three subcomponents: the IEC 61499 event to FMI conversion logic, the FMI to IEC 61499 event conversion logic and a step control logic which manages the execution of FMUs. Depending on the event handling strategy, the three subcomponents may be loosely coupled or interact very closely.

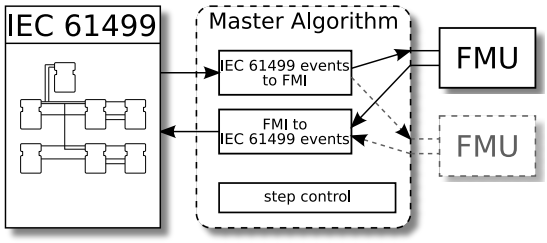


Figure 3.8: Basic abstract co-simulation system model

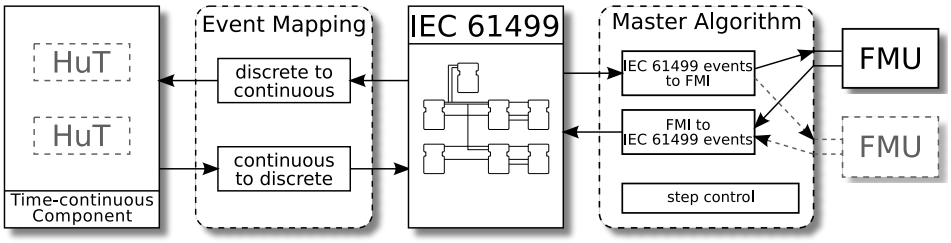


Figure 3.9: Extended abstract co-simulation system model

The IEC 61499 event to FMI and FMI to IEC 61499 event conversion logic have to synchronize their operation with the step control logic. Only if the executed FMU has finished its step, data exchange will be feasible. Depending on the applied event handling strategy, the IEC 61499 event to FMI conversion logic has to buffer incoming events and related values until the step control logic has finished the corresponding simulation step. Interpolating time-discrete values in a time-continuous domain is done by the FMUs itself and not by any external logic [38, 40]. The FMI standard does not restrict interpolation to ZOH, it also supports higher-order interpolation. It provides additional functions for accessing state derivatives at the current sample point optionally provided by an FMU [38, p.27f]. The IEC 61499 does not specify any higher-order interpolation but higher-order interpolation may be implemented by IEC 61499 FBs [50]. Alternatively, the event handling components may provide functions featuring higher-order interpolation.

The master algorithm used may not be restricted in handling only one connected FMU. Multiple FMUs can either be synchronized by an explicitly given master algorithm or by connecting several single master components by means of IEC 61499. To indicate both states, Figure 3.8 shows an optional second FMU but it is important to notice that the system may handle the integration of multiple FMUs in different ways. Like in Section 3.2.1, the basic system model is extended by an additional time-continuous system such as a HuT in a HIL setup. Figure 3.9 shows the extended system model integrating a time-continuous component. In the extended setup, the co-simulation FMUs are not able to directly communicate with the time-continuous components.

3.3.2 Event Handling

The FMI for co-simulation does not provide any mechanism for directly communicating discontinuities [38]. Instead, the FMI requires the master algorithm to exchange values, which is possible at given instances of time only. Between two consecutive communication points, continuous inputs may be interpolated by a finite order polynomial, but discontinuities of in- and outputs cannot be represented [21, p.173]. Instead, the function calculating the next step may fail and return the status code `fmiDiscard`. A master algorithm may query the time until the last communication step was successfully calculated [38, p.30] and adapt its step size accordingly. By the discard mechanism, events can be signaled indirectly. However, a master algorithm cannot distinguish various reasons for discarding a result. For instance, the step may fail due to an output discontinuity or because some internal algorithms of the FMU do not converge. Regardless of the actual reason, the FMU as well as the master algorithm have to support the discard mechanism.

Periodic Event Mapping

The strictly periodic event mapping approach, which is described in Section 3.2.2, can be adapted to the FMI for co-simulation. Instead of providing a solver, each FMU is accessed directly and data aggregated by the IEC 61499 to FMI conversion component will be communicated periodically. In the simplest implementation, a ZOH interpolation is used and no state derivative is accessed. The strictly periodic approach does not require any optionally supported FMU feature and provides compatibility to all FMUs which follow the FMI specification. If an FMU indirectly signals the master that an event has occurred, the master algorithm will have to ignore the event and continue the step. Processing any interrupted step requires subsequent FMUs to accept variable step sizes or to reject communication steps, which is not supported by all FMUs.

A critical parameter in the strictly periodic event mapping approach is the chosen communication step size T_a. Selecting a proper step size highly depends on the included simulation and the capabilities of the tool. According to Shannon's sampling theorem, a time-continuous signal can only be reconstructed if its bandwidth B is limited and does not exceed the half sampling frequency $\frac{f_\mathrm{s}}{2}$ [20, p.72ff]. If the step size $T_\mathrm{a} = \frac{1}{f_\mathrm{s}}$ is chosen too large, the reconstructed signal will be biased and will not contain the intended information. Small step sizes, on the other hand, may negatively affect the simulation performance and can also hinder a successful co-simulation. Some tools may only accept a limited range of step sizes and the step size of the entire system has to be adapted to the capabilities of the tools. Although the second version of the FMI includes a preferred step size definition, step size limitations cannot be communicated directly [38, 40].

The IEC 61499 event to FMI conversion component may utilize the information associated with incoming IEC 61499 events to approximate the FMUs input derivatives. Prentice proposed a numerical differentiation algorithm approximating derivatives of a function without the need of constant sampling step sizes [76]. On receiving an IEC 61499 event between two subsequent communication points t_i and t_{i+1}, the associated data will

be sampled and taken as node. Since Prentice's algorithm does not require the point of approximation to be a node, the derivatives can be approximated at t_{i+1} without receiving an incoming event at this point in time. Alternatively, the derivatives may be approximated by interpolating cubic polynomial splines using the same, not necessarily equidistant nodes [24, p.549ff]. In general, numerical differentiation suffers from decreased accuracy compared to numerical quadrature [24, p.559]. Depending on the IEC 61499 event source and the frequency of events, a constant interpolation is preferred. For example, a noisy analog input signal may cause large errors in the numerically calculated derivatives. In this case, a constant interpolation value is expected to be more accurate.

Some FMUs may provide state derivatives approximating the output function. A component which generates IEC 61499 events may utilize the provided information to approximate the simulation output between two subsequent communication points. These values can be forwarded on request of an IEC 61499 application which decouples the strictly periodic operation of the event mapping component. Triggering IEC 61499 events between two subsequent communication points and approximating values between these points introduces an error R_n where n corresponds to the order of the highest output derivative provided. Furthermore, t corresponds to the time of the intermediate IEC 61499 event and t_i denotes the last communication point. Assuming that the approximated time-continuous function $f(t)$ is n times continuously differentiable in $[t_i, t]$ and $n+1$ times continuously differentiable in (t_i, t), the magnitude of R_n can be given by [21, 76]

$$R_n = O((t - t_i)^{n+1}) \tag{3.12}$$

Using Taylor's theorem, the n-th Taylor approximation of the function $f(t)$ is given by

$$f(t) = \sum_{k=0}^{n} \frac{f^{(k)}(t_i)}{k!}(t - t_i)^k + R_n$$

According to the theorem, R_n is given by (3.13), with $\xi \in (t_i, t)$:

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(t - t_i)^{n+1} \tag{3.13}$$

To prove (3.12), it has to be shown that a constant value $C > 0$ exists such that (3.14) is fulfilled [21, p.159]:

$$\forall n \in \mathbb{N} : \left| \frac{R_n}{(t - t_i)^{n+1}} \right| \leq C \tag{3.14}$$

From (3.13) easily follows (3.15) and thereby the existence of a constant value C for every $n \in \mathbb{N}$.

$$\left| \frac{R_n}{(t - t_i)^{n+1}} \right| = \left| \frac{f^{(n+1)}(\xi)}{(n+1)!} \right| = C \tag{3.15}$$

As a consequence of (3.12) and (3.13), the error term increases by increasing the distance to the last communication point t_i. If the IEC 61499 application needs more accurate

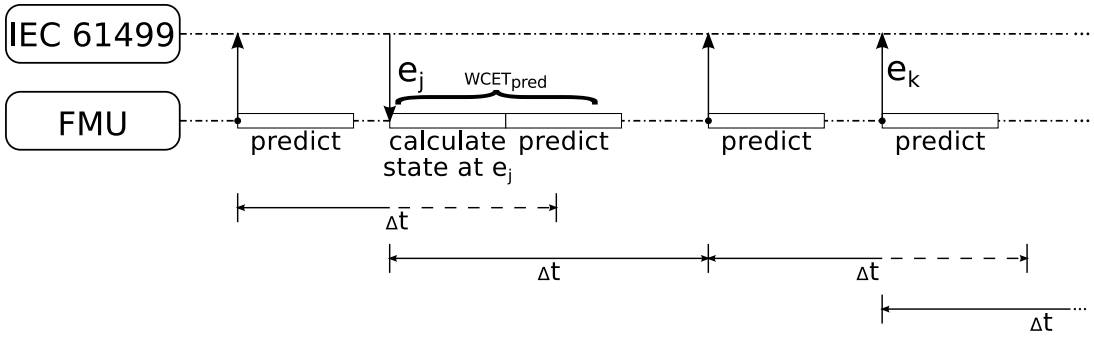


Figure 3.10: Prediction-based event mapping using the FMI for co-simulation

timing data, generally more precise values can be obtained by decreasing the sampling period T_a according to the control requirements. By decreasing the sampling period the value of $f(t)$ can be directly communicated without introducing any approximation error. On the other hand, decreasing T_a may raise performance issues or may not be supported by an FMU. In such cases, a polynomial approximation of intermediate IEC 61499 events is still advisable.

Prediction-based Event Mapping

The prediction-based event handling approach for the model exchange FMI described in Section 3.2.2 can only be adapted to the FMI for co-simulation in a very limited way. It strongly relies on calculating, storing and resetting an FMU state at intermediate points in time until the prediction horizon is reached [68, 108]. An FMU may only optionally provide the capability of resetting its state and even suitable FMUs in FMI 1.0 are only able to reset the very last communication step [38]. The state machine of calling sequences defined in the FMI for co-simulation restricts the next simulation time of a communication step to be either the current simulation time or the previously used simulation time. In contrast to the FMI for model exchange, directly accessing the state of a co-simulation FMU is not possible using the first version of the FMI only. FMI 2.0 introduces an optional mechanism to store and recall the entire state of a co-simulation tool at a particular communication point [40]. As the limited reset functionality provided by FMI 1.0, the state retrieval mechanism of FMI 2.0 is still optional.

For the prediction of future events, the FMU has to provide the capability of triggering events. In case of an FMU event e_k, the processing function `fmiDoStep` may return prematurely in which case the event time can be retrieved [38]. Figure 3.10 illustrates the event mapping approach using the FMI for co-simulation. On receiving IEC 61499 events e_j, the current simulation state has to be reset to the beginning of the communication step before the event has happened. In case of FMI 1.0, only the very last step may be reverted. On the other hand, multiple intermediate states may be saved with the FMI 2.0 which allows a more fine-grained reset mechanism. After the FMU is reset, the

whole step has to be re-calculated until the point of time when the event was triggered. Re-calculating the state does not only require the FMU to reset predicted states after the event e_j happens but also lacks in efficiency by discarding valid results before the event e_j. Interpolation of intermediate states is not possible because no details regarding the state are exposed. The prediction-based approach is limited to FMUs which support variable communication step sizes, rejecting communication steps or exposing the state and signaling internal events.

Also, a step size reduction which dynamically generates IEC 61499 events on large output deviations requires a re-calculation of valid results. If an output deviation is larger than tolerated, the current step has to be re-calculated using a smaller step size. A larger step size may only be applied for future steps without discarding any results. One may naively take another step in one prediction, if the output deviation is still tolerable and no IEC 61499 event should be triggered. When conducting more than one step in a single prediction period, only the last step of a 1.0 FMU can be properly cancelled while preceding states, which are sill predictions, cannot be reset. Consequently, to extend the step size of a single prediction, one would need to discard the valid but short step and perform a larger step instead. The second version of the FMI limits the need of discarding results by providing a mechanism for directly accessing the FMU state [40]. If the optional access mechanism is supported, the FMU may be reset to arbitrary states and a finer-grained prediction mechanism which includes dynamic step size adaption can be used.

3.3.3 Data Model Transformation

Since FMI for co-simulation and FMI for model exchange share a common base [38, 39, 40], most findings regarding the data model transformation of Section 3.2.3 can be applied to co-simulation as well. The only significant difference regard the representation of model equations and variable extrapolation functions which are available in co-simulation only. In case the model equations are imported to algorithms, no separate solver logic needs to be created. Nevertheless, the same limitations regarding the capabilities of an RTI also exist for co-simulation. An RTI still most likely has to handle external dependencies which is beyond the scope of IEC 61499. Hence, an interface using SIFBs may still be favorable.

FMI for co-simulation optionally allows to set and query derivatives of model variables to extrapolate values in simulation steps. Neither IEC 61499 nor IEC 61131 provide facilities to amend variables with their derivatives [19, 50]. As stated above, an event-mapping component may utilize and numerically estimate the derivative values without exposing them directly. Consequently, an advanced control application cannot directly use the derivative values. In order to avoid loss of information, derivative values may be exposed by a separate set of variables. Thereby, either an implementation which adds the derivatives as separated FB in- and outputs or an implementation which encapsulates the derivatives and the corresponding variable in a dedicated structural type are conceivable.

In any case, manual variable selection may be beneficial to reduce the amount of exposed data and to gain readability and maintainability.

3.3.4 Real-time Operation

The timely operation of FMUs for co-simulation is very similar to the operation of model exchange FMUs described in Section 3.2.4. The strictly periodic operation as well as the predictive operation described in Section 3.3.2 are based on the approaches presented in Section 3.2.2. Hence, most concepts are applicable to co-simulation FMUs as well. The following section will only highlight differences in the real-time analysis of both FMIs without re-stating common concepts.

Periodic Event Mapping

In contrast to the FMI for model exchange, the WCET of solving a complete simulation step $WCET_{\text{sol}}$ cannot be divided in an event update and solving portion. The WCET of the `fmiDoStep` function $WCET_{\text{step}}$ already contains any event update and equation solving logic. When using the strictly periodic event mapping approach in conjunction with FMUs that signal events by partially calculating the next step, `fmiDoStep` may have to be called repeatedly. Assuming that the WCET of value in- and output operations needed to access the in- and outputs of an FMU is bound to $WCET_{\text{io}}$ and that n_i and $m_i = n_i + 1$ correspond to the number of triggered events and the number of simulation steps to take respectively, $WCET_{\text{sol}}$ can be given by

$$\begin{aligned} WCET_{\text{sol}} &= m_i \cdot WCET_{\text{step}} + WCET_{\text{io}} \\ &= (n_i + 1) \cdot WCET_{\text{step}} + WCET_{\text{io}}. \end{aligned} \tag{3.16}$$

By using (3.16), it can be seen that the system will not be able to fulfill the WCET condition (3.4) if the number of events n_i is unbound. Similar to model exchange FMUs, the real-time criteria (3.2) can only be guaranteed if the number of events is limited or the FMU does not communicate any event at all.

Prediction-based Event Mapping

State prediction in the FMI 1.0 for co-simulation is performed by calculating one step ahead. In contrast to the state prediction in FMI for model exchange, it is necessary to re-calculate the whole period if an external IEC 61499 event has occurred. Similarly, the period from the last saved state to the event time has to be re-calculated in FMI 2.0 when using the state retrieval mechanism. In case of an external event, the step function of the FMU has to be called twice and the prediction WCET evaluates to

$$WCET_{\text{pred}} = 2 \cdot WCET_{\text{step}} + WCET_{\text{io}}. \tag{3.17}$$

As a result of the inefficient operation stated in (3.17), the minimum interval between any communication point and an IEC 61499 event issued to an FMU is artificially extended and restricts the real-time capability of the system.

A WCET analysis may not only have to cover the code of the FMU itself but also that of connected tools as well as communication facilities. Even if the source code of coupled tools is available, it is very unlikely that complex simulation tools which are not tailored for real-time operation provide any tight WCET. Currently, no co-simulation slave targeting hard real-time operation is known or known to be developed [33]. In addition, the used communication facility may introduce another source of uncertainty and may not provide any real-time guarantees either. An application running in real time will most likely have to implement a best-effort approach without any guaranteed response time.

3.3.5 Tool Coupling

The FMI for co-simulation needs a master algorithm which actively schedules the execution of slaves [38, 40]. The master algorithm may either be implemented by means of IEC 61499 or externally using some kind of FMI wrapping component. On the one hand, using IEC 61499 FBs to implement the master algorithm reduces the implementation effort of the FMI wrapper and increases the flexibility. The master algorithm can be adapted to the application requirements without changing the FMI wrapper and a lightweight wrapper can be used. On the other hand, implementing the master algorithm directly in the IEC 61499 application adds complexity to the application and may require the control engineer to deal with the complexity of the master algorithm. Especially on implementing advanced master algorithms, additional information like the capabilities of the slave have to be passed on to the IEC 61499 application.

An advanced master algorithm coupling several FMUs does not only have to transfer the last step results but also has to consider the structure of the FMUs by solving any occurring algebraic loop at each communication step [38]. When resolving an algebraic loop, a subset of connected FMUs may have to re-calculate the last step without actually increasing the global simulation time. Co-simulation slaves which are not involved in any algebraic loop do not need to be considered while resolving algebraic loops and should not be reset to gain proper performance. Hence, the advanced master algorithm has to be aware of the connection structure to detect any loop and resolve it properly.

Although an advanced master algorithm may query connection information of FBs by using management FBs [50, p.49ff], it adds another layer of complexity to transparently handle the execution of the FMI FBs. Especially if the control engineer who is configuring the IEC 61499 application should not have to deal with details of the master algorithm and if the master algorithm should be able to handle a variable number of connected FMUs, it is more constructive to implement the master algorithm separately. In case of an external master algorithm, connection information has to be provided to the external algorithm. Although the connection information may be tailored to the needs of the master algorithm, it requires a user and configuration interface dealing with the additional information.

In case of simple master algorithms such as fixed step size and non-repeating algorithms,

the algorithm can be implicitly implemented by means of IEC 61499. The value connections between different FBs which wrap used FMUs directly represent the system structure. Simple scheduling may be implemented by using event connections. Because the master algorithm will not utilize any capability information and advanced time management, the interfaces of the IEC 61499 application to the FMUs can be kept simple as well. In contrast to the FMI for model exchange, coupling multiple FMUs by using IEC 61499 connections does not introduce additional errors. Each FMU only communicates its values at specific points in time and cannot utilize any communication between these points. For each communication point, it is feasible to trigger an event for exchanging the calculation result.

3.3.6 Software Interfaces

According to the system model described in Section 3.3.1, a master algorithm as well as some logic managing connected FMUs are needed. The master algorithm controls the execution and data exchange of FMUs and may be implemented by simply calling the FMI value access and `fmiDoStep` functions on receiving incoming events. Management logic is needed to load and instantiate connected FMUs and, depending on the implementation of the master algorithm, to configure the connection structure.

The stated interaction scheme between an IEC 61499 application and one or more FMUs does not differ significantly between FMI for co-simulation and FMI for model exchange. Software interface concepts developed for the FMI for model exchange are applicable to co-simulation as well, assuming the model data is not directly accessed and the operation of the solver is not restricted beyond the limits provided by the FMI for co-simulation. Co-simulation constraints include the restriction of data exchange to discrete points in time only and require the solver to be able to interrupt its operation. All criteria are met by common solving algorithms which justifies the use of SIFB concepts described in Section 3.2.6 for co-simulation scenarios as well.

Because every FMU for co-simulation just communicates the in- and output values of the system instead of requiring a solver which calculates the continuous states, only a solver which handles algebraic loops has to be provided. Although an FMI service faces a reduced implementation effort compared to an implementation containing a full-blown ODE solver, using external libraries that support FMU management still might speed up the development process. Like for the FMI for model exchange, libraries supporting the FMI for co simulation are already available [32, 36, 48].

3.3.7 Use Cases

The testing, training and validation use cases presented in Section 3.2.7 can also utilize the FMI for co-simulation, if each involved tool supports the interface. Instead of using a solver common to all included models, a set of specialized solvers maintained by coupled simulation tools can be deployed. By using the FMI for co-simulation, models which do not provide an ODE representation but maintain an FMI interface, can be included as

well. Such models include lookup tables or other data-driven models like trained artificial neural networks and support vector machines. These models may be implemented by the FMI for model exchange which utilizes the event mechanism as well. However, using the FMI for co-simulation introduces less overhead due to simpler access functions. Currently, no tool focusing on data-driven models is known but some tools that only support the FMI for co-simulation exist [33].

As described in Section 3.3.4, it is unlikely that a co-simulation FMU will be able to provide any hard real-time guarantees. Hence, it is not advised to use these FMUs to control safety-critical applications. Instead, FMUs may be used to simulate critical system behavior parallel to its operation and pass the system inputs on to the connected FMUs [75]. In the parallel observer setup [63, p.335f], large deviations between measured and simulated values indicate a possible system failure. Although the FMU may not be able to provide any real-time guarantees, it may be used in safety-critical setups featuring fault detection and diagnosis.

3.4 Encapsulating IEC 61499 Applications in Model Exchange FMUs

3.4.1 System Architecture and Model

Encapsulating any external control logic into an FMU requires the executing component to provide a set of features usually not found in industrial control applications [37]. To be fully FMI-compatible the notion of time of the FMU must not simply increase linearly according to real time but has to be controlled by the external solver which includes the FMU [39, 40]. Additionally, an FMU for model exchange 1.0 is required to use given memory management functions only [39, p.16] and may be instantiated several times. In FMI 2.0, an FMU may signal that it is not able to be instantiated multiple times or to use external memory allocation functions [40] which may slightly ease implementation. In order to encapsulate an IEC 61499 application or a set of applications into an FMU, for both FMI versions it is necessary to provide a specialized RTI or to abstract a common RTI using a component which emulates the code of the RTI.

Developing a specialized RTI requires detailed considerations about the used software architecture which is beyond the scope of this thesis. The basic system model illustrated in Figure 3.11 shall support a basic discussion of encapsulating IEC 61499 applications into a model exchange FMU and does not provide further details needed to implement any specialized RTI [55, p.29ff]. The proposed system model also focuses on run-time aspects of the FMI and does not deal with the infrastructure which is needed to develop and pack the FMU.

The model encapsulating the IEC 61499 applications, which is called IEC 61499 FMU, has to expose several features using the FMI. A static model description file containing the variables of the FMU may be derived from the IEC 61499 system description. Because the IEC standard 61499 does not specify a unit system based on physical units, a correlation

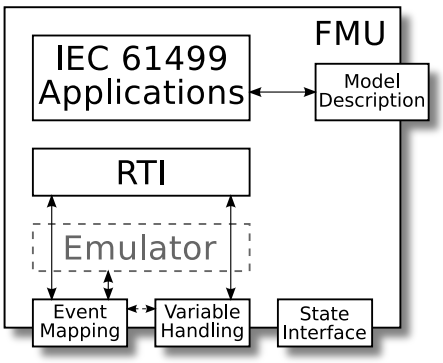


Figure 3.11: Basic model exchange IEC 61499 FMU

between unit definitions included in the model description and used data types has to be established [39, 40, 50]. Further information regarding the semantic of a model variable may be used to enhance the model description by providing adequately scaled units. Because every component using the FMI has to access the static data before any C function can be used, the XML-based model description has to be generated before the IEC 61499 FMU is distributed.

The objective of the event mapping component is to properly indicate changes in the internal time-discrete state, especially focusing on exposed model variables. If an event is issued to the IEC 61499 FMU, the FMU will propagate it by using the implemented event handling strategy. In order to determine the time of an event, the event mapping component has to implement the time management function which sets the notion of time for the application. The variable handling component displayed in Figure 3.11 provides access to virtual in- and outputs of IEC 61499 applications. It has to interpolate every read variable between two IEC 61499 events and has to provide a continuous time base. It may also have to trigger IEC 61499 events, if some application-defined criteria are met. A state interface handling the continuous states of the system must be provided by an FMI-compatible FMU [38, 40]. Because the IEC 61499 does not define any continuous states and the electro-physical behavior of a controller will not be covered in this thesis, no continuous states are communicated. The state interface only provides simple function stubs which establish FMI compatibility.

Depending on the purpose of the model, either an adapted RTI or a real RTI implementation which is executed by a software emulation tool may be used. On the one hand, an adapted RTI may support event prediction and may execute IEC 61499 applications without the need of emulating any low-level machine code. On the other hand, a software emulator like an Instruction Set Simulator (ISS) will be able to model the behavior of the controller more accurately [87] and will not introduce timing deviations based on RTI modifications.

A controller can only utilize external hardware if the hardware output does not depend

on the current time instant. The solver needs to adjust the current simulation time according to the implemented algorithm and does not necessarily run in real time. As the controller generally interacts with external hardware in a bidirectional way and the FMI neither provides any real-time guarantees nor restricts the use of an FMU, driving external hardware is not possible. Hence, no system model including external hardware like those stated in Sections 3.2.1 and 3.3.1 has been developed. If the interaction with external hardware shall be studied by several independent models, it is advised to use the FMI for coupling the controller and the hardware model.

3.4.2 Event Handling

Assuming that the RTI which is simulating the IEC 61499 system maintains timed event queues and provides the capability of predicting the next IEC 61499 event, the FMI can be integrated by using the provided FMI event mechanism. If the solver conducts an event update procedure, the next item is removed from the event queue and the corresponding algorithm is scheduled. If the event update procedure is called by an externally triggered event, no suitable IEC 61499 event will be present in the event queue and a resource-initiated IEC 61499 event may be triggered. After the event update has been executed, the next event time has to be reported to the solver. Reporting may either be done using the event indicator variables or by returning the next event time in advance [39, 40]. If the next event time depends on continuous inputs, only event indicators are feasible. In such an event mapping scheme, IEC 61499 events may be triggered whenever the input deviation exceeds a certain threshold.

If the RTI is able to predict the next event occurrence, it does not need to reset its internal discrete state. Algorithms associated with a certain event are executed on calling the event update function. To model the execution time of algorithms, any subsequent event can be delayed by an emulated execution time or by annotated static timing data. If an external event is triggered, it will be scheduled conforming to the scheduling function of the RTI but does not interrupt any algorithm which is executed on the resource [50]. Even if the intermediate event changes the executed FB inputs, it will not change the outcome of the algorithm. The ECC state machine will sample input values after an event is triggered and the algorithm will use the sampled values only [50, p.34]. Since no incoming event can interfere with the executed algorithm, it is not necessary to reset the system state.

The assumptions made above require an FMI-tailored RTI which is not available off-the-shelf. By using an existing RTI which does not provide any interface to the internal event queue, only events which are propagated to external components will be available. An event mapping component may conduct the execution of the RTI until the next IEC 61499 event is triggered or a prediction horizon is reached. The predicted event time may be passed on to the solver which triggers the next prediction. If an FMI event happens before the predicted IEC 61499 event is triggered, the IEC 61499 system state has to be reset to the time of the last event. Resetting may be done by frequently taking snapshots and re-calculating any period between the last snapshot and the incoming

event. Although it is believed that the emulation-based approach gains the most accurate results, managing the state of an RTI may lead to poor run-time performance.

A periodic event-mapping approach which features an emulated off-the-shelf RTI eliminates the need of resetting the system state. The event-mapping component will periodically indicate events and any communication between the emulator of the controller and variables which are exposed by the FMI will take place at these communication points only. On receiving an event update indicating a communication point, the RTI is executed until the subsequent communication point is reached. The RTI or the executing emulator still have to be able to stop the simulation at any communication point, but resetting the sate is not needed anymore. Deferring the communication to a dedicated communication point introduces additional errors and may not accurately reflect the timing of the controller. Only if the simulated RTI uses a strictly periodic event-mapping approach too, resulting errors can be seen as a part of the system behavior without introducing any additional numerical error. In this case, it must be ensured that the used communication points equal the control sampling points.

3.4.3 Data Model Transformation

The task of an automated or semi-automated data model transformation is to generate the static FMI model description and binary executables from the IC 61499-compliant system description according to a particular coupling strategy. Thereby, as much information as possible should be preserved and manual intervention should be kept at a minimum. In order to provide IEC 61499 compliant interfaces to external components such as the simulation tool which uses the IEC 61499 FMU, SIFBs have to be deployed [50]. Depending on the embedded RTI, an FMI implementation may access more information than those exported via SIFBs, but access methods cannot be represented in terms of IEC 61499. Similar to Section 3.2.3, an abstraction via generic FBs is used to discuss the model transformation. Beside an SIFB implementation, some interface logic may also be represented in terms of IEC 61499 and encapsulated into CFBs.

In order to encapsulate IEC 61499-based systems into FMUs, data types from the target system need to be transformed to the FMI data type representation. In particular, IEC 61499/IEC 61131 types need to be casted to the basic FMI data types which are used in the C interface. Casting can be done either directly in the IEC 61499 system or by the FMI wrapping component. In case casting is explicitly done by means of IEC 61499, the interface FB just needs to support the basic data types which can be directly mapped to FMI types.

Since types which are not directly compatible with the basic FMI type system will not be exposed, some type information may be lost. When converting types within the wrapping component, used IEC 61131 types may be transformed to custom type definitions within the model description. Minimum and maximum values may restrict the range of FMI types according to the range of IEC 61131 elementary types. Nevertheless, for types such as `TIME` and `DATE`, no corresponding FMI type exists. A mapping convention or

custom mapping function has to be deployed in order to access these types via FMI. For structural data types of the IEC 61499 system, multiple FMI model variables which follow the proposed naming convention have to be exported. Since there is no way of directly specifying units in terms of IEC 61499 [50], FMI unit definitions cannot be automatically generated. However, a transformation facility may support user-defined type information to amend exported variables.

The selection of exposed model variables may either be done manually or automatically via specialized SIFBs which communicate exposed variables. In any case, exported variables need to be chosen while transforming the data model and generating the IEC 61499 FMU. A manual selection may reduce the amount of exposed variables without changing the IEC 61499 application but an automatic selection may reduce user intervention while transforming the system. The encapsulated RTI may additionally provide model variables beyond the IEC 61499 standard. Such variables may be used to debug the system and the RTI itself. For instance, an RTI may expose the internal variables of an FB to debug it. Nevertheless, for the selection of model variables, a separate mechanism which is beyond the IEC 61499 has to be deployed.

Since IEC 61499-based systems operate in a discrete mode only [50], an IEC 61499 FMU does not have to expose continuous state variables. Only when interpolating outputs by other means than ZOH, state variables and continuous outputs are necessary. Since an IEC 61499 FMU may also allow continuous inputs which will be discretized, state events may still be used to signal triggered IEC 61499 events. Hence, the FMU may expose a set of event indicators to trigger event processing.

3.4.4 Real-time Operation

Running the IEC 61499 applications encapsulated into an FMU requires timed data and event flow from the solver to the IEC 61499 applications. The FMI does not specify any timing requirements kept by the solver and maintains its own simulation time [39, 40]. The simulation time does not necessarily correspond to real time and the real-time duration between any two FMI function calls may be arbitrary and unbound. As a consequence, real-time operation of IEC 61499 applications which requires to receive events or data before the associated real-time instant is generally not feasible. Particularly, the event mapping schemes requiring to reset the state of the RTI do not execute in real time. If a period in time is once executed in real time it cannot be executed again using the same real-time instances.

The event mapping approach which manipulates the resource event queues requires a specialized RTI version providing access to the event queues and introduces a certain access overhead. Because such an RTI is not found in real-time applications, the real-time execution may either be defined corresponding to the event timing of a reference RTI or corresponding to external interface timings. The first case strongly depends on the timing of the FMI event update function, which is not restricted by the FMI. The second case

may decouple the reference timing of the interface up to a certain extent but generally also requires a timed event processing.

A strictly periodic event-mapping approach further decouples the timing of the controller from the FMI operation. The solver has to issue the synchronization event before the synchronization point in real time is reached, but if it is triggered too fast, the event mapping component may block the operation of the solver. For real-time applications, it is still necessary to restrict and analyze the timing of the solver, but synchronization efforts do not cover every IEC 61499 event anymore.

Hard real-time operation only based on the FMI specification is not possible but the used solver may provide certain additional real-time guarantees. To analyze the real-time behavior of the system, only the event update functions have to be considered. Other functions which are accessing exposed variables do not directly trigger the IEC 61499 execution [50] and do not have to be targeted during timing analysis. Only when determining the overall execution time of the solver, the access function timing needs to be considered. Depending on the implemented event mapping approach and the component which requires real-time execution, a subset of event update function calls, denoted by $\mathbb{E}$, must be issued in time. Following the nomenclature described in Section 3.2.4, the real time (3.2) and (3.9) may be generalized to (3.18) which is applicable to various use-case-specific timing requirements.

$$\forall e_k \in \mathbb{E} : t(e_k) = \lim_{t \to t(e_k)^-} t_\mathrm{c}^\mathrm{fmu}(t) \tag{3.18}$$

For instance, the periodic event mapping approach only requires periodic communication events to be issued in real time and $\mathbb{E}$ is given by

$$\mathbb{E} = \{e_i | i \in \mathbb{N} \wedge t(e_i) = T_\mathrm{a} \cdot i\}.$$

Because the FMI specifies passive FMUs which are not initiating any action, (3.18) must be guaranteed by the solver. Every WCET analysis to guarantee the event timing has to cover the implementation of the solver. Necessary prerequisites of analyzing the timing of the solver are bound WCETs of used FMU functions including those provided by an IEC 61499 FMU.

3.4.5 Model Coupling

The IEC 61499 application which is encapsulated within an FMU may be coupled to external system models using the FMI [39, 40]. Any logic coupling multiple FMUs must be implemented by the including software and does not have to be covered by the FMU itself. However, an IEC 61499 FMU may utilize model coupling to communicate to other IEC 61499 FMUs via the FMI. Splitting the IEC 61499 system across multiple FMUs reduces the implementation effort of an FMU to single device or resource implementations but requires different FMUs to maintain parts of the same IEC 61499 conform system description. Additionally, the resource or device topology has to be reflected by combined

FMUs and the connections between these FMUs. The solver has to directly utilize IEC 61499 connection information or the information has to be transformed for a particular solver to properly model the IEC 61499 system.

Encapsulating every IEC 61499 device into one FMU eliminates the need of synchronizing any connection information but increases the implementation complexity of the FMU. The wrapping FMU has to differentiate between multiple independent devices and must properly shield used resources like virtual hardware ports, memory and processor time. Moreover, the wrapping FMU must implement virtual network links passing on information between devices and may have to handle communication aspects such as the packet timings and failure rates.

3.4.6 Software Interfaces

An IEC 61499 FMU may maintain a set of internal model variables which are exposed by the FMI. These internal model variables may be accessed by an IEC 61499 application via SIFBs either in an application-initiated way or by triggering IEC 61499 events whenever a model variable changes significantly. Each exposed model variable may simulate a virtual in- or output connection, linking the controllers to external hardware.

Depending on the used event mapping approach, additional software interfaces which control the current execution and the controller time are necessary. The periodic event-mapping approach makes the least demands on additional software interfaces. Only a common time base which properly synchronizes the controller and the FMU time is needed. If a none-real-time simulation is performed, the event mapping component has to be able to halt the execution of the RTI upon reaching the next synchronization point. The event-mapping approach utilizing the event queues of the RTI also requires software interfaces between the event-mapping component and the RTI which provide access to the used queues and predict events or reset the state of the RTI. These additional software interfaces are not specified by the IEC 61499 [50] and have to be implemented in an RTI-specific way.

In contrast to the other general ways of interaction which were stated so far, the encapsulation of IEC 61499 applications into a model exchange FMU requires extensive modifications of the encapsulated RTI or an intermediate layer emulating RTI code. Some open-source RTIs are currently available which may be modified to support additional queue interfaces [37, 102]. Alternatively, an ISS which emulates the target device code may be used to decouple the RTI timing. Some ISS implementations already exist and may provide a basis for implementing the emulator component [72, 78].

Since the FMI 1.0 for model exchange specifies that only the memory management function given by the solver shall be used, existing software may violate the FMI 1.0 specification. In contrast to other light-weight FMUs, the use of existing, heavy-weight software may not be applicable to embedded platforms anyway which limits the impact of the standard violation drastically. It does not make much sense to implement an IEC 61499 simulation on an embedded platform that does not feature standard memory

management functions. Instead, the IEC 61499 RTI may be directly deployed. In this case, any functionality gained by the FMU may be much more efficiently implemented by the RTI itself. FMI 2.0 relaxes the memory management requirement by providing an optional capability flag.

3.4.7 Use Cases

The main use cases of the FMI encapsulation correspond to complex simulation and testing scenarios. By using an IEC 61499 FMU, the behavior of the controller may be integrated into complex plant models. When using an emulator which is executing the IEC 61499 RTI, it is expected to model timing very accurately down to single processor cycles. Especially when examining fast, transient phenomena, an emulation-based IEC 61499 FMU may give detailed results. On the other hand, the event queue-based model is expected to execute much faster due to the decreased overhead and the coarse-grained granularity making it feasible to simulate long operational periods.

In contrast to the presented co-simulation scenarios [88, 90, 91, 110], an IEC 61499 FMU does not require the including tool to maintain an application-specific interface. Also, timings may be modeled much more accurately, compared to static delay-based approaches. However, implementing an IEC 61499 FMU introduces considerable initial development effort and run-time overhead by frequently triggering FMI events. In simulation scenarios which include a large amount of equally programmed IEC 61499 application instances, an IEC 61499 FMU may reduce the implementation effort by avoiding manual instantiation of loosely coupled IEC 61499 controllers. It is expected that the encapsulation of IEC 61499 applications in an FMU also enhances re-usability between different models and different tools. A properly packed FMU can be imported in various tools without establishing a tool-specific connection.

3.5 Encapsulating IEC 61499 Applications in Co-Simulation FMUs

3.5.1 System Architecture and Model

The FMI for co-simulation requires an FMU to be able to stop its simulation at arbitrary communication points [38, p.16ff]. Like the FMI for model exchange, a co-simulation IEC 61499 FMU has to halt the RTI operation. In contrast to the FMI for model exchange, halting does not necessarily occur if an IEC 61499 event is triggered but on previously defined time instants. An emulator which is executing the RTI code or an adapted version of the RTI is needed to control the execution time of the RTI. The resulting FMI-compliant system model which is illustrated in Figure 3.12 is very similar to the system model described in Section 3.4.1.

The execution control component stops the execution of the RTI if a communication point is reached or further execution is not reasonable. It also restarts the execution if the

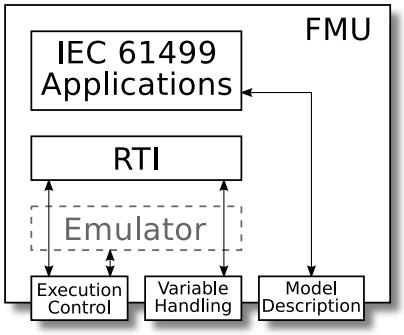


Figure 3.12: Basic co-simulation IEC 61499 FMU

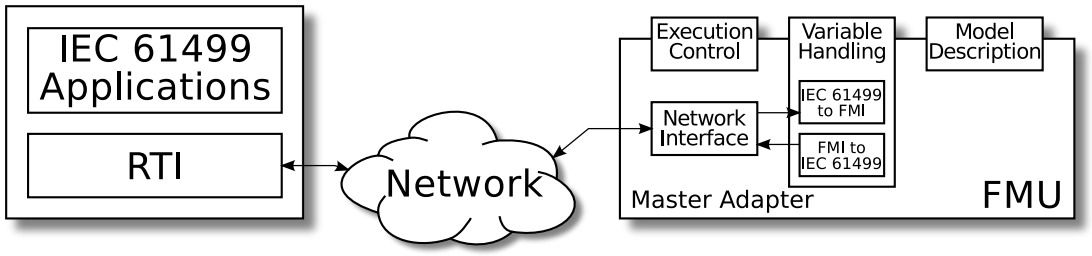


Figure 3.13: Standard communication facility-based co-simulation FMU

simulation master requests the next simulation step. The variable handling component interpolates exposed variables between two consecutive IEC 61499 events or between two consecutive communication points. Depending on the used event handling approach, it also has to implement a virtual input concept which triggers IEC 61499 events according to the event mapping approach.

The FMI for co-simulation specification explicitly mentions a distributed co-simulation scenario connecting different co-simulation tools by a communication infrastructure [38, 40, p.14]. Also, the IEC 61499 specifies a distributed infrastructure and CFBs accessing remote services [50, p.47f]. The system model which is illustrated in Figure 3.13 introduces a simple master adapter which connects a co-simulation master and an IEC 61499 infrastructure by using control-specific communication facilities. Similar topologies can be found in various co-simulation frameworks and standards such as the HLA [15, 38, 40, 51, 107], but the architecture specifically assumes the use of control network facilities. On utilizing the control network facilities, the IEC 61499 infrastructure does not have to be encapsulated into an FMU and may be deployed by common IEC 61499 configuration tools. Since the IEC 61499-based infrastructure generally does not maintain any simulation time different to real time, the master adapter does not fully conform to the FMI specification and can only approximate the co-simulation computation flow.

It is easily possible to include external hardware which is connected to the IEC 61499 controllers because the standard communication facility-based architecture operates in real time only. It is not necessary to modify conventional RTIs which reduces the implementation effort drastically. However, this simplicity causes decreased accuracy compared to a time-controlled approach. Whether the simple approach is able to provide adequate accuracy strongly depends on the implemented master algorithm and different timings between FMI function invocations. If the master algorithm increases the simulation time slower than real time, an accurate emulation is generally not feasible.

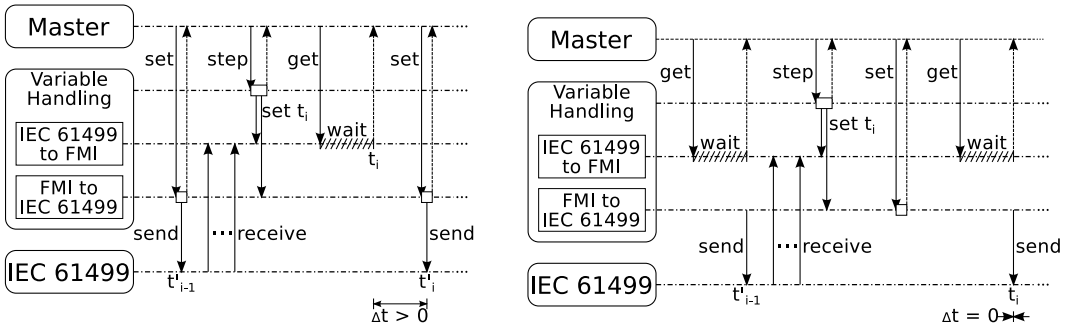
3.5.2 Event Handling

The event handling controlling the notion of time of an RTI requires the RTI to stop its operation at a communication point without further increasing its reference time. This can be done either by an emulating middleware that controls the RTI execution or by providing an RTI which is directly capable of stopping its execution. In contrast to model exchange FMUs, it is sufficient to stop the operation without being able to reset the state or to manipulate the event queue [38, 40, p.16]. Each FMI step function call executes the IEC 61499 system until the next communication point is reached.

To properly access values exposed by the FMI during communication points, the values have to be buffered by the variable handling component because an IEC 61499 event which queries values cannot be issued if the RTI currently halts. If the IEC 61499 infrastructure is executed again, the variable mapping component will accept output values by receiving IEC 61499 events. In an active event mapping strategy, the variable handling component has to trigger IEC 61499 events on its own, if a variable value set by the master algorithm or during interpolation changes significantly. Alternatively, it may implement a passive concept, providing the current variable value only on receipt of a requesting IEC 61499 event.

When using the standard communication facility-based architecture and a standard RTI, a non-blocking event handling strategy has to be used. The proposed event handling strategy does not require the controller to stop its execution and does not require the communication facility to transfer timed values. Following these constraints, the master adapter FMU actively has to control the system time as well as the sample time of exposed variables and transfer the sampled data to the IEC 61499 controller. The IEC 61499 system may either actively query values which are buffered by the master adapter or passively receive requests from the master adapter FMU which indicate value changes.

Each calculation needed to compute the result of the next step is implicitly done by executing the IEC 61499-based system. The `doStep` function only saves the next synchronization time t_i and returns immediately. If the step function blocks until the next communication point t_i is reached, the time between sampling the current state and setting new values, Δt_i, increases by the activity of the master algorithm between invoking the step function and accessing exposed FMU values. Figure 3.14 illustrates two possible event mapping sequences and locates the synchronization error Δt_i. Following



(a) Inexact synchronization (b) Exact synchronization

Figure 3.14: Standard communication facility-based event mapping

the FMI for model exchange specification, the simulation time has to stop during a communication point and $\Delta t_i = 0$. A synchronized sampling point cannot be guaranteed in a real-time operation but the event mapping strategy tries to approximate $\Delta t_i \ll T_\mathrm{a}$. To minimize Δt_i, the step function returns immediately and the first call to any getter function will block until t_i is reached.

The FMI for co-simulation does not specify the order of setter and getter function invocations and the implementation has to deal with every possible invocation order. On the one hand, if a value is set before a getter function has been called and t_i is reached, it has to buffer the value until t_i. Figure 3.14b shows that if the setter function is called before t_i, it will return immediately and sending is triggered at t_i without any delay. On the other hand, if a value is set after the communication point t_i, it has to be set immediately in order to avoid unnecessary delays. In Figure 3.14a the getter function has to delay until t_i is reached but no data was previously set. As soon as the data destined for the IEC 61499 applications is available, the variable mapping component will send the values.

To expose an output variable on a controller to a co-simulation master, the master adapter may either buffer any output value actively transmitted by the controller or request the output variable values on every synchronization point. The applicability of each alternative strongly depends on the used communication facility and the capabilities of the controller in decoupling network outputs from the control operation. If a controller is not able to store results and has to calculate the output upon request, the resulting timing may be strongly biased.

Different virtual in- and output schemes can be used to decouple synchronization points from the scheduling of the IEC 61499 application. Scheduling may either be triggered by significant input deviations or by timed events which are triggered inside an IEC 61499 application. In the first case, the master adapter keeps track of significant changes and actively initiates a transmission. The second case requires a passive master adapter returning previously set values on request. It is important to notice that in both cases the event time does not necessarily correspond to any communication point.

At each step, getter and setter function take a certain amount of time to execute. Limited by the execution times, a step size smaller than the accumulated execution time is not feasible. As a consequence, zero-size steps for event handling and step rejection are not supported. However, other optional features like variable communication step sizes and input interpolation can be implemented without restricting the real-time execution.

3.5.3 Data Model Transformation

Transforming variable types and unit definitions is very similar to FMI for model exchange. Especially for a dedicated co-simulation RTI which encapsulates and executes the system description, the same model variable transformations apply. Hence, the findings which are stated in Section 3.4.3 will not be repeated. Instead, the following section highlights important differences in transforming the IEC 61499 compliant system description to an FMU for co-simulation.

When using a standard communication facility-based interaction, one may not be able to automatically differentiate communication SIFBs which are used by the FMU for co-simulation and communication SIFBs which send messages to external components. Although the address property of a communication FB may hint the usage, no certain mapping can be achieved. Additionally, the external system description may no be part of the FMU itself. Separating the FMU implementation and the IEC 61499-based system description may ease re-usability of a certain FMU. Hence, user input is required to generate and configure exposed variables of the co-simulation FMU.

FMI for co-simulation adds optional support of supplying derivatives of model variables at communication points. An event mapping component may utilize or even generate derivatives to interpolate continuous IEC 61499 inputs and external continuous signals. Nevertheless, the maximum order of derivatives has to be encoded in the model description. The maximum order may depend on the actual application and may have to be provided by the user before generating the IEC 61499 FMU. Dynamic executable code may have to be generated according to the maximum order of derivatives, but the model description is not affected otherwise.

An encapsulated RTI which uses communication facilities may require additional parameters such as network addresses to run in a co-simulation. The parameters of an FMU may be automatically extended to take these run-time parameters into account. Since exposed model variables are hard-coded into the model description, no generic FMU which handles all types of systems can be generated. At least exposed variables need to be set in the FMU on demand. Nevertheless, storing the IEC 61499-based system description may still be beneficial because often communication interfaces are more stable than the control logic implementation.

3.5.4 Real-time Operation

Similar to the FMI for model exchange which is described in Section 3.4.4, the FMI for co-simulation does not specify any real-time requirements or real-time parameters [38,

40]. As a consequence, hard real-time operation covering every master algorithm is not possible and even a soft real-time implementation suffers from strong limitations and possible inaccuracies. The first event mapping approach relies on halting the execution of a system for every communication point. During the system halts, simulation time is considered to be constant and no event can be processed during the real-time instants used for communication. Every event e_k which is scheduled at a real-time instant reserved for communication violates the real-time constraint (3.18) given in Section 3.4.4. Unless the application excludes events during communication points, the application will generally not be able to trigger every event in real time. Additionally, the emulating component and the modified RTI, respectively, add another layer of complexity which complicates triggering events in real time.

The standard communication facility-based approach relies on a real-time operation of involved controllers but also suffers from similar synchronization inaccuracies. Real-time operation of this approach is defined by issuing every communication point t_i in time. To issue a communication point in time, data passed on to the IEC 61499-based system has to be known before t_i occurs, and data sampled at t_i must be passed on to the simulation master. Especially the order of the getter and setter function invocations determines the timely accuracy of the analyzed approach. If the setter functions are called before any getter function is triggered, the value passed on to the IEC 61499 applications may be buffered and issued correctly. In this case, the timely synchronization error Δt_i is determined by the implementation of the master adapter and may be approximated with $\Delta t_i \approx 0$. However, if the master algorithm inverts the function call order, the results of the IEC 61499 applications have to be returned before new data can be set. At an inverted call order, the synchronization inaccuracy is mainly determined by the time between the getter and the setter function invocation; the inputs at the controller are applied late.

If the execution of the co-simulation does not gain any results before the real-time instance of the next communication point occurs, the system may be sampled without any getter function call. In this case, the additional delay adds to the synchronization inaccuracy Δt_i and a buffer has to be implemented which holds previously sampled results. To maintain the sampling scheme which is specified by the co-simulation FMI and to minimize the synchronization error, it is important that the master algorithm and other included co-simulation components finish the simulation step before t_i. Any late result will be biased and therefore be of limited use even in a soft real-time application.

Providing hard real-time guarantees is not possible in general and practically not feasible, but a best-effort operation may provide adequate results. To produce resilient results, the synchronization error as well as any buffer overflow due to consecutive deadline misses should be recorded and evaluated. Only if synchronization errors can be considered small compared to the sampling size, the real-time approach will be feasible.

The communication facility introduces additional delays and jitters which may influence the simulation output. If the communication facility does not support a timed transmission or time-stamped values, its timing will have to be considered as a part of the control

operation. Including the communication timing may on the one hand increase the simulation accuracy if the master adapter behaves like a remotely added in- and output device. On the other hand, network delays may not be well controllable and add a source of uncertainty. Only if a timed operation is possible, the master adapter will be able to operate without an own clock and the communication timing can be neglected [55].

3.5.5 Tool Coupling

The communication between different tools within an FMI co-simulation setup is handled by the FMI master. Each FMU only has to maintain the connection to the coupled tool or simulator [38]. The first event mapping approach accessing a modified RTI or an emulator may either directly include the code of the tool into the FMU or maintain a connection to an external implementation. Encapsulating the whole RTI may ease the FMU deployment by eliminating the need for maintaining a separate tool installation, but it also restricts the system integration of the tool. An FMU-encapsulated tool may only use files contained in the archive and may not be able to modify system parts which require a high level of privileges.

The standard communication facility-based approach assumes a distributed scenario and may use various communication media to couple the IEC 61499-based RTI. IEC 61499 only defines communication via an ASN.1-based protocol [50] but available RTIs implement different network protocols such as Modbus TCP or MQTT as well [37]. Compatibility to different existing RTIs will be enhanced, if an existing protocol is used instead of creating a new one. By relying on implemented protocols, the RTI can be used out of the box without implementing any communication facility on the controller side. Contrary, a tailored network protocol or communication facility such as shared memory may reduce the impact on the simulation results by providing a time-stamped transmission and less communication overhead.

3.5.6 Software Interfaces

Like for the model exchange FMI described in Section 3.4.6, co-simulation IEC 61499 FMUs following the stopping-based approach also require a modified RTI or an emulator which executes the RTI. Exposed model variables may be accessed in a very similar way but the execution control can be implemented on a coarse level. The FMU only has to stop the operation of the controllers if a communication point is reached and does not need to take single events into account. Although the approach also needs a communication channel which transfers execution control information, it can be much simpler and modifications on the RTI or an emulator may be limited.

A master adapter which accesses a standard communication facility does not have to deal with any IEC 61499 RTI directly but only implements the communication protocol. The communication mapping between the FMI interface and the communication facility highly depends on the communication paradigm of the protocol. In case of an ASN.1 over TCP/IP-based solution, the master adapter may either implement a passive server

or an active client application [50]. A passive master adapter which implements the server listens until a client controller connects to it and the server accesses exposed variables buffered within the master adapter only on client requests. An actively acting master adapter on the other hand connects to a server provided by the controller and queries values at every communication point. It reduces the network load by requesting only needed values but requires the controller to decouple its operation from incoming communication requests. If the controller only schedules FB execution on incoming requests, the resulting timing depends on the communication points of the FMI and may not reflect field operation.

Some software libraries which implement different network protocols are already available and may support the development process of a master adapter. In particular, many operating systems already implement a TCP/IP network stack [54, 111], some fieldbus protocol implementations are available [60, 65, 106], and ASN.1 libraries and compilers can be used to support the development of ASN.1-based protocols [53, 105]. Considerations taken in Section 3.2.6 regarding the usefulness of ODE solving libraries also apply to communication libraries. It is still important to check supported features, licenses, development statuses and supported environments before using an external software library.

3.5.7 Use Cases

On using a co-simulation IEC 61499 FMU, control aspects can be included in co-simulations which cover multiple simulation tools. In such a co-simulation setup, the IEC 61499 FMU models control aspects like process automation based on an IEC 61499-compliant control system description. On successfully testing the control scheme, it can be deployed on the targeted infrastructure including only minor changes of interfaces. The common XML format specified by IEC 61499 eliminates the need of manually transforming the application model into the input format of the controller. In contrast to the co-simulation approach using a master algorithm which is embedded in a controller, the IEC 61499 FMU-based approach features an external master algorithm which may adapt to the other co-simulation tool capabilities.

Possible use cases of an IEC 61499 FMU for co-simulation include testing and validation scenarios which integrate multiple co-simulation tools coupled by an external master algorithm. The stopping-based approach focuses on scenarios requiring a detailed view on the timely behavior of the controller. If the co-simulation focuses on transient activity with time constants near the control response time, it is expected that the stopping-based approach will deliver more accurate results. If the stopping time of the model is short compared to a communication step and it is feasible to run the simulation in real time, the standard communication facility-based approach will entangle a simpler implementation and the availability of standard components.

For HIL setups using an IEC 61499 FMU for co-simulation, only the presented real-time-capable approach is feasible. It is important to notice that the approach does not

provide any real-time guarantees at all and is therefore not well-suited for safety-critical applications. Any safety-critical application must implement safety measures which avoid any harm, in case that the FMU does not deliver correct results in time. Controlling, for example, a HIL-based experiment is up to the external master algorithm which may be embedded into a comprehensive scientific workflow. Such a workflow may not only include the experiment itself but also some setup activities and some post-experiment evaluations.

3.6 Comparison

Tables 3.1 to 3.3 summarize different event handling, model and tool coupling approaches which are described in Sections 3.2 to 3.5. The tables further indicate strengths and weaknesses of the approaches by scoring different factors. Scores are given from "+ +" corresponding to the best expected results to "- -" corresponding to the worst. Since objectively measuring discussed criteria lies far beyond the scope of this thesis, only a rough estimation based on previous insights can be given.

3.6.1 Event Handling Approaches

The periodic event handling approaches described in Table 3.1 sample the state of the FMU and exchange information at discrete points in time only. The prediction-based event handling approaches try to estimate the next FMU event occurrence in order to issue IEC 61499 events in time. The internal event propagation described in Section 3.4.2 triggers an FMI event on every occurrence of an internal IEC 61499 event. Interface event mapping does not propagate every internal event but triggers an FMI event on every IEC 61499 event passed on to the FMI. The stopping-based co-simulation approach which halts the system on every communication point and the communication-based approach which utilizes standard infrastructure are both described in Section 3.5.2 and are considered in Table 3.1.

HIL setups require the system to include external hardware driving the HuT. As stated in Table 3.1, not every event handling approach supports the use of external equipment. The encapsulation of IEC 61499-based controllers into a model exchange FMU shields the controller and sets the execution time of the controller according to the synchronized simulation time. Both features are highly counterproductive in HIL setups. On the other hand, periodic event handling, especially when it includes model exchange FMUs, allows highly predictive timings which are controlled by the IEC 61499 application and may be successfully used in HIL setups.

The real-time operation column summarizes the ability of providing real-time guarantees and of running the IEC 61499-based controller in a real-time mode. Since real-time operation is a vital part in HIL setups, results highly correlate. Prediction-based event handling in co-simulation most likely will not achieve any hard real-time guarantees but may be operated by a best-effort strategy suitable for some HIL setups. In this case, the

Point of Interaction	HIL Setups	Real-time Operation	Accuracy	FMI Features	Software Support	Implementation Effort
Model Exchange in IEC 61499:						
Periodic Event Handling	+ +	+ +	-	+ +	+ +	+
Prediction-Based Event Handling	-	- -	+ +	+	+	-
Co-Simulation in IEC 61499:						
Periodic Event Handling	+	+	-	+ +	+ +	+ +
Prediction-Based Event Handling	-	- -	+	-	-	-
IEC 61499 in Model Exchange:						
Periodic Event Handling	-	-	-	-	+ +	-
Internal Event Propagation	- -	- -	+ +	+ +	- -	- -
Interface Event Propagation	- -	- -	+	+ +	-	-
IEC 61499 in Co-Simulation:						
Stopping-Based Integration	- -	- -	+	+	-	-
Communication-Based Integration	+	+	- -	- -	+ +	+ +

Table 3.1: Comparison of event handling approaches

Transformation Approach	Variable Types	Units	Interface Variables	Manual Intervention	Implementation Effort
FMI to IEC 61499:					
Type Restricted FMI	-	- -	+ +	+ +	+
Generally Typed FMI	+	-	+ +	+ +	- -
IEC 61499 to Model Exchange FMI:					
Type Restricted IEC 61499 Model Variables	-	- -	+	+ +	+
Generally Typed IEC 61499 Model Variables	+	- -	+ +	+ +	- -
IEC 61499 in Co-Simulation FMI:					
Direct Encapsulation	+	- -	+ +	+ +	+
Communication-Based Integration	- -	- -	-	-	+ +

Table 3.2: Comparison of automatic model transformation approaches

HIL capability is scored slightly better than the real-time capability which focuses on predictive timings.

Accuracy mostly focuses on errors introduced by the co-simulation and event mapping approach. Some approaches such as the prediction-based event mapping for model exchange communicate every relevant event at the time of its occurrence and do not introduce any additional error. Other approaches, such as the communication-based co-simulation, delay event transmission and introduce notable deviations. The communication-based approach only exchanges data at communication points and, depending on the master algorithm, introduces an additional timely inaccuracy at every synchronization point.

The FMI feature column weighs the number of optional FMI features and assumptions about the FMI operation which are necessary to implement the approach and which could be provided to external software, respectively. For example, the event propagation approaches for a model exchange encapsulation of IEC 61499-based controllers fully support the FMI specification without relying on frequent step-event emissions. The communication-based co-simulation approach only receives a low score in the FMI feature category because it requires the master algorithm to follow an invocation sequence which is not specified by the FMI standard.

The software support and implementation effort scores deal with additional software to be developed but do not cover any configuration details necessary to integrate FMUs. Software support summarizes the availability of tools and libraries which support the development process. For many approaches using FMUs in IEC 61499 applications, some libraries and tools supporting FMU handling already exist. Encapsulating IEC 61499 applications into FMUs mostly requires significant changes of the executing RTI and suffers from high implementation efforts.

3.6.2 Data Model Transformation Approaches

Automatic and semi-automatic model transformation approaches are briefly summarized in Table 3.2. Since the data models of FMI for model exchange and FMI for co-simulation are closely related in terms of model variable and type definitions, the integration of FMUs in IEC 61499-based applications is covered by one entry. Type-restricted coupling refers to transformation strategies where the set of allowed data types which are used in an interface is restricted to common types only. Casting may have to be done manually within the model or application. Contrary, generally typed approaches try to transform and generate custom data types whenever feasible. The direct encapsulation of IEC 61499-based systems in FMUs for co-simulation includes the RTI into the FMU and allows a more extensive model transformation. Communication-based integration, in contrast, restricts interfaces to general communication interfaces only and therefore limits model transformation.

The transformation of data types is always limited by FMI type definition facilities. Especially on using standard communication facilities, types are bound to supported protocols. Nevertheless, generally typed approaches may exploit existing facilities to

preserve as much information as possible. Similarly, the transformation of units is strongly limited by the lack of IEC 61499 unit support. A generally typed approach may encode units into custom IEC 61499 type definitions, but native unit support is not achieved.

The interface variable column of Table 3.2 rates the level of automatic variable transformation support. It is thereby assumed that the interface of the included system is transformed to the including system. For instance, the model variables of an FMU are transformed to FB variables in an IEC 61499-based system. Since the FMI clearly defines all variables and introduces a variable naming convention for structural data types, transformation of interface variables is generally well-supported. Only in a communication-based integration an IEC 61499 system description may not be fully available to generate an FMU for co-simulation. Additionally, the set of relevant communication FBs may not be easily identified.

Some transformation approaches may require manual user intervention which limits the applicability of an automatic transformation. For instance, the communication-based approach requires a manual identification of relevant FB definitions to generate a set of model variables. Contrary, other approaches support automatic detection of interface variables in case dedicated FB types are used. Hence, automatic transformation is considered to be generally well-supported.

The implementation effort in Table 3.2 considers the effort of implementing metadata transformation routines only. Tool-specific interface implementations such as implementing a custom RTI are not covered. Since generally typed approaches require management of custom types, implementation effort is considered to be high in comparison to type restricted approaches. Due to the limited support of automatic transformation, the implementation effort of a communication-based integration is limited to variable identification and therefore considerably lower than for other approaches.

3.6.3 Model And Tool Coupling Approaches

Table 3.3 summarizes different approaches to use multiple FMUs or to distribute an IEC 61499-based system to a set of FMUs. When using a set of connected FMUs within an IEC 61499 application, they may either be coupled by the interface logic implemented in the RTI or by IEC 61499 FB connections. When using IEC 61499 FB connections, the interface implementation which accesses single FMUs becomes less complex but the configuration effort which is necessary to integrate the FMUs in the application rises. An IEC 61499-based control system can either be encapsulated in a single FMU or distributed among several FMUs which are connected by a solver or master algorithm. In case of a distributed scenario, the connection information contained within an IEC 61499-compliant description has to be synchronized with the connection information which is maintained by the simulation master or solving tool.

The first evaluated criterion is the ability to resolve algebraic loops. It only applies to IEC 61499 applications including some FMUs because the resolution of algebraic loops is up to the master algorithm and the solver, respectively. For the encapsulation of

Point of Interaction	Algebraic Loops	Utilization of IEC 61499 Connections	Integration Effort	Implementation Effort
Model Exchange in IEC 61499:				
IEC 61499-Based FMU Coupling	-	+ +	- -	+ +
External FMU Coupling	+ +	- -	+ +	-
Co-Simulation in IEC 61499:				
IEC 61499-Based FMU Coupling	-	+ +	-	+ +
External FMU Coupling	+ +	- -	+	-
IEC 61499 in Model Exchange:				
Multiple Devices Per FMU		+ +	+	- -
One Device Per FMU		-	- -	- -
IEC 61499 in Co-Simulation:				
Multiple Devices Per FMU		+ +	+	-
One Device Per FMU		-	- -	- -

Table 3.3: Comparison of model and tool coupling approaches

IEC 61499 applications into FMUs, no scores are given. IEC 61499-based tool coupling requires the IEC 61499 application to implement an algorithm which solves algebraic loops. Although it is not impossible to implement a stable algorithm which solves equations by means of IEC 61499, it is most likely more tedious than using a service interface capable of connecting multiple FMUs.

The IEC 61499 already defines a mechanism to specify connections between FBs. This mechanism may also be used to specify connections between different included FMUs. Additionally, it might be necessary to preserve the device connection structure if an IEC 61499 application is distributed among several FMUs. The corresponding score illustrates the ability of utilizing this kind of connection information. In case coupling of multiple FMUs is directly implemented by an IEC 61499 application, no further connection input is needed.

The integration effort score corresponds to the work which is necessary to integrate a set of FMUs into an IEC 61499 application and to encapsulate an IEC 61499-based system into FMUs. External model exchange FMU coupling ideally only requires to include a proper SIFB. Coupling multiple FMUs by means of IEC 61499 requires a much higher integration effort such as manual consideration of algebraic loops to gain an equally good result.

Similar to the implementation effort score provided for event handling approaches, the implementation effort of different model and tool coupling approaches is estimated. Since no standard software is available which encapsulates IEC 61499 applications into FMUs, the implementation effort of these approaches is expected to be considerably higher than that of other approaches. If the FMI wrapper which includes FMUs into an IEC 61499 application does not have to deal with connection information, a decreased implementation effort will be expected but integration work will increase.

No single approach summarized in Tables 3.1 and 3.3 outperforms every other approach. The actual choice mainly depends on the use case and required accuracy. For example, some use cases require real-time execution and tolerate a decreased precision while others strongly focus on most accurate results. Before selecting a point of integration, the use case should be specified and the scope of this use case has to be defined.

CHAPTER 4

Implementation

The previous Chapter 3 describes several orthogonal approaches in coupling IEC standard 61499-based controllers with FMI-compliant models and tools. The practical demonstration and evaluation of all approaches would be far beyond the scope of this thesis. Nevertheless, the value of the theoretical work is to be demonstrated by implementing and evaluating some of the major concepts. Findings of Chapter 3 are used to select, justify, and guide the implementation of a particular point of interaction. The chosen coupling strategy and its implementation are quantitatively evaluated and compared to the theoretical expectations.

4.1 Software Development Objectives

Two principal ways of interaction which feature virtual components in IEC 61499 applications were discussed in Sections 3.2 and 3.3. Either model exchange or co-simulation FMUs may be included in IEC 61499-based applications. One of the most commonly used and broadly studied coupling strategies is periodic synchronization [20, 26, 62, 63, 82, 101, 112]. Thereby, results and measurements are exchanged periodically between the virtual component and the automation infrastructure. In case some value changes in between two synchronization points, it is artificially delayed. Another coupling strategy which predicts future states and resets the model in case the predictive assumptions fail to hold is also analyzed in Chapter 3. Due to its novelty [68, 108], the real-time performance of the predictive approach was only studied in a very limited way [85, 86]. At the time of writing, no closed-loop interface component which implements the predictive approach is available. The chosen implementation should fully support both, the periodic and the predictive approach.

4.1.1 Coupling Requirements

Since FMUs for co-simulation impose several major restrictions in using the predictive approach, FMI for model exchange was chosen to interface virtual components. From the requirements regarding the coupling approach, two software development objectives are derived. First, an interface software must support and implement both, predictive and periodic synchronization. A user may choose the one which best suits her requirements before starting the virtual component. Secondly, the interface software must support the inclusion of FMI for model exchange version 1.0 and 2.0 compliant models. Since multiple FMUs may be coupled to a single model via external programs, the interface program does not need to couple multiple FMUs in a single program instance.

4.1.2 Timing Requirements

One of the results of Chapter 3 is that hard real-time operation of arbitrary FMUs is not feasible. Consequently, the interface component is to be designed as soft real-time program which operates in best effort. To judge the quality of simulation results, any timing deviation and delays according to the internal reference clock must be recorded. An interface program must log the expected time of each synchronization point as well as the time when the data is actually exchanged. Since data transfer takes a finite amount of time, it is advised to record the time which is required to distribute the data, too. Facilities must be provided to evaluate the timing records and to list achieved worst observed execution delays.

4.1.3 Numerical Integration Requirements

An interface tool which emulates components via FMI for model exchange must solve the models via a numerical integrator. Various numerical integration methods are available which handle different classes of models and which differ significantly in properties such as execution time predictability and precision [21]. For instance, a particular adaptive algorithm which dynamically adjusts the integration step size may solve problems which cannot be solved otherwise but imposes considerable timing uncertainties. It is not expected that a single integration method serves all targeted use cases. Therefore, an interface software must implement multiple integration methods from which an appropriate one can be chosen before starting the virtual component. One objective is to provide at least one method which supports adaptive step sizes and one which uses fixed step sizes only. A user must also be able to adjust major integration parameters such as step sizes and targeted precision.

4.1.4 Interface Requirements

Several options in integrating a virtual component into automation infrastructure exist. For instance, an RTI which directly solves included models or a dedicated real-time hardware which solves the model and provides simple analog IO interfaces may be deployed. Automation infrastructure often uses specialized communication facilities to link multiple

subsystems. Sensors, for example, are often connected via fieldbus connections to PLCs which implement the control logic. On using standardized communication infrastructure, even devices from multiple vendors may be deployed in one connected system. In order to reduce dependencies from specific vendors and RTIs, a stand-alone interface component design which connects to automation infrastructure via communication links is chosen. Since many FMU vendors only provide binary FMUs, the interface component must not rely on a dedicated real-time hardware. It is not expected that one communication protocol will be sufficient to serve all targeted use cases. Various industrial automation protocols co-exist and many devices only implement a few of them. Consequently, a versatile interface component which creates virtual components must be protocol-agnostic in a way that another communication protocol can be easily implemented. Nevertheless, to demonstrate the value of virtual components and the chosen coupling approach, only a single protocol needs to be actually implemented. One objective is that the interface component must at least implement the ASN.1-based communication protocols which are specified in IEC 61499. A user must be able to configure an arbitrary amount of connections and must be able to specify the mapping of exposed FMI variables to network connections. The interface component must be able to receive data from the automation infrastructure and to send the results from the model according to the chosen processing scheme. Hence, a closed-loop operation must be implemented.

4.1.5 Software Design Requirements

Special emphasis should be put on a flexible software design which eases maintenance and adoption to changed requirements. During the first efforts in developing an open-loop interface program, some design targets were stated [85]. In particular, it should be possible to make certain changes without re-factoring unrelated parts of the program. The following change cases were listed in [85]:

- Add additional configuration options to refine the behavior of the program.
- Use another predictive event source.
- Add new transmission protocol implementations.
- Transfer the management of real-time instants (the authoritative clock) to another connected device.
- Add various external event sources.

In addition to these change cases, the following cases have been identified and should be considered during the software design of the closed-loop interface component.

- Add another numerical integration method.

- Use an event source instead of the model as long as it is able to return future events and to process arbitrary incoming events which date between the last and the next predicted event.

To gain scientifically sound results, it must be assured that the interface component correctly executes the specified workflow. To increase the confidence in a correct operation, quality assurance measures have to be taken. Each unit of the interface component has to be thoroughly tested via automated unit tests. Hence, testing aspects also have to be considered in the software design. A standard development process which includes issue management as well as source code and user documentation rules is to be implemented. Nevertheless, the detailed description of any development process is beyond the scope of a thesis which focuses on computational aspects.

4.2 Simulation Program Flow

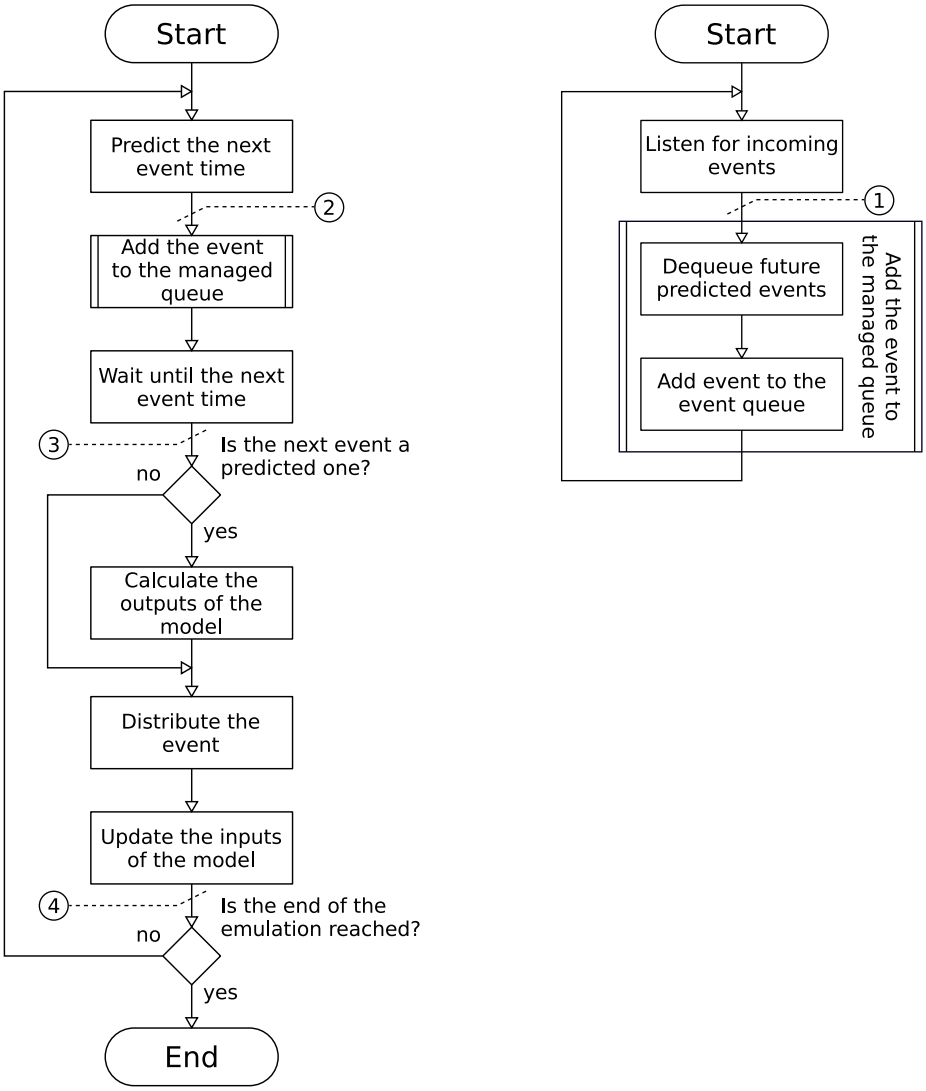
The specified interface program requires at least two modes of operation, one which executes a predictive approach and one which implements periodic synchronization. For each mode of operation, the main program flow is specified and analyzed. Processing steps such as parsing a user configuration, initializing a model, and error handling are omitted, in order to increase readability. Processing a possibly continuous model is abstracted in a single step which returns the next prediction or the state at the next synchronization point.

4.2.1 Predictive Program Flow

The basic program flow was already described in the open-loop implementation [85, 86]. Figure 4.1 shows the basic program flow and additionally highlights dedicated queue management tasks. Since the occurrence of external events is independent from the main program flow, it is divided into multiple concurrent tasks. Figure 4.1a describes the main program flow of the task which does the predictive steps, and Figure 4.1b shows the basic program flow of each concurrent receiver. A central event queue is used to decouple the main and the receiver program flow. Additionally, it avoids event loss in case the implemented best-effort approach fails to deliver all events in time.

Main Program

In each emulation step, the next event time is predicted based on the current knowledge of the system state. Note that only the next event time and the state before the event, not the final state after the event are initially predicted. The event itself is not executed in order to allow a reset operation of the included FMU. The predicted event is added to the queue according to the prediction time and the main program flow is delayed until the next event in the queue must be scheduled. The next event may either be the predicted event, which was inserted before, or an external event, which was added by a concurrent entity. In case a predicted event is returned, the final outputs still need to



(a) Main program flow (b) Event receiver program flow

Figure 4.1: Basic interface program flow [85]

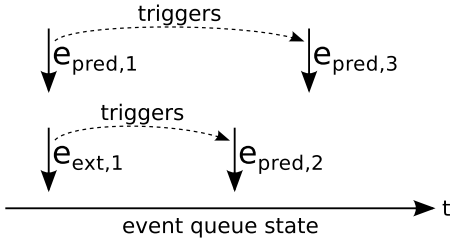


Figure 4.2: Concurrency issues of the basic program flow

be calculated. Since in this case the real-time instant of the predicted event is reached, it is now safe to fix the state of the model by executing the FMI event and querying the outputs. An external event does not require any additional processing step and the program can directly begin to distribute the event. Finally, the model is notified of any scheduled event and may need to update its input variables according to the received data. It may also be feasible to execute the input update step if and only if the event is an external one, but an unconditional update step was chosen in order to simplify the software design and to ease the implementation of other processing schemes.

Each predicted event is based on the assumption that no external event changes the system state in the prediction period. In case an external event is received prior to the next predicted event, the predicted event must be invalidated and removed from the queue. The queue management task is done in the threads which manage external receivers before adding a newly received event. Each concurrent receiver needs to terminate its operation as soon as the main thread stops the emulation. The termination of any receiver is intentionally omitted in Figure 4.1 in order to simplify the program flow diagram.

Important Corner Cases

Special attention has to be put on processing late external events, external events which are added before the previous prediction finishes, and concurrent event occurrences. Take, for instance, the situation in which an external event $e_{\text{ext},1}$ is to be scheduled at the same time instant as a predicted event $e_{\text{pred},1}$, i.e. $t(e_{\text{ext},1}) = t(e_{\text{pred},1})$. It is assumed that hardware outputs never directly depend on inputs from the software, i.e. the event $e_{\text{pred},1}$ never causes an instantaneous external event $e_{\text{ext},1}$. The assumption is justified by noting that external, digital hardware always takes a finite amount of time to process an FMI event. Consequently, both events do not causally depend on each other and may have associated data which need to be distributed. Figure 4.2 illustrates the occurrence of two simultaneous events and the resulting queue state.

The first event being scheduled always updates the state and calls the FMI event update function. On following the basic scheme, a new prediction $e_{\text{pred},2}$ is added to the queue. In the next iteration, the second concurrent event is processed and another prediction $e_{\text{pred},3}$ is added to the queue, although the previous prediction $e_{\text{pred},2}$ is not removed. Without further considerations, it may still be valid that more than one predicted event, e.g.

$e_{\text{pred},1}$ and $e_{\text{pred},2}$, is part of the event queue. Simply removing all predicted events from the queue on inserting another predicted event is not an option. In order to overcome the limitations of the basic interface program flow in [85, 86], the presented work is refined and border cases are explicitly handled.

Extended Queue Management

To gain consistency, each predicted event which has the same timestamp as an external event will be scheduled before the external event. Consequently, the model state is updated before the external model inputs are settled and the external event is processed. This policy also improves the real-time performance since only model events need to be scheduled in real time. Prioritizing predicted events also avoids that transient intermediate results from predictions that remain valid are lost.

In addition to a guaranteed event order, the specification of queue management functions is further refined. Instead of directly adding events to the queue, a cleanup and add procedure is performed. The correctness of the entire program flow relies on the following assumptions for scheduling events.

1. A predicted FMI event $e_{\text{pred},j}$ will be scheduled if and only if it depends on every event which has to be scheduled strictly before the predicted event. This condition uses two assumptions on the dependency structure. The first one is that all previous events possibly influence the outcome of the current prediction. Some predicted events may not depend on every previous event but the generalization allows a judgment without detailed knowledge of the included model. The second assumption is that any external event with the same time as a predicted one cannot be caused by the predicted one. Consequently, predicted events can be safely scheduled before concurrent external ones. In case another predicted event is directly caused by an external event, the predicted one will be scheduled after the external event is processed. This design decision may degrade real-time performance but avoids loss of important intermediate events.

2. Every external event $e_{\text{ext},i}$ will eventually be scheduled and used to update the model. In case it arrives late, it may be scheduled to a later instant of simulation time but it will be scheduled. The scheduling guarantee is needed to avoid data loss and to assure the correct state of the model.

3. In case the operation is not delayed and all real-time criteria are met, all valid events will be scheduled in the order of their associated time.

4. After an event has been added to the queue, the queue will never be empty. The basic program flow specifies a step in which the next event time is awaited. In case the event queue is empty, the next event time is undefined. Hence, it will be assured that at least one next event is available.

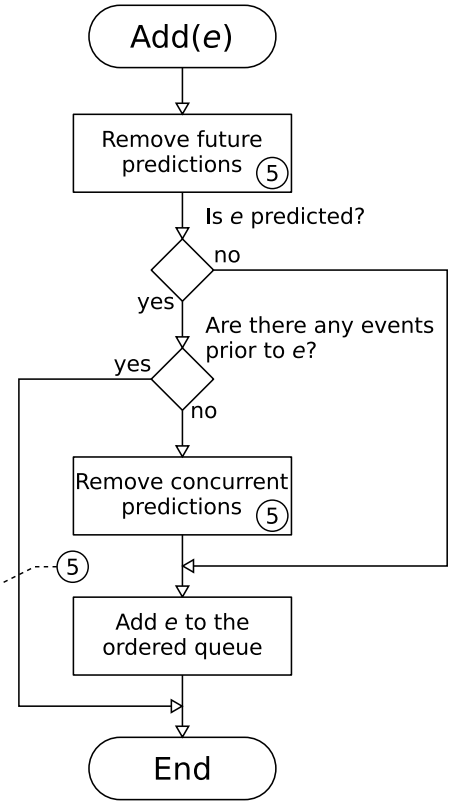


Figure 4.3: Program flow of the add operation

Figure 4.3 shows the program flow of adding an arbitrary event e to the queue. It combines both operations, a cleanup step and the actual step of adding the event to the queue. First, every predicted event having an event time strictly later than the time of the current event e will be removed from the queue. In case e is predicted and there are already some events strictly before $t(e)$ in the queue, e is considered outdated and will not be added to the queue. In case e is a predicted event and not already outdated, all other predictions which have the same timestamp as e will be removed from the queue before e is added. Finally, each external event is directly added to the queue.

Queue Management Implications

In the following discussion, the state of the event queue is represented by a finite sequence of events $(e_1, \ldots, e_m)$. An optional queue entry e is denoted by $[e,]$. To argue the first condition regarding the scheduling of predicted events, two invariants are introduced which hold before and after every queue operation. In particular, the content of the queue always follows the form

$$\begin{gathered}([e_{\text{pred},k},]\, e_{\text{ext},l_1}, e_{\text{ext},l_2}, \ldots, e_{\text{ext},l_n}) \coloneqq (e_1, \ldots, e_m) \\ \forall i : 1 \leq i < m, t(e_i) \leq t(e_{i+1}).\end{gathered} \tag{4.1}$$

The very first element e_1 of the ordered queue $(e_1, \ldots, e_m)$ may be a predicted event, all other elements must be external events. Additionally,

$e_{\text{pred},k}$ must depend on all events which have been scheduled before. (4.2)

Consequently, if there is a prediction $e_{\text{pred},k}$ in the queue, it must be the prediction which was conducted last. Otherwise there exists an event, namely the event which triggers the newer prediction, which is not considered in $e_{\text{pred},k}$ and hence, (4.2) is violated.

It is still open to show that the conditions are indeed invariants. For an empty event queue, as it is present just after initializing the program, the invariants (4.1) and (4.2) hold trivially. In case the invariants hold and the first event of the queue, e_1, is scheduled and removed from the queue, no predicted event will be present and the invariants hold for the new queue state, too. Assume that (4.1) and (4.2) are fulfilled and that an arbitrary event e is added to the queue. The event may either (1) be strictly earlier than any other event e_i in the queue, (2) equally timed to the first event e_1 or (3) it may be later than the first event. Assume case 1, $\forall 1 \leq i \leq m, t(e) < t(e_i)$. After the first step of the algorithm in Figure 4.3, any predicted event is removed from the queue and the queue is of the form

$$(e_{\text{ext},l_1}, e_{\text{ext},l_2}, \ldots, e_{\text{ext},l_n})$$

Since there are no events prior to e, the path omitting the add step is not feasible and e is added to the queue. Also, there are no more concurrent predictions which would have been removed, so that after adding the element the structure of the queue follows (4.3) and consequently (4.1) holds.

$$(e, e_{\text{ext},l_1}, e_{\text{ext},l_2}, \ldots, e_{\text{ext},l_n}) \tag{4.3}$$

Event e may be the only prediction in the queue. In case e is predicted, it follows from the basic program flow in Figure 4.1, that e directly depends on all previously scheduled events. Hence, both invariants are fulfilled in case 1.

Next, assume case 2, $\exists e_1 \wedge t(e) = t(e_1)$, i.e. the event e has the same time stamp as the very first event in the queue. Since there are no predicted events after e_1, the first step of the algorithm will not alter the queue. Furthermore, there are no elements strictly prior to e and e will be added to the queue. Assume that both events, e and e_1 are predictions. In this case, e will replace e_1 (e_1 will be removed and e will be added in place) and the invariants (4.1) and (4.2) are fulfilled. In case only e_1 is a predicted event, it will remain the last predicted event which takes every previously scheduled event into account and e will be inserted after e_1. If e is a predicted event and e_1 is not, e is the most recent prediction and e will be inserted before e_1. Consequently, the invariants (4.1) and (4.2) hold, too.

Last, assume case 3, $\exists e_1 \wedge t(e) \gneq t(e_1)$. In any case, no event will be dropped by the first step of the algorithm because the only prediction is earlier than e. Consequently, the invariants hold after the first step, too. If e is a predicted event, it will be immediately

dropped because there is at least one event e_1 prior to e. Consequently, the structure of the queue does not change and e is never scheduled. If e is not a predicted one, it will be inserted after e_1 and the structure condition (4.1) still holds. Since $e_{\text{pred},k}$ does not change, it still depends on all previously scheduled events. Hence, the invariant (4.2) is also fulfilled in case 3 and generally both invariants (4.1) and (4.2) will hold after the execution of the add procedure.

From the invariant, it can be trivially concluded that, if a predicted event $e_{\text{pred},j}$ is scheduled, it will depend on all previously scheduled events. In case a predicted event $e_{\text{pred},j}$ depends on all previously scheduled events, it will be generated and added to the queue just after all previous events have been scheduled. Otherwise it cannot incorporate all previous events. Assume that the queue is in an arbitrary state in which (4.1) holds and $e_{\text{pred},j}$ should be added to the queue. Since $e_{\text{pred},j}$ is added after all previous events have been scheduled, there are no events prior to $e_{\text{pred},j}$ in the queue. In case the first event of the queue is a predicted one, it will be replaced by $e_{\text{pred},j}$, and $e_{\text{pred},j}$ will be inserted at the very first position. By the assumption that all events which should be scheduled before $e_{\text{pred},j}$ are already scheduled, and the assumption that the queue is ordered, every event which is added after $e_{\text{pred},j}$ will be inserted after the current prediction. Consequently, $e_{\text{pred},j}$ will eventually be scheduled by the main program flow and the first queue management condition holds.

Every operation in the add function which deletes an event before it is scheduled exclusively deletes predicted events. External events cannot be removed from the queue without scheduling them. On assuming progress on simulation time – there is still the possibility that only events with the same simulation time instant are added and no progress is observed – every external event will eventually be scheduled. From the fact that (4.1) holds, the queue will always be ordered according to the simulation time. In case the real-time operation is not delayed, each event will be scheduled in the order of its simulation time. Note that for equally timed events, in general, no order is defined.

Finally, have a look at the last condition which states that a queue never becomes empty on executing the add workflow. The queue may only be empty if the current event e is not added, which leaves a single execution path. Consequently, e must be predicted and there are some events prior to e after removing future predictions. This directly contradicts the assumption that queue is empty afterwards and it can be concluded that the queue cannot become empty by calling the add function.

The discussion shows that the refined queue management functions meet the requirements which are imposed by the basic simulation flow and that special measures are needed to deal with concurrent events. In practice, concurrent events will drastically deteriorate the simulation performance and should be avoided if reasonably possible. It has to be noted that, for instance, models which directly output an event as soon as an input event occurs cannot be executed in strict real time. Thereby, the minimum delay condition would be violated and two events would have to be processed simultaneously. Nevertheless, achieved delays may be tolerable and the application may still be more responsive than other approaches.

4.2.2 Periodic Synchronization Program Flow

In order to feature a unified software design, the program flow of the periodic synchronization approach strongly relies on the program flow of the predictive approach. A central event queue is used to manage both, FMI and external events, although an operation without queuing FMI events may be feasible. Thereby, code implementing the queue and real-time management can be used for both simulation workflows and the objective of featuring both approaches is supported. Figure 4.4 shows the main program flow of the periodic synchronization approach. The program flow of all concurrent receivers is identical to the main program flow of the predictive approach in Figure 4.1b.

At the end of each synchronization period, an event which conveys the simulation results is triggered and added as a prediction to the managed queue. As a consequence of using the same queue implementation for both simulation approaches, a synchronization event may be dequeued in case an external event is received. Hence, it is deleted and must be added again until the event is finally scheduled. To keep the overhead of using the same event queue implementation within reasonable bounds, a simple caching mechanism is used to store the latest valid simulation result. At the beginning of each main iteration, it is checked whether a valid result is available. In case no results are present, the model is forwarded to the next synchronization point and the outputs are directly calculated and cached. Direct output evaluation is feasible since the model will not have to be reset again and inputs will always be applied to the current state of the model. The synchronization period may either be chosen as fixed intervals or the first FMI event determines the end of a synchronization period. Note that none of the two methods allows a dynamic step size according to external events because no reset operation is to be implemented. However, the step size can optionally be chosen according to the next FMI event.

In a next step, the current synchronization event, either a newly calculated or a cached one, will be added to the event queue. For the add operation, the very same procedure as for the predictive approach is used. After the current event has been scheduled, it is distributed. In case an external event was obtained from the queue, any associated data is delayed until the end of the current synchronization period. Since the model is in exactly this state, external data can be directly applied without a reset operation. Any scheduled model event will be taken as an indicator that the simulation time is reached and that the model event is actually taken. Consequently, cached results are invalidated and the next synchronization period is going to be processed in the upcoming cycle.

In case an external event $e_{\text{ext},i}$ with $t(e_{\text{sync},j-1}) \leq t(e_{\text{ext},i}) < t(e_{\text{sync},j})$ is received after the synchronization event $e_{\text{sync},j}$ is scheduled, the model will not be reset either. Consequently, $e_{\text{ext},i}$ will not be applied at $t(e_{\text{sync},j})$ but on the next synchronization point, e.g. $t(e_{\text{sync},j+1})$. Since the queue always schedules synchronization events before external ones, external events which date to a synchronization point will be delayed by an entire synchronization period. The design decision of prioritizing synchronization events slightly decreases the worst case delay of a timely operation by the smallest representable time unit. Prioritizing

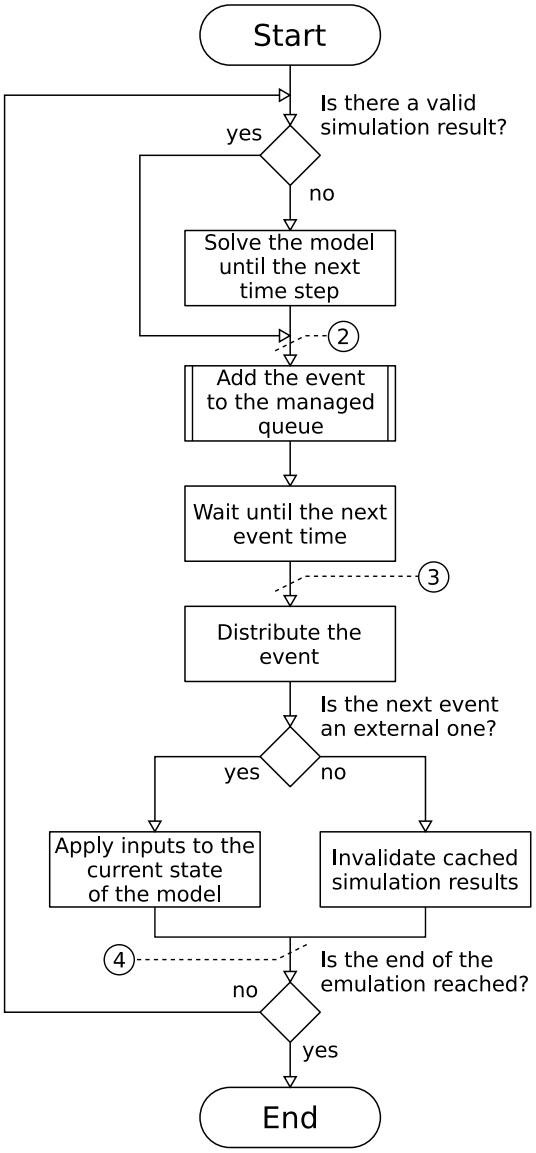


Figure 4.4: Main periodic synchronization program flow

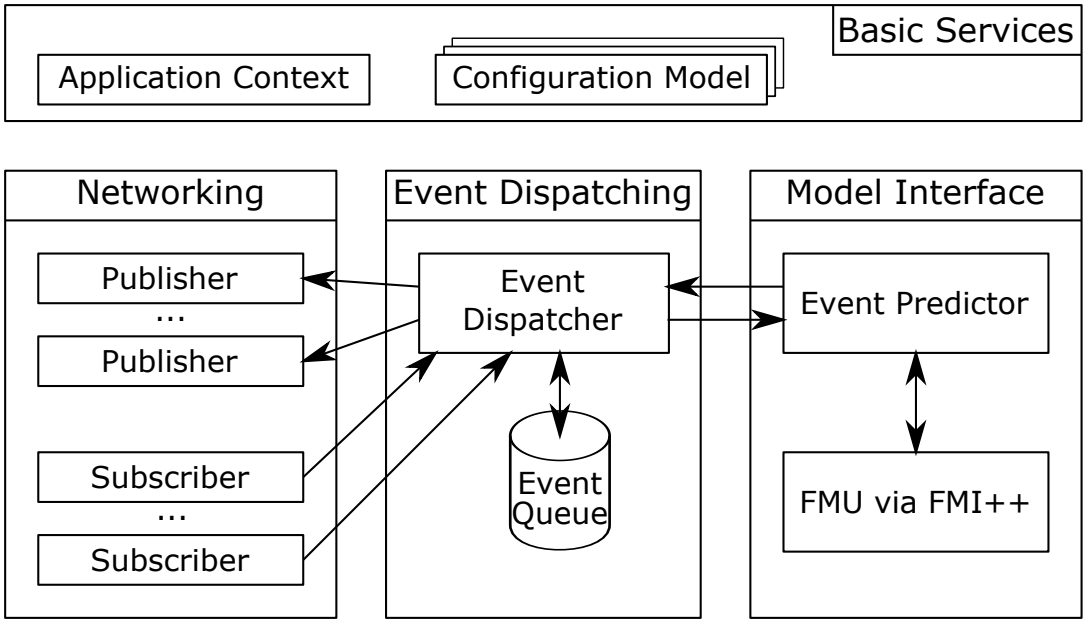


Figure 4.5: Basic program design [85]

external events in case of a concurrent synchronization event would allow to instantly reflect external data in the model, but an approach which unifies both emulation program flows is chosen. Termination of the emulation run is handled in the same way as in the predictive approach. At the end of each processing cycle, the current event time is evaluated and the emulation is terminated, if necessary.

4.3 Software Design

A basic design of a bidirectional FMI IEC 61499 interface program was already presented [85]. Nevertheless, only a one-directional communication was actually implemented. Due to extended objectives, several major redesign and refactoring steps were necessary. However, the basic program structure was kept and successfully extended. Figure 4.5 shows the basic program design which widely uses the existing structure [85].

The basic services provide functionality which is commonly used by all other building blocks. An application context object encapsulates the global configuration as a unified property tree. Furthermore, several utility classes form a compound configuration model which explicitly stores configuration options in a parsed format. The network stack is implemented as an independent unit which encapsulates all network-related functionality. Dedicated interfaces are used to receive and submit events. An event dispatching component synchronizes the progress of simulation time with the computer clock and organizes scheduling according to the policies of Section 4.2.

The model interface implements the generalized event prediction and solves the connected model. Note that, although the periodic synchronization approach does not use prediction in the strict sense, it is also accessed via generalized prediction software interfaces. The actual FMU is included and solved via the software library FMI++. Although some extensions to FMI++ were necessary to implement the interface component, the detailed design of the FMI++ library is beyond the scope of the thesis.

4.3.1 Basic Service Design

Basic services mainly target user configuration and extendability objectives. Although the services implement a configuration model which stores main aspects of the run-time configuration in dedicated objects, they do not directly implement application-specific functionality. E.g. several objects which encapsulate the intended structure of the network stack are implemented in the basic services but the actual data routing and networking functionality is implemented outside.

One of the main classes which are used to access most of the basic services is the ApplicationContext class. ApplicationContext was already introduced in [85] but substantially modified to support a closed-loop operation. Still, a unified property tree is used to store all user configuration options as well as various dynamically added content such as the name of the currently running executable. Since the property tree does not statically limit stored information, new configuration options may easily be implemented by querying the corresponding keys.

ApplicationContext also implements access functions which create the configuration model classes from the property structure on request. The configuration model classes provide several functions to directly list and access encoded information but also link to the corresponding branch in the property tree. Consequently, configuration options may be added without the need of directly extending the configuration objects or re-implementing the parsing logic. On using dedicated configuration objects, such as transmission channel configurations, duplicate code for parsing the user configuration is avoided and common functionality is hosted by the basic services. Figure 4.6 illustrates the structure of the main configuration objects.

A ChannelMapping object stores all the data routing information and associates FMI model variables and the corresponding network variables, called ports. The network implementation supports multiple independent transmission channels which group an arbitrary number of network ports. One transmission channel implements exactly one protocol and is intended to have one end point. E.g. one transmission channel connects to a single Transmission Control Protocol (TCP) / Internet Protocol (IP) endpoint and sends one static set of model variables which are encoded via IEC 61499 ASN.1. In case an event is emitted, associated data of each network port is combined to a single message. A protocol implementation may dynamically query associated properties to configure its operation.

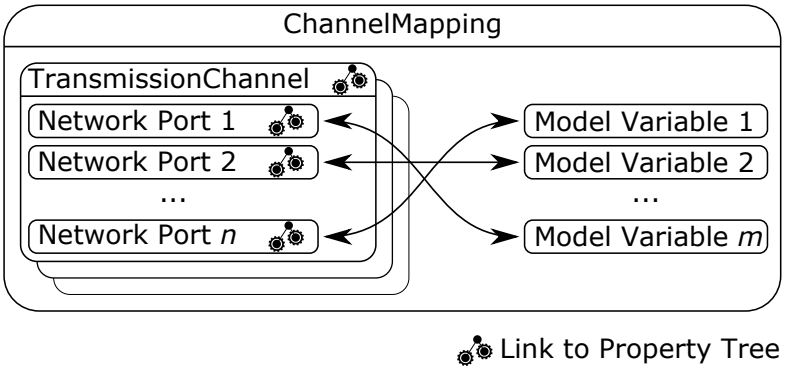


Figure 4.6: Channel mapping configuration objects

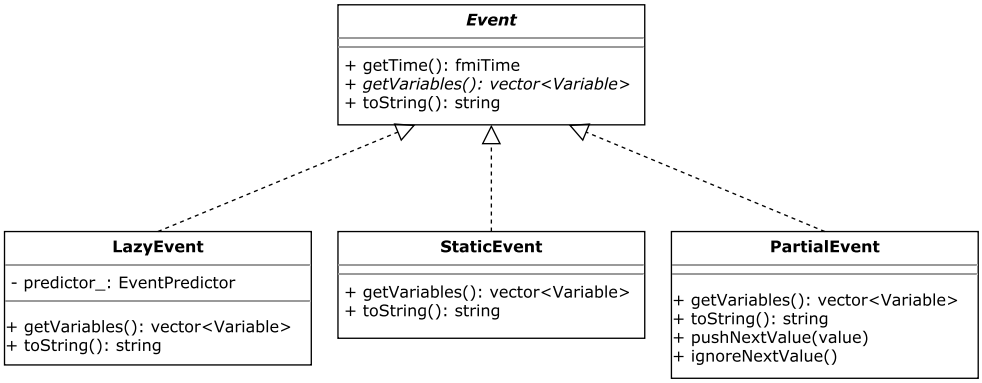


Figure 4.7: Event class hierarchy

Each model variable is identified by an internal identifier. The identifier combines the FMI data type and an arbitrary integer. Since early versions of FMI++ did not expose the FMI value references, the used variable identifier does not correspond to this reference. Each ChannelMapping object stores the association of model variable names and the internal identifier. The generalized event predictor queries this information and uses it to properly access the included FMU. FMI also defines unique model variable names which may also be used to internally identify the model variable. Within the simulation flow, variable identifiers are frequently used to label associated data. To avoid the overhead of human-readable identifiers, a numeric identifier was introduced, and it was chosen not to utilize the variable names to directly label data.

4.3.2 Event Queuing and Dispatching Design

The event queuing and dispatching components implement the main program flow. Data exchange and scheduling is entirely based on timed events. Therefore, an abstract event class was defined which stores the time of the event and defines an abstract function which queries associated data. Figure 4.7 outlines the event class and the major child

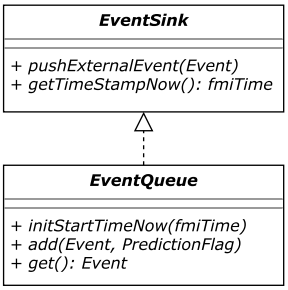


Figure 4.8: Event queue and event sink interface

classes. Data is stored in terms of an arbitrarily sized vector of variables. Each variable carries the unique identifier which is defined by the basic services and a `boost::any` field which actually hosts the data.

An event implementation may either statically store associated data or query the model outputs on request. StaticEvent and PartialEvent both implement the first storage approach which embeds associated data before the event is added to the queue. In contrast to StaticEvent, PartialEvent allows to sequentially populate an event, in case not all variables are instantly available. For instance, a network protocol implementation utilizes PartialEvent to intermediately store external events which are not entirely received. Still, PartialEvent does not allow to calculate the model outputs on request. In contrast, LazyEvent implements the second approach and does not require all outputs to be present when added to the queue. As soon as the data is requested, LazyEvent queries the event predictor and returns the model outputs. Since the event dispatching component does not query any variables before the event is actually scheduled, the model may still be able to reset the state in case a LazyEvent instance is dropped.

Scheduled events are distributed via an event listener interface, as shown in Figure 4.9, which simply defines an abstract function to receive events. The event dispatcher object maintains a list of registered event listener instances and distributes scheduled events among them. Additionally, it returns an event sink instance which may be used to register external events. Internally, the event dispatcher maintains an event queue instance which implements the queuing and real-time logic. Figure 4.8 shows the event sink and event queue interfaces. In case another scheduling mechanism, such as external real-time clocks, should be used, an abstract queue interface may be implemented, and the currently used queue implementation can easily be exchanged.

The event dispatcher object expects an AbstractEventPredictor instance which is then used to generate predicted events. Consequently, alternative event predictor implementations can be used as long as they are able to receive scheduled events and return the upcoming event upon request. The abstraction of all major functions does not only support defined change cases but also allows to implement dedicated testing facilities. For instance, an event predictor which simply returns equidistant events is used to test the event dispatcher and scheduling implementation.

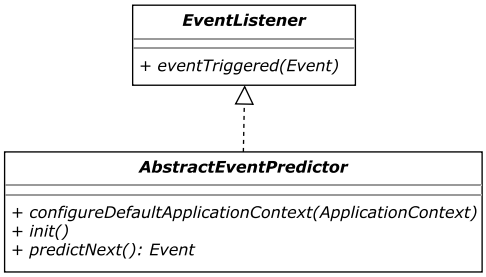


Figure 4.9: Abstract event predictor

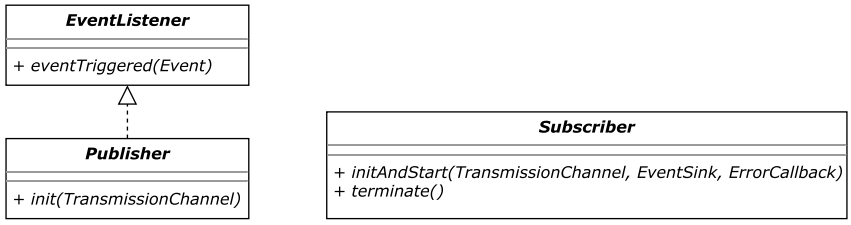
To reach the real-time logging objective, a dedicated event timing logger was implemented. An event dispatcher as well as an event queue use the event logger to record the current timing status at various processing stages. The logger queries the current real-time instant from the operating system and stores it in a dedicated timing file. In order to keep the run-time overhead as low as possible, delay and delay distribution analysis are done in post-processing steps and are not directly implemented in the interface program.

An additional event data logger is used to record all variables associated with a certain event. In contrast to the event timing logger, the event data logger directly uses the event listener mechanism to receive all scheduled events. The data logging functionality can be loosely coupled to the other components without the need for a tight integration into the queuing and dispatching facilities. As a consequence, recorded data only contains the simulation time and does not directly refer to real-time parameters. Like the timing file, the logged data file does only contain variable data associated with a certain event and no further processing is implemented in the interface component. Post-processing may be used to interpolate values which are not directly associated with a certain event and to filter events which do not change the state of any variable.

4.3.3 Model Interface Design

The model interface is used to access and solve FMUs. Each approach is implemented in a dedicated abstract event predictor class which interfaces the FMU via FMI++. Figure 4.9 illustrates the interface design of an event predictor. A factory class instantiates the event predictor according to the user's choice. The actual FMI++ interface depends on the emulation approach. Periodic synchronization accesses a wrapper class which provides integration facilities but does not provide a reset mechanism. The predictive event source uses another FMI++ layer which stores intermediate predictions and implements the reset operation. Via FMI++, both FMI versions can be handled transparently and version-specific code can be reduced to a minimum.

Since FMI++ is used to implement the computational details of solving a model and predicting future states, the logic of each abstract event predictor focuses on managing the prediction program flow and converting the internal data representation into a format which can be handled by FMI++. Shared functionality, such as loading an FMU



(a) Publisher interface (b) Subscriber interface

Figure 4.10: Main networking interfaces

according to the user configuration, is implemented via dedicated helper classes. FMI++ allows to dynamically change the deployed numerical integration algorithm as well as major parameters thereof. Among others, it implements a variable-step Runge-Kutta method (Dormand-Prince) as well as a fourth-order Runge-Kutta method with constant step size [96, 97]. A solver configuration class evaluates the user configuration and provides the integrator parameters to fulfill the corresponding design objectives.

Since most numerical aspects are handled by FMI++, the change case of implementing another integration algorithm either needs to be directly tackled in FMI++ or implemented via another predictor instance. FMI++ uses dedicated integrator interfaces [96] which may be used to implement another integrator. The change case of an alternative event source is directly addressed by the abstract event predictor interface. Each event source which follows the specification can be easily integrated into the interface component.

4.3.4 Network Design

The implementation of the network functionality is mainly organized via two independent interfaces, a publisher and a subscriber interface, which are illustrated in Figure 4.10. Each publisher receives scheduled events, selects the events according to the configuration objects and distributes the data according to the implemented protocol. A subscriber manages incoming data and registers external events via an event sink interface. Two life-cycle functions, a start and a terminate function, are used to control the possibly concurrent operation of a subscriber. Since each subscriber operates independently and registers events autonomously, it does not need to provide any other interface function. A network manager class controls the life cycle of each publisher and subscriber. In particular, it creates the concrete objects which implement a specific protocol according to the user configuration.

Two change cases are directly addressed by the publisher and subscriber interfaces. First, another transmission protocol may be implemented via a dedicated pair of subscriber and publisher instances. Second, various external event sources may be connected concurrently, because network entities can be instantiated independently of each other. A drawback of the simple network interface design is the lack of a shared network connection between certain publishers and subscribers. In order to save resources by sharing a connection

and to easily include endpoints which require a shared, bidirectional connection, future versions of the interface program may extend the publisher and subscriber interfaces. A generic shared network connection may be introduced which is passed to exactly those entities which require access to it. Nevertheless, to ease the initial implementation, which focuses on the IEC 61499 ASN.1 protocol only, shared connection objects are not included in the current design.

The IEC 16499 ASN.1 implementation uses two separate class hierarchies for decoding and encoding packets, respectively. Two abstract base classes implement the ASN.1 encoding and decoding logic. Transport protocol specifics are outsourced to dedicated child classes which are instantiated by the network manager. In addition, a concurrent subscribe class was created which generically starts a subscription thread and manages its termination. The ASN.1 subscriber base class inherits from the concurrent subscriber to operate independently of the main program flow.

Data routing is implicitly implemented via filtering and buffering of event variables. In general, each publisher receives all scheduled events, regardless of the set of contained event variables. Therefore, the received event may contain all, none or some of the published variables. Since the ASN.1 protocol implementation always sends a fixed set of variables, each ASN.1 publisher maintains the current status of all published network ports and updates it according to the received event variables. In case the event does not contain any published variables, the output event is suppressed. The design decision of applying filtering within each publisher unifies and simplifies the event distribution mechanism and avoids the change of a scheduled event by filtering variables. Nevertheless, central hierarchical filtering and event distribution may be able to further increase the performance of the developed interface component.

4.4 Implementation and Quality Assurance

The previously described and designed interface component, which is called FMITerminalBlock, is implemented in C++11. The toolchain and several external libraries are described in detail in the corresponding Bachelor thesis [85]. FMITerminalBlock is released as an open-source project [35]. Some development tools, such as a public issue tracker and a publicly hosted usage documentation, are added to the toolchain.

To speed up the development process and to increase the quality of FMITerminalBlock, several external libraries are utilized. In particular, FMI++ Import Utilities and some Boost libraries [85] are included. Since the required C++ version was enhanced to C++11, several new language features and C++ standard library components are available. For instance, thread support is now provided by the standard library and does not have to be obtained by using the corresponding Boost library anymore.

Several measures which tackle quality assurance are implemented. First, an issue tracker is used to record known issues and planned refactoring operations. Each function is thoroughly documented in the source code, and a usage documentation describes the

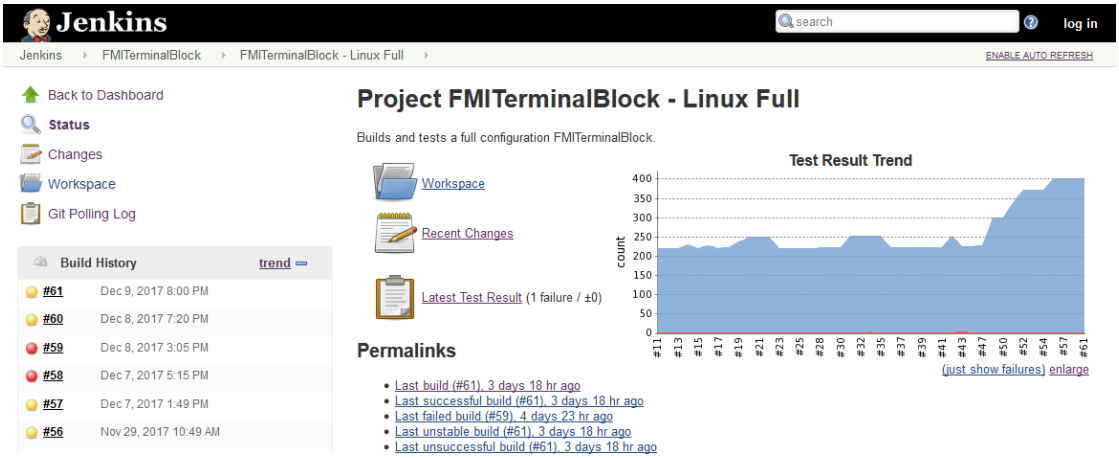


Figure 4.11: Exemplary Jenkins project overview

intended operation and features of FMITerminalBlock. Each software component is tested via a set of automated unit tests. Several mock-up and test classes are implemented to assess specific functionalities without external dependencies. For instance, a raw TCP test data source sends arbitrary packets to validate the ASN.1 decoding functionality.

All test cases are integrated into a Jenkins CI [52] which executes the test cases as soon as the source code of FMITerminalBlock or FMI++ changes. Additionally, the unit tests of FMI++ are executed on the target platforms to check ongoing support. Figure 4.11 shows an exemplary project summary of the Jenkins setup. The build and test steps are executed in various environments such as Windows 7 with Microsoft Visual Studio Compiler 14.0 and Debian Jessie with GCC 4.9.2. Further manual tests and the detailed evaluation within various use cases complement the implemented quality assurance strategy which is demanded by the development objectives.

CHAPTER 5

Evaluation

The envisioned and implemented methodologies of creating virtual components are evaluated in two test cases. Conducted experiments mainly target research questions regarding the feasibility, limits, and possible improvements of all implemented coupling approaches. It is not focused on describing a targeted average case experiment but on implementing experiments which introduce demanding requirements, e.g. with respect to timing parameters.

A basis for both experiments is an OLTC setup that includes a smart transformer and a controller which maintains the output voltage of the transformer. The OLTC itself is a device which switches the number of active windings in the transformer while the transformer is operating. Consequently, the output voltage of the transformer is changed and can, for instance, be adapted to volatile RES production [3]. Simple load and grid models border the virtual system under test and feed the artificial test pattern.

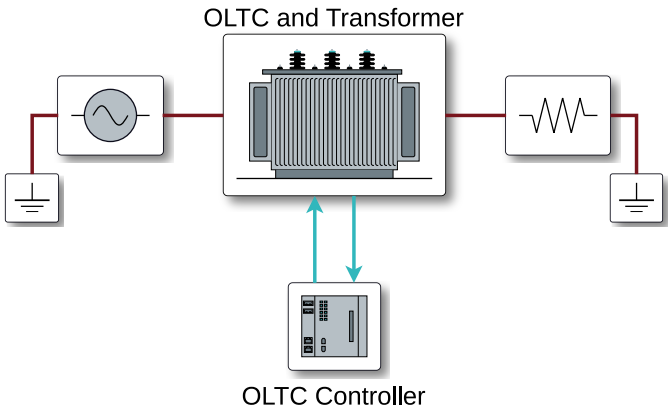


Figure 5.1: Overview of the smart transformer setup, including voltage source (left), load (right) and controller

Figure 5.1 illustrates the structure of the setup. The first test case, which is described in Section 5.3, compares the output of several controller implementations in an open-loop setup. One implementation is thereby physically available, others are introduced as virtual components. The test case is conducted in order to show and evaluate various delay effects of the coupling strategies. Additionally, it points towards a model-based testing and assessment workflow in which the outputs of developed hardware are directly compared to the reference model.

The second test case in Section 5.4 covers various closed-loop setups of a virtual transformer and an OLTC controller. The controller is once realized as dedicated hardware, once as a programm running on a software PLC, and once in a monolithic model. The transformer is always included as a virtual component. Test case 2 mainly focuses on demonstrating closed-loop aspects such as stability and the propagation of varying communication delays. Hence, it complements test case 1 which uses an open-loop setup, only.

Both test cases are conducted in the context of the ERIGrid horizon 2020 infrastructure research project [25]. The initial model as well as any hardware implementation are provided by the project partners from Ormazabal [73]. The thesis only coarsely describes given models and implementations and focuses on applied changes and optimizations.

5.1 General Timing Evaluation Methodology

The design of the interface component, which is called FMITerminalBlock, requires a careful evaluation of timing parameters during a simulation run. In case the emulation run is not able to execute in real time, results and measurements from one component may be incorporated late and may deteriorate gained results. Acceptable timing deviations strongly depend on the specific use case. For instance, the analyzed electrical transformer model shows transitive electrical effects which last well below 100 ms but mechanical and control delays are in the range of 500 ms to 1 s. In general, the investigation of transitive electrical effects requires a much higher timing precision than observing effects caused by mechanical delays [62, 63]. A general timing evaluation methodology and tools which do not focus on a particular use case should be provided in order to ease the implementation of other use cases. The implemented timing evaluation methodology focuses on the general quantification of achieved timing parameters rather than quantifying the effects on results of any timing deviation. Hence, a continuative discussion, which relates the timing effects revealed in the general timing evaluation to a particular test case, is still required to assess the overall quality of gained results. The presented timing evaluation methodology forms a foundation of such a comprehensive evaluation.

Section 4.3.2 briefly outlines the software component which records raw timing data during an emulation run. At various points in time, the current state of the computer clock and the destined simulation time of the currently processed event are recorded. Since the computer clock usually uses another standard epoch than the simulation, time values are converted between the two representations. For instance, a computer clock

Nr.	Name	Description
1	Real-time Generation	After the event submission by an external real-time entity
2	Prediction	After the event submission by an abstract event predictor
3	Begin of Distribution	After scheduling a certain event
4	End of Distribution	On finishing the distribution and model update
5	Outdating	On deleting a predicted event from the queue

Table 5.1: Sampled real-time instants

may count time from 1970-01-01 00:00 on, and the simulation may use the instant of time from where it is started to mark its standard epoch, $t = 0$. In addition to the absolute time of the computer clock, FMITerminalBlock converts each real-time stamp to the simulation time representation. The simulation time representation of real-time instants allows to quickly quantify the timing status of the program flow.

The timing of each event is recorded at several locations in the program flow. Table 5.1 summarizes each call to the time logging facility. The numeric annotations in Figures 4.1, 4.3, and 4.4 correspond to the sampling point numbers in Table 5.1 and illustrate the location of each call in the program flow. Some of the timing facilities are already present in the first implementation of the interface component [85]. Nevertheless, the timing data interface as well as the assessment methodology have been substantially extended by recording dropped events and by converting the standard epochs. The first extension allows to easily recover the state of the event queue at a particular point in time, and converting standard epochs avoids a time offset assumption on estimating achieved delays.

In a post-processing step, the timing of each event at the recorded points is re-assembled and combined into one data set. Thereby, the state of the event queue is simulated and only the set of active events is kept in memory. Since the event time stamp is used to identify an event, simultaneous events may not be properly processed. In case of concurrent (external) events, the affected events will either be dropped or a warning is issued. Future implementations may utilize an event ID mechanism which introduces an unique identifier for each event. Since the ID mechanism introduces several other issues, such as a possible reduction of concurrency, the current implementation is reduced to simple time stamping. In practice, no concurrent input event was observed because each test case only uses a single subscriber.

From the processed data set, the delay of each event at various stages in the program flow is calculated, i.e. the difference between the current real-time instant, which is expressed in terms of simulation time and the associated simulation time itself is expressed. For each stage, delay statistics such as average, minimum and maximum delay are calculated. Filtering is applied to limit the statistics to a certain set such as predicted events.

Histograms of each set of delay values are further used to give an impression of the shape of the delay distribution.

For real-time operation, the timely aligned distribution of predicted events is a significant aspect. According to the intended program flow, FMITerminalBlock begins the distribution of each event as soon as the simulation time is reached. Late events are scheduled immediately. For protocol implementations which block until the event has been sent, the time span in which an event is emitted can be estimated from the begin and end time stamp of the distribution phase. Hence, mainly the statistics of these two stages are used to assess timing quality.

Recorded timing data sets are also used to gain additional insights into possible sources of delay and to optimize included models. When calculating the time spans between one processing stage and another, the duration of each stage is estimated. Especially periods not affected by the wait operation of the scheduler may reveal further optimization potential. For instance, the duration from scheduling one event to predicting the following event highly depends on the performance of the included model and its solver. Similarly, the time of the distribution phase, especially when using the periodic synchronization approach, highly depends on the performance of a protocol implementation. A post-processing framework which is written in Python implements main analysis steps and is used to conduct the timing analysis in test case 1 and 2.

5.2 General Assessment Methodology

The assessment methodology first targets FMITerminalBlock in general and the implemented coupling approaches in particular. Additionally, the embedded controller hardware needs to be tested and validated in the system context of an OLTC transformer. Both test cases are designed such that FMITerminalBlock as well as the controller hardware can be evaluated. Therefore, recorded outputs are compared to a purely virtual baseline model.

The initial models of the system, i.e. the smart transformer setup, were provided as monolithic Simulink models. First, the purely virtual Simulink models were slightly adapted for real-time operation. Some blocks which caused unpredictable timing and inaccuracies due to resetting the simulation time within one integration step were substituted by equivalent subsystems. For instance, transport delay blocks, which caused inaccuracies were substituted by equivalent implementations and zero-crossing was enabled to accurately locate events. Sections 5.3.2 and 5.4.2 provide more details on the conducted model optimizations. A test set, which was applied to the model, was used to asses the quality of the model changes. The improved monolithic Simulink models were used as a baseline for evaluating the coupling methodologies.

In a subsequent step, relevant models were exported as FMUs via FMI Kit for Simulink [31] and accessed via FMITerminalBlock. As target version, FMI 2.0 was selected. Although FMITerminalBlock also supports FMI 1.0, the dedicated event modes of

FMI 2.0 ease debugging of connected FMUs and allow a more fine-grained control over event execution procedures. An Eclipse 4diac FORTE [37] software PLC is used to couple models and hardware as well as to implement a reference controller. Additional data logging functionality is implemented in the software PLC to gain timing and data records which are more independent from FMITerminalBlock.

It is expected that the quality of any results strongly depends on the configuration parameters of FMITerminalBlock. For instance, improper integrator settings may prevent real-time operation and therefore may drastically degrade the accuracy. Nevertheless, a comparison between the implemented coupling approaches should be performed. To limit the impact of improper configuration, most settings, such as the network protocol and most integrator configurations, are kept unchanged between the simulation runs. Only parameters that are specific to a certain coupling approach are optimized before conducting the actual simulation run.

For the periodic synchronization approach, the cycle time is minimized such that observed timing deviations are within reasonable bounds and do not decrease the accuracy. For the predictive approach, the look-ahead horizon is chosen according to the delays within the simulated models. Intermediate integrator steps are reduced such that no significant deviations from the expected model output can be observed. All optimization steps were conducted manually. It is expected that automatic parameter optimization would result in even better configurations, but the implementation of such an optimization procedure is beyond the scope of the thesis.

For each data source, recorded data is stored in a CSV file. Several Python scripts are used to process, evaluate and plot gained results. The external Python libraries Pandas [74], Numpy [70] and Matplotlib [64] are used to aid the evaluation.

For each experiment, every software component was executed on a Windows 7 PC which features an Intel Xeon dual core CPU W3505 at 2.53 GHz clock rate and 6 GB of main memory. In order to reduce the dependencies to unrelated programs and services, relevant processes, such as the interface program and software PLCs, were started with high process priority.

5.3 Test Case 1: Open-loop Controller Verification

5.3.1 Experimental Setup

The first test case uses an experimental setup which compares the outputs of two virtual component configurations and one hardware implementation simultaneously. Figure 5.2 illustrates test case 1. Each FMU instance as well as the hardware controller implements the same control algorithm. In particular, the same FMU, which was exported from the Simulink implementation of the OLTC controller, is instantiated twice to evaluate different coupling configurations. The PLC implementation of the OLTC controller is not used in test case 1, because it would not improve the visibility of delay effects. The

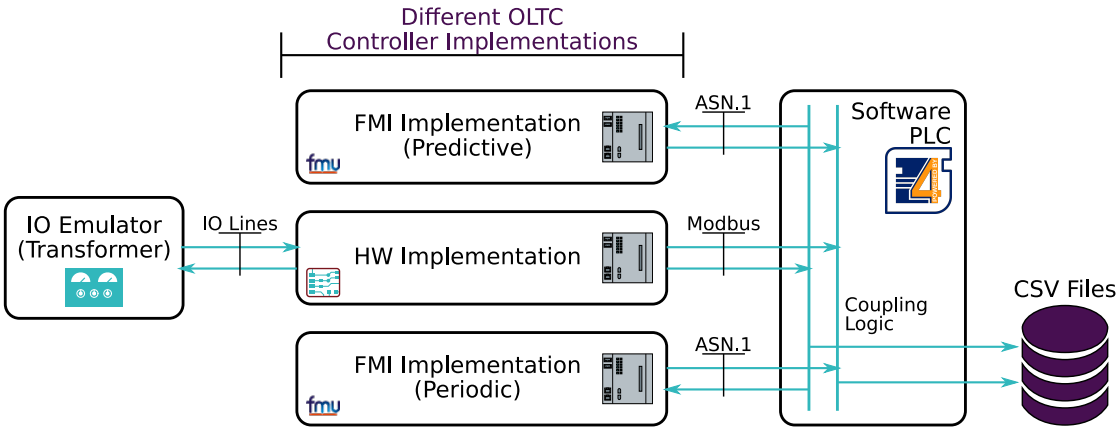


Figure 5.2: Setup of test case 1

outputs of all controller instances are recorded and used to evaluate differences in the implementations and coupling strategies.

In its destined operation, the hardware OLTC controller uses dedicated IO lines to directly control the OLTC actuators and to read the Low Voltage (LV) measurement signal. The current prototype implements a 0 V to 5 V signal to represent analog reading of the LV sensor and some digital signals which adhere to the same voltage levels to encode the ready signal and the control commands. For debugging and configuration, the hardware controller implements a Modbus TCP interface [65] over Ethernet. The interface exposes, for instance, the currently read and averaged LV values, the current control output, as well as a transformer status signal. The scaling of the analog voltage input can be adjusted via Modbus. An embedded development board featuring an ATmega2560 Microcontroller Unit (MCU) [7] from Microchip and a W5100 Ethernet – TCP/IP driver [104] from WIZnet is used to build the hardware controller.

A software PLC periodically polls the current and average voltage readings as well as the control and status output via Modbus. Since Modbus does not allow a slave device-initiated communication, polling is used for both, the periodic and the predictive configuration. A polling interval of 200 ms is chosen – as short as possible such that the MCU is not congested. As soon as a changed value is encountered, it is distributed. Changed low voltage and transformer status readings are instantly relayed to all connected FMUs. Changed control outputs from the hardware controller need to be post-processed. A single output variable, which stores the control action, is polled. In total, seven control actions from the OLTC controller are encoded; two blocking actions, a standard up, a fast up, a standard down, a fast down, and no action. The control action is periodically updated by the hardware controller as soon as new voltage readings are available. In particular, the control variable is updated in the same processing cycle regardless of being a fast or standard action. Nevertheless, to overcome transient inputs, the control algorithm requires to delay a control action depending on its particular type. For instance, a fast up action requires less delay than a standard up action until the OLTC is actually

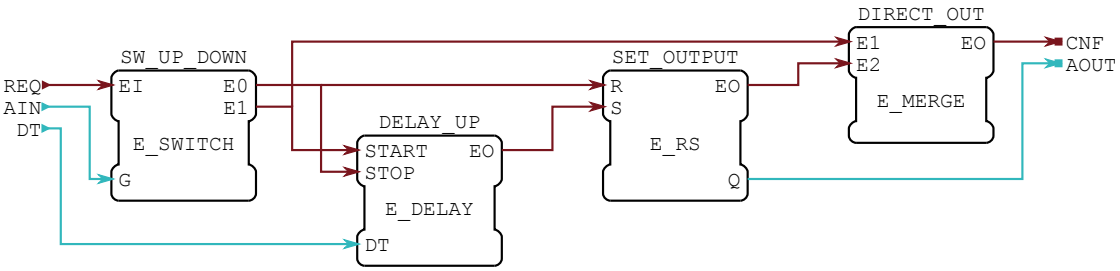


Figure 5.3: Control action delay composite network

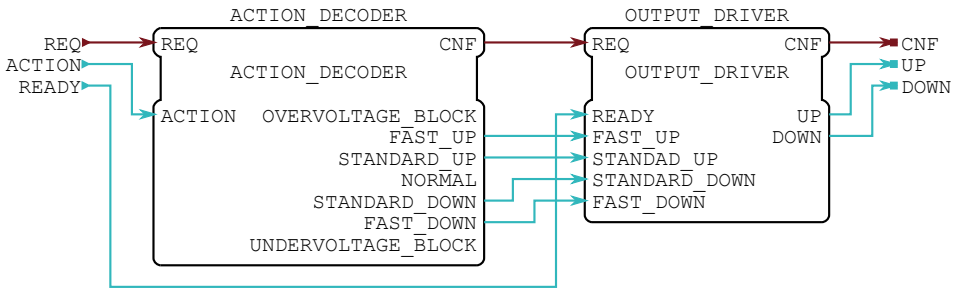


Figure 5.4: Delay compensation logic

commanded to switch one step up. The following Section 5.3.2 describes the control algorithm in more detail.

To emulate the missing delay operations of the hardware controller, an FB network is created which delays the output signal according to the control action. The FMU implementations of the controller internally delay the control actions themselves and are therefore not affected by the software PLC implementation. Figure 5.3 illustrates the composite network which delays a single boolean action control signal. The whole delay FB network is implemented into a dedicated CFB type. First, the event switch function block splits the execution path according to the action signal. The delay FB is started as soon as the corresponding action input is true and resets when the input becomes false again. An RS flip-flop block converts the execution path back to a single event semantic. The event merge FB is used to ease subsequent processing by generating an output event immediately, even if the output variable is delayed.

For each output action, an action delay FB is instantiated with the corresponding delay. A simple boolean disjunction connects all up and down switching signals, respectively. In case the ready signal is set to false, no output control will be asserted. The whole delay and merge logic is encapsulated into a dedicated output driver CFB type. Figure 5.4 shows the FB network which adapts the action signal from the hardware controller to the jointly used format.

The software PLC records every reading from the hardware controller as well as delayed control actions into a set of Comma Separated Values (CSV) files. Each recorded event is

amended by its time stamp. Therefore, a function block which queries the local clock of the PLC is implemented. Similarly, control events from both included FMITerminalBlock configurations are recorded. Each FMITerminalBlock instance sends output events via the IEC 61499 ASN.1-based protocol to a listening TCP server SIFB. In contrast to the Modbus connection, FMITerminalBlock directly exposes the up and down control signals from the controller model without the need of further processing them. For each event between the software PLC and an FMITerminalBlock instance, also a time stamp which is recorded by the PLC is available. Although the PLC instance and the FMITerminalBlock instances are executed on the same machine, the PLC time stamps are used to assess the timing performance of FMITerminalBlock and the included model. It has to be noted that deviations originating from the computer clock cannot be assessed by the deployed methodology, but communication delays and scheduling inaccuracies of FMITerminalBlock may still be assessed.

Since the inputs of all controller implementations are manually generated via a simple IO interface, the control inputs may significantly vary between one experiment run and another. The hardware controller implementation also supports to acquire its inputs via Modbus instead of the simple IO interfaces. Hence, a fixed input sequence could be applied to the controller via the network protocol and the software PLC which implements the coupling logic. To test the IO interfaces of the controller and the application of external inputs which do not origin from the system itself, it was decided to use the external inputs instead of the Modbus connection. Still, two FMITerminalBlock configurations should be evaluated and compared to each other. Hence, it was decided to run both FMITerminalBlock instances in parallel on a multi-core system. It is assumed that roughly the same input data, timing and processing load constraints now apply to both virtual components. It may still be feasible to replay recorded inputs via a scheduling mechanism at the software PLC but this scheduling mechanism may suffer from the same theoretical drawbacks as the scheduling mechanism of FMITerminalBlock. Nevertheless, it has to be noted that the operating system scheduler may still distort the timing of gained results in a parallel setup but it is believed that these deviations are less significant.

A single simulation run which covers multiple control output changes is evaluated in a post-processing step. According to Section 5.2, the deviations between each controller implementation are evaluated and the timing of each control action is quantified. Since one experiment covers multiple control actions, it is reasonable to statistically evaluate timing deviations of control actions. Due to the large time span covered by a simulation run, except for some outliers, repeating a simulation run will not drastically alter the outcome. In particular, only small deviations of the final results were observed during setup tests.

As shown in Figure 5.2, the setup of test case 1 covers three distinct configurations of the same controller which are all executed in real time. All of these configurations include components with unknown timing and execution properties. The FMI implementations use FMITerminalBlock which is to be evaluated and the exact behavior of the hardware implementation is not fully known either. A purely virtual reference simulation of the

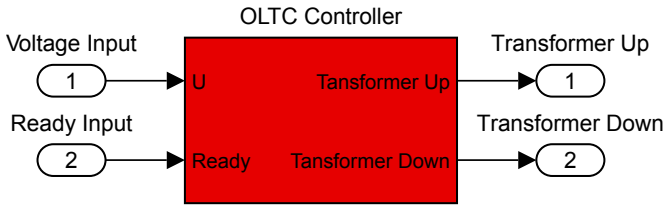


Figure 5.5: External interfaces of the OLTC controller model

Simulink controller model is additionally created to establish a common reference. The reference simulation is conducted offline and without any real-time synchronization in order to minimize synchronization effects and in order to keep the complexity as low as possible.

To create a useful reference, both the virtual simulation and the real-time setup of Figure 5.2 require synchronized inputs. Either a defined input pattern is applied to both, the hardware implementation via the IO emulator and the reference model, or inputs of the hardware implementation are recorded and passed on to the Simulink reference model. Since the manually operated IO emulator does not allow to playback a defined pattern, the second option was chosen and input voltage recordings from the hardware implementation are used to drive the Simulink model. Consequently, input voltage readings from the hardware implementation are biased by any deviations of the input measurement unit of the hardware controller, but the reference simulation can be directly used to evaluate the FMI implementations of the controller. An independent measurement device which records applied input voltages would be required to also assess the measurement unit of the hardware controller. Nevertheless, the detailed assessment of the hardware controller is beyond the scope of the thesis which focuses on virtual lab components, and consequently the simpler approach without external measurement equipment is chosen. Still, the hardware control algorithm can be compared to the reference simulation because (under the assumption that measurements are communicated correctly) it does not alter any results of the measurement unit.

Since the hardware controller does not time-stamp any measurement, the clock of the software PLC is also used to timely align the inputs of the reference simulation. Except for some timing evaluations which use the clock of the FMI interface component, all evaluations refer to the timing of the software PLC process and its clock. Consequently, results can be directly compared without taking any clock deviation into account.

5.3.2 Models and Configurations

In test case 1, a virtual controller, which is based on a Simulink model, is instantiated twice. The corresponding Simulink model, which is exported as an FMU, is outlined in Figure 5.5. The model is continuously solved, i.e. voltage inputs and the boolean ready signal are not sampled. The controller divides the input voltage into several consecutive voltage bands. In particular, one voltage band is spanned around the set point and does

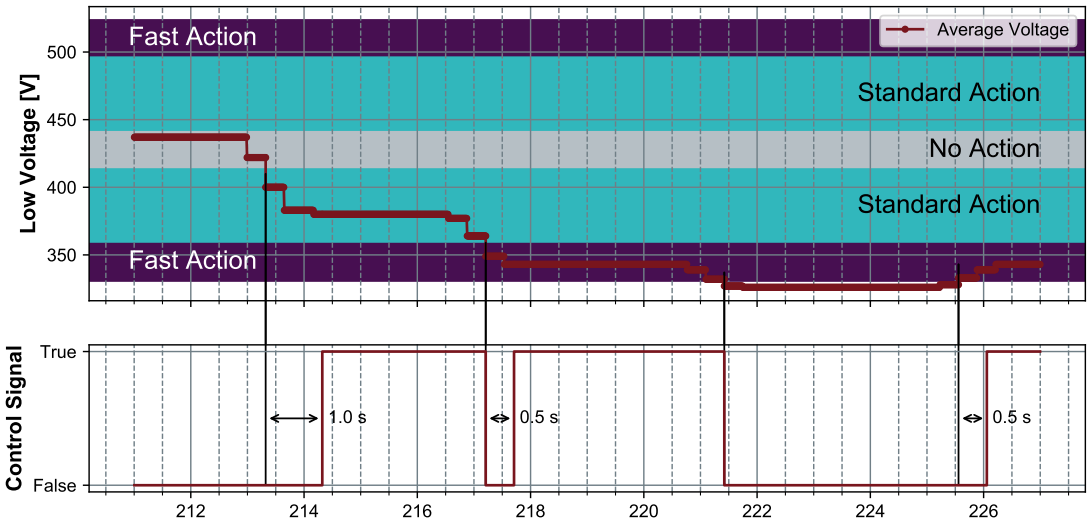


Figure 5.6: Exemplary controller operation

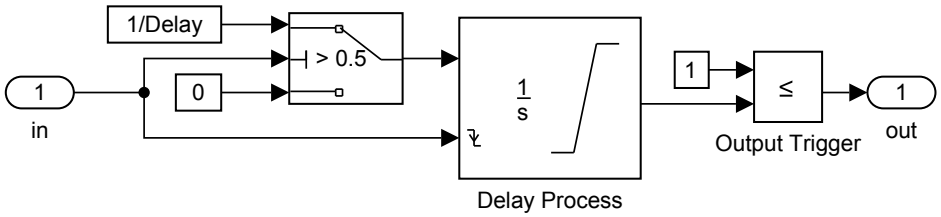


Figure 5.7: Optimized control action delay model

not trigger any control action. As soon as an adjacent voltage band is reached, a standard control action is initiated. Similarly, the outer voltage bands trigger a fast control action and, in case the input voltage deviates too much from the set point, the operation of the OLTC controller is entirely blocked for safety reasons.

Figure 5.6 illustrates a typical controller output as captured by the reference simulation. As soon as a standard control action is initiated, a delay of 1 s is awaited until the corresponding up or down signal is asserted and the OLTC is instructed to switch the tap by one step. A fast action request will be delayed by only 500 ms until it is presented at the output. Changes on the ready signal will instantly affect the model output and no delay operation is performed. In particular, if the ready signal is deasserted, all control outputs will immediately evaluate to false. In case the input voltage switches from a fast to a slow action voltage band or vice versa, the delay starts again.

Initially, the delay operation was implemented via transport delay blocks which, in an idealized view, delay the incoming time-continuous signal by a finite and constant amount of time [99]. In practice, a transport delay block stores various intermediate values and outputs them according to the currently set simulation time instant. For storing

intermediate values, an internal buffer, which was not further analyzed, is used. When exporting the model as an FMU, the buffer is not exposed in any state variable and can therefore not be properly reset to a previous state. Additionally, large intervals between calling the complete integrator step function decrease the accuracy of the transport delay block and limit the applicability of the predictive approach.

Although a transport delay block may process value-continuous inputs, the controller model exclusively uses the blocks for boolean signals which are encoded into floating point numbers. In general, a transport delay block may store multiple transitions of the input data at the same time. Nevertheless, the model is designed such that only one positive transition is delayed at once. Storing multiple transitions at once may even lead to unwanted glitches. To overcome the limitations of the existing action delay implementation, an implementation which uses an integrator state variable to store the current progress of a delay operation is introduced. Thereby, the complexity of storing and managing multiple intermediate nodes is reduced to a single state variable. The exposed state also simplifies reset operations and increases the accuracy of the prediction-based approach.

Figure 5.7 shows the revised one-sided action delay implementation. It provides the same functionality as the PLC implementation in Figure 5.3 but uses continuous Simulink signals instead. In case the input of the action delay block is asserted, the integrator linearly increases its state from 0.0 until the maximum value of 1.0 is reached. The state derivative therefore is the inverse of the targeted delay. The output is asserted as soon as the maximum value is reached. A dedicated comparison block is used to enable zero-crossing detection and to properly determine the timing of the switching event. It turned out that, without enabling zero-crossing detection, the generated Simulink FMU does not trigger an FMI event. As soon as a falling edge in the input is observed, the integrator resets to its initial value and a derivative of 0.0 is applied. Consequently, the output immediately evaluates to false and does not change until the next delay period is awaited.

The whole model as shown in Figure 5.5 is exported as a version 2.0 FMU. In addition to the main model in- and outputs, the FMU also exposes several state and debug variables. Nevertheless, to reduce timing overhead, none of these additional variables was accessed in the final experiment run. The entire model exposes exactly four derivative variables, one for each deployed control action delay block. The other Simulink blocks of the controller do not manage and expose a continuous state. In total, 22 event indicator variables control the the timing of model events. Although also the time event mechanism may be used to schedule the occurrence of discrete actions, the controller solely uses state events to indicate event occurrences. During debugging FMITerminalBlock in general and the FMU in particular, no time event was observed.

Table 5.2 summarizes the main simulation settings of both FMITerminalBlock instances. In the predictive configuration, direct output dependency support of FMITerminalBlock was enabled, i.e. every input event of the model directly triggers an output event which communicates the state of the model after the input event has been applied. Although

Property	Predictive Config.	Periodic Config.
Direct Output Dependency	Yes	-
Variable Synchronization Step Size	Yes (Always)	No
Event Search Precision	1 ms	1 ms
Integration Method	Euler	Euler
Integration Step Size	Default (50 ms)	Default (3 ms)
Look-ahead/Synchronization Time	1 s	30 ms
Look-ahead Step Size	500 ms	-

Table 5.2: FMITerminalBlock simulation settings in Test Case 1

this option decreases the real-time accuracy of a particular event, it is necessary in order to immediately send control output changes due to the new input conditions. In particular, when exiting a certain voltage region, the control output is immediately deasserted on processing the input events and calling the event handling functions. Due to the immediate nature of the control action, no event which follows the input event is triggered. In case direct output dependency support would be disabled, the immediate control action output would be delayed until the next model event or by the look-ahead horizon time.

In contrast, the periodic configuration does not support direct output dependencies and always delays communication to the next synchronization event. Any direct output dependency support on the periodic approach would require to delay the final event execution and output calculation beyond the next synchronization point. Such a retardation, which is similar to the predictive approach, may be technically feasible but degrades the real-time performance of synchronization events and violates the described principle of the periodic approach. Again, the implementation of another execution variant and the detailed study of delay effects thereof is beyond the scope of the thesis.

The predictive approach inherently supports variable step sizes. In contrast, the periodic operation may be configured to schedule the next synchronization event as soon as a model event is encountered. To simplify the evaluation and to cleanly relate observed effects to the simulation approach, the variable step size feature is not enabled in the periodic configuration. For both configurations, an event search precision of 1 ms is used. The precision controls the accuracy of any state event detection. As soon as the time range, in which a zero-crossing of an event indicator is observed, drops below the event search precision, the event location is approximately fixed to one point inside this range.

To study the effects of the synchronization methodology, both configurations use the same integrator. Since all continuous states of the model directly relate to the state of the integrator block in Figure 5.7, it can be seen that the state derivative will most likely be constant between two FMI events. Consequently, a very simple numerical integration algorithm is sufficient to solve the model accurately. Euler's algorithm was chosen as the most simple and deterministic one. Due to the constant step size, the same number of

integrator step is used to solve a single look-ahead or synchronization step. Due to the different timing horizons in both configurations, no unified integrator step size is feasible. For both configurations, the sensible default value of 50 ms and 3 ms, respectively was chosen, which corresponds to ten integrator steps per look-ahead or synchronization step. Although the state equation may be exactly solved in just one integrator step, it was decided to use the default parameter to account for possibly nonlinear implementations. Additionally, during the manual optimization step, only little impact was observed on changing the integration step size by one order of magnitude.

Two of the main influencing factors of the timing accuracy are look-ahead horizon time and synchronization step size, respectively. Both were selected such that average control action delays according to the software PLC clock are as low as possible. Additionally, the timing evaluation output of FMITerminalBlock was checked whether observed event delays eventually dissipate and an average real-time execution is feasible. Especially for the periodic configuration, the size of the synchronization step crucially influences the timing accuracy. In case the step size is chosen too coarse, control actions are delayed unnecessarily. In case a too fine-grained synchronization is chosen, the model cannot be solved in real time and processing delay is introduced. It turned out that a synchronization period in the order of 30 ms provides best results. In the predictive configuration, the length of the look-ahead horizon in conjunction with the look-ahead step size have the greatest impact on the accuracy. Due to the simplicity of the state structure, only few nodes are sufficient to properly reset and interpolate the state. A look-ahead horizon of 1 s and one intermediate node turned out to produce satisfying results. Larger values result in an extended prediction time while smaller values create numerous unnecessary events.

5.3.3 Results

Before assessing the accuracy of all configurations, the real-time performance is evaluated as described in Section 5.1. Especially the delay, which is the difference in the measured real-time instant to the intended simulation time, is evaluated. Table 5.3 summarizes the main statistics of observed delay values of both FMITerminalBlock instances. The stage column lists the point in the processing flow at which the time is sampled. The registration stage subsumes both possible event sources, model events and externally added ones. Since model events may not be scheduled in case external events outdate the results, the number of sampled registrations is higher than the number of actually scheduled events.

The periodic configuration generates much more events, due to a small synchronization period of 30 ms compared to the look-ahead horizon of 1 s for the predictive one. In particular, the average distance between two events in terms of simulation time is 29 ms for the periodic configuration and 303 ms for the predictive one. Since the execution time of both experiments is (roughly) the same, the difference in the event rate by one order of magnitude is also reflected in the total number of recorded samples. Note that the high number of scheduled samples for the periodic configuration does not

Config.	Stage	Clean	Samples	Delay Statistics			
				Mean [s]	Variance [s^2]	Min. [s]	Max. [s]
Predictive	Registration	no	1061	-0.425	0.2184	-1	0.078
	Registration	yes	1009	-0.422	0.2162	-0.974	0.0322
	Begin Dist.	no	791	0.0162	2.566e-4	0	0.078
	Begin Dist.	yes	753	0.0155	2.161e-4	0	0.0462
	End Dis.	no	791	0.0169	2.712e-4	0	0.125
	End Dist.	yes	753	0.0162	2.184e-4	0	0.0468
Periodic	Registration	no	11285	-0.00802	2.218e-4	-0.03	0.149
	Registration	yes	10721	-0.0087	1.543e-4	-0.0289	0.0258
	Begin Dist.	no	11015	0.0194	1.753e-4	0	0.179
	Begin Dist.	yes	10465	0.0191	1.412e-4	3.65e-4	0.0439
	End Dist.	no	11015	0.0201	1.773e-4	7e-6	0.179
	End Dist.	yes	10465	0.0197	1.398e-4	9.49e-4	0.0442

Table 5.3: FMITerminalBlock real-time performance parameters (open-loop experiment)

necessarily correspond to a high number of actually distributed events. In the IEC 61499 ASN.1 implementation, no event message is sent in case the scheduled event does not contain any changed data. Hence, the periodic implementation allows a relatively small synchronization period without frequently overloading connected components.

For each processing stage in Table 5.3, one cleaned row is presented in which outliers are removed prior to calculating the statistics. A simple outlier detection is used which symmetrically drops 5 % of the samples sorted by their delay. For both configurations, the outlier removal reveals that only few events show very high delays of up to 179 ms. While for the predictive configuration a maximum delay value at the end of the distribution phase of 125 ms was observed, 97.5 % of the delays stayed below 47 ms. Similar drops in the maximum delay values are also observed for the periodic configuration.

Figures 5.8 and 5.9 further illustrate the delay distribution of the predictive and periodic configuration, respectively. Both figures show the delay histogram for the processing stages in Table 5.3 without removing outliers a-priori. From these figures it is also visible that the majority of observed delays stayed well below 50 ms. For the controlled process of an OLTC transformer, which features mechanical switching times of about 500 ms and intended control delays in the same order of magnitude, achieved delays are, for both configurations, denoted as acceptable.

From Table 5.3, it is obvious that all distribution phase statistics reside within the same order of magnitude. Only minimum and maximum registration delays strongly differ between the configurations. In principle, the registration delay is the delay when a new event is added to the queue. Since a predicted event is ideally added before it should

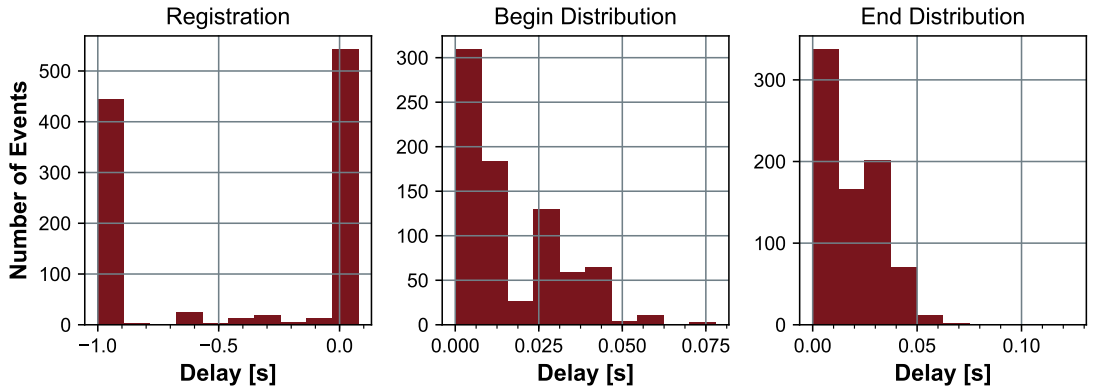


Figure 5.8: Histogram of observed execution delays (predictive open-loop configuration)

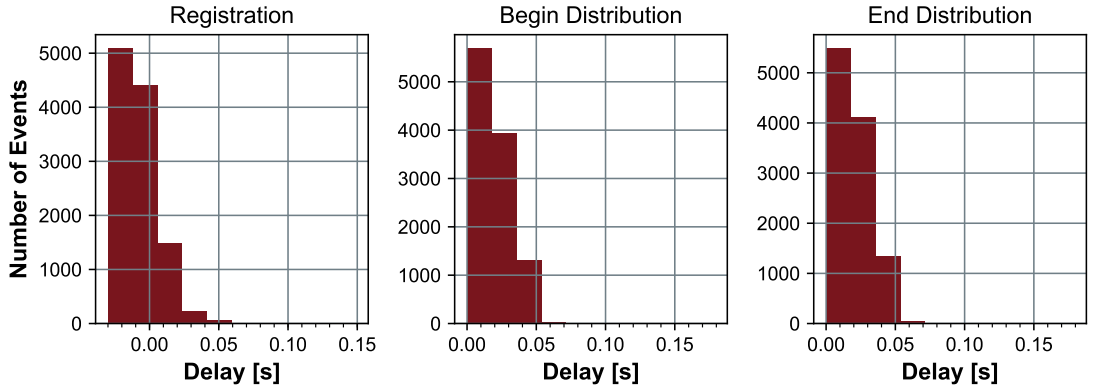


Figure 5.9: Histogram of observed execution delays (periodic open-loop configuration)

be scheduled, negative delay values do not degrade the real-time performance. For both configurations, minimal delay values which equal the synchronization period and the lookahead horizon, respectively, are observed. For the minimum delay events, generalized prediction could be conducted faster than the platform-dependent granularity of the timing recordings. In the current implementation, registration delays of external events are always positive because the simulation time is set to the real-time instant on receiving an external event. Consequently, the maximum registration delay of all events is equal to or bigger than zero as soon as a single external event is received. Nevertheless, the maximum registration delay for both configurations is caused by a predicted event which was submitted late.

Figure 5.10 shows recorded values during the entire experiment. The top row illustrates the low voltage reading which was accessed from the controller hardware and sent to all other components of the setup. Several voltage band crossings are included in the entire voltage curve. Each crossing triggers a control action as described in Section 5.3.2. In

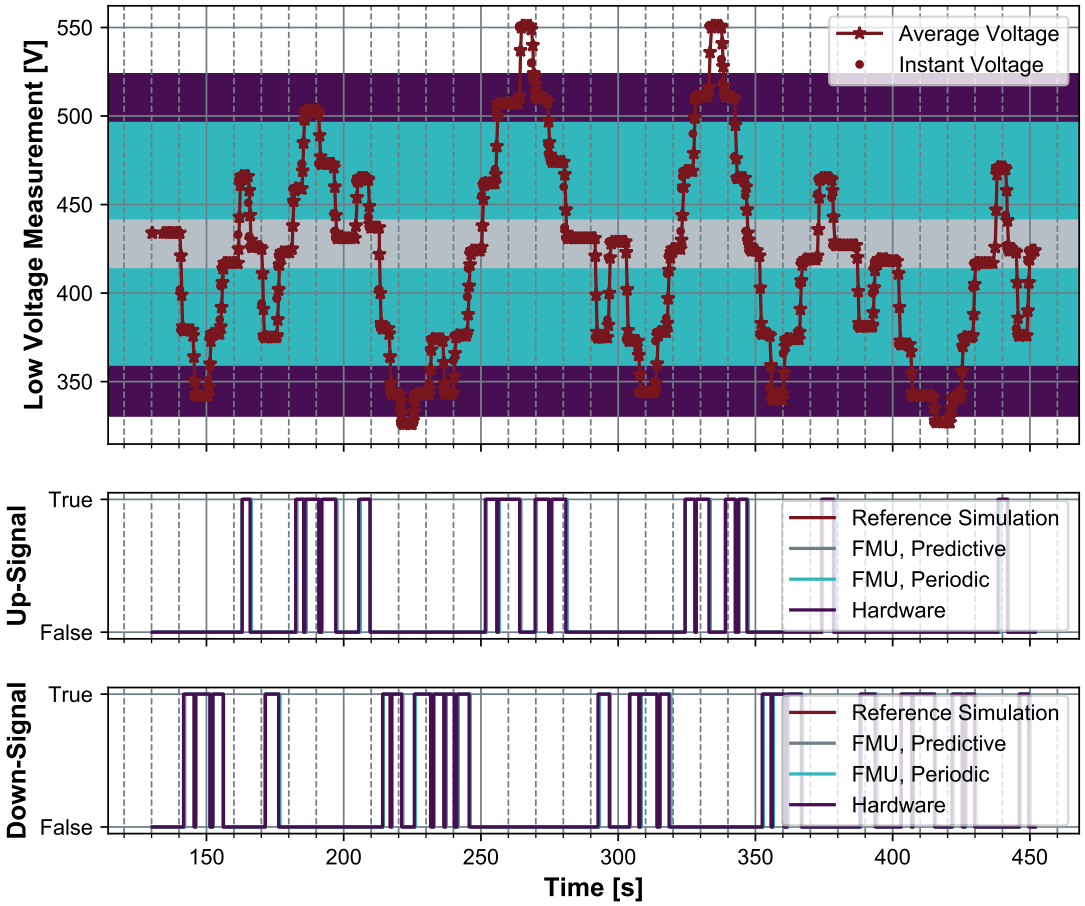


Figure 5.10: Overview of the open-loop experiment

principle, both configurations and the hardware output the same control action signals. In particular, no control action is omitted and no spurious control action with respect to the offline reference simulation was observed. Nevertheless, the recorded timing slightly varies between each controller realization. Figure 5.11 shows three control signal transitions in more detail and visualizes typical timing deviations.

In a post-processing step, the time of each signal transition is identified and the delay between the signal and the reference simulation is calculated. Table 5.4 summarizes the delay statistics of every recorded signal. Additionally, a cumulative statistic which includes both signals of each configuration is given. Most obvious, all delays, except for the hardware implementation, are positive, i.e. the output of the virtual components in the predictive and periodic configurations are always changed later than the reference outputs. In contrast, minimal hardware delays of up to $-516\,\mathrm{ms}$ are observed. Consequently, the hardware output is delivered before the reference output changes, but also the reversed effect of late hardware outputs is observed.

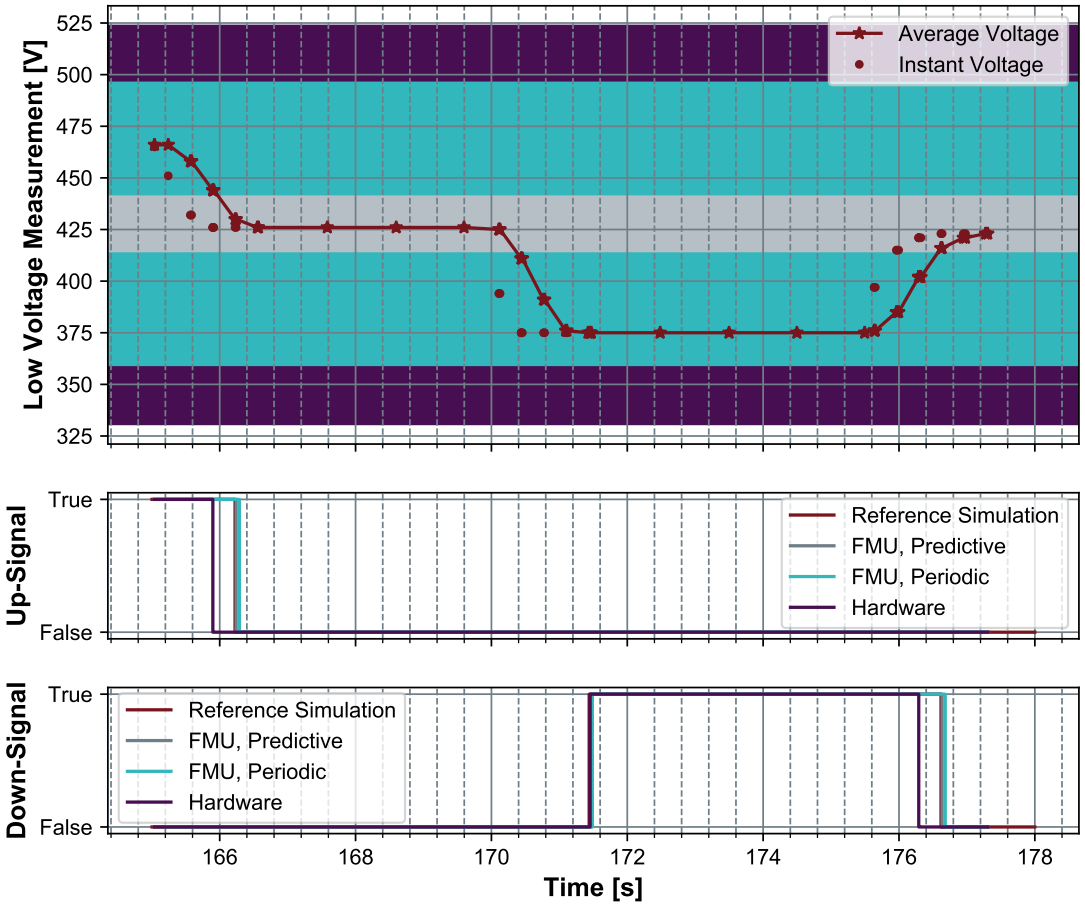


Figure 5.11: Exemplary output timing in the open-loop experiment

The software PLC queries each value from the hardware independently and does not execute an atomic read operation. After each variable is queried, the next polling cycle is awaited. The hardware controller updates the Modbus register values periodically and processes pending read requests after the update has been performed. Due to the non-atomic nature of request operations, an inconsistent view of the controller state is transiently processed. In particular, it is observed that the voltage readings are queried before the controller updates its state and the control outputs are queried after an update. Hence, one event which contains the old voltage but the updated control action values from the hardware is processed by the software PLC. At the next successful polling cycle, the updated voltage value is processed, too.

Due to the limited resources of the controller hardware, in rare cases successful polling cycles of up to 515 ms can be observed during the experiment run. Each extreme minimum delay of up to −516 ms can be tracked down to a combination of late voltage updates and delayed polling cycles. The positive maximum values of up to 169 ms can be tracked

Config.	Signal	Samples	Delay Statistics Median [s]	Mean [s]	Variance [s^2]	Min. [s]	Max. [s]
Predictive	Down	46	0.03	0.0246	0.0001616	0	0.084
	Up	30	0.016	0.0428	0.007932	0	0.498
Periodic	Down	46	0.062	0.0567	0.0002295	0.03	0.094
	Up	30	0.062	0.0781	0.006728	0.031	0.498
Hardware	Down	46	-0.327	-0.264	0.02085	-0.516	0.015
	Up	30	-0.327	-0.236	0.02868	-0.515	0.169
Predictive	Both	76	0.023	0.0318	0.003307	0	0.498
Predictive (Cl.)	Both	75	0.016	0.0256	0.0004148	0	0.169
Periodic	Both	76	0.062	0.0651	0.002904	0.03	0.498
Periodic (Cl.)	Both	75	0.062	0.0594	0.0004109	0.03	0.169
Hardware	Both	76	-0.327	-0.253	0.02413	-0.516	0.169

Table 5.4: Signal delay statistics (open-loop experiment)

down to scheduling latencies within the software PLC. The voltage and control output readings are queried synchronously from the hardware, and, consequently, the reference model is updated as expected. Nevertheless, the delayed response of the software PLC is timed well beyond the intended timing and a positive delay was observed. Since both phenomena which cause the extreme delay are not caused by FMITerminalBlock but by the connected HuT, no further investigation is conducted.

Since none of the FMITerminalBlock instances receives a voltage reading before it is processed by the software PLC, all delays are in the positive domain. In rare cases, delays of up to 500 ms are recorded by the PLC. The switching event can be tracked down to a particular set of FMI events which show only a delay of up to 40 ms in the FMITerminalBlock timing recordings. The output event timing in terms of simulation time corresponds exactly to the expectations from the Simulink model. Hence, the delay may either be caused by an erroneous real-time tracking of FMITerminalBlock or by network and scheduling delays within the software PLC. Since the time tracking facilities of FMITerminalBlock were carefully assessed within the development phase, both configurations simultaneously showing the peak value and similar scheduling effects being observed at the hardware configuration, it is strongly believed that the outliers were generated outside the domain of FMITerminalBlock. On removing the single outlier, which accounts for 1.3 % of the samples, from the statistics, at most 169 ms of delay is observed. When considering that the reference Simulink model mostly uses a step size of 10 ms to store simulation results, maximum delay values of 169 ms directly correspond to the expectations from the timing analysis.

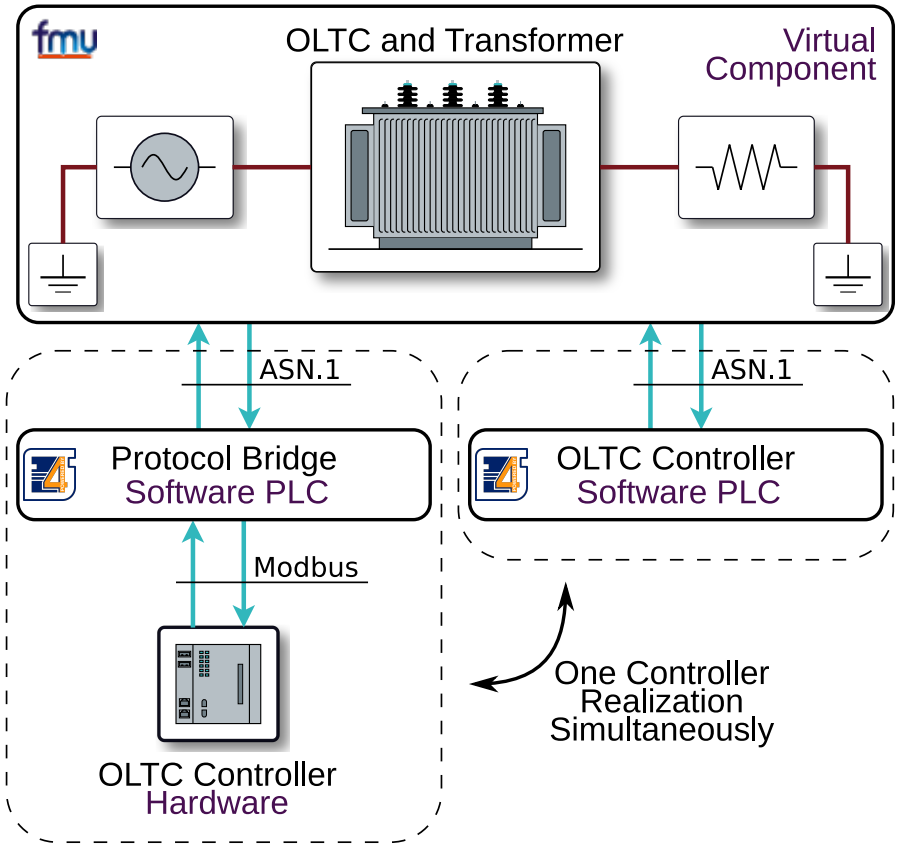


Figure 5.12: Setup of test case 2

5.4 Test Case 2: Closed-loop CHIL

5.4.1 Experimental Setup

In contrast to the open-loop setup of test case 1, which primarily targets the quantitative evaluation of delay sources, the test case 2 experiments aim at qualitatively describing the effects of the coupling approaches on a closed-loop setup. The test case consists of multiple experiments which implement a single configuration at once. Figure 5.12 illustrates the basic scheme of each experiment. A power hardware model is exported as an FMU and used to create a virtual component via FMITerminalBlock. The OLTC controller is realized in two different ways. First, the OLTC controller hardware from test case 1 is accessed via a protocol bridge which translates the ASN.1-based protocol and Modbus. Alternatively, a PLC OLTC controller implementation is used to suppress delays which are introduced by polling a Modbus device. FMITerminalBlock connects to the software PLC and sends two process variables, the LV measurement and a status flag indicating whether the OLTC is ready to accept new control commands. Simultaneously, it receives the status of two control signals, one for increasing and one for decreasing the

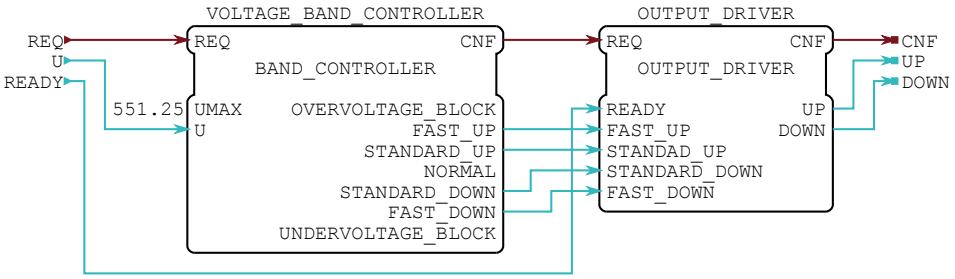


Figure 5.13: OLTC controller PLC implementation

output voltage.

The protocol bridge for accessing the embedded OLTC controller hardware converts between the periodic communication scheme of Modbus and the event-based communication scheme of the IEC 61499 protocol. The control outputs of the OLTC controller are polled periodically with a nominal polling cycle of 200 ms. Since the same hardware controller implementation is used for test case 1 and 2, also the same polling interval is used. In contrast to the first test case, the controller hardware is now configured via Modbus registers such that the sampled input voltage is read from another Modbus register instead of sampling analog IO lines. As soon as the virtual component reports a changed voltage value, the new reading is relayed via Modbus without awaiting the next polling cycle. Hence, in one direction, the software PLC has to poll the control outputs and in the other direction, new voltage readings can be instantly applied.

After each polling cycle, the software PLC internally generates an event regardless of any actual value changes. To fully exploit the capabilities of an event-based execution model within the protocol bridge, events which are not associated with changed control outputs are filtered before processing the data. Still, the semantic and the timing of a control output has to be adapted to the requirements of the OLTC transformer model. The same delay compensation logic from the first test case in Figure 5.4 is also used in the protocol bridge. After adapting the representation of control commands, output events are directly relayed to a connected FMITerminalBlock instance.

To compare the effects of multiple virtual component configurations more easily, an event-based software PLC implementation of the OLTC controller was created. The software PLC controller thereby uses the same functional principles and timeouts as the hardware implementation which is already described in Section 5.3.2. In contrast to the hardware implementation, no polling is involved and changed control outputs can be directly sent to the virtual OLTC transformer instance. Figure 5.13 shows the main components of the PLC implementation. A voltage band controller BFB implementation converts the voltage reading to a set of variables which indicate the currently active voltage band. A maximum voltage parameter is used to internally scale the voltage bands according to the actual transformer implementation. The output driver FB delays and converts the band signals into the final control output signals. The implementation of

the output driver is taken directly from the protocol bridge and from test case 1 which increases comparability by using the same timeout mechanisms.

The test case focuses on two main aspects. The first one is to qualitatively assess the closed-loop performance of involved hardware and software components and the second one is to compare the performance of the periodic and the predictive approach. Two experiments, which both use the predictive configuration, specifically tackle the first focus of closed-loop assessment. The first one directly deploys the controller at the software PLC and the second one uses the dedicated controller hardware. When comparing both experiments with the reference simulation, differences in the controller implementations and communication strategies in a closed-loop operation become visible. A third experiment is conducted which uses periodic synchronization in order to compare different synchronization strategies of virtual components. To limit the impact of connected hardware on the simulation results, only a software PLC implementation of the OLTC controller and no polled hardware is used. Again, the results of the virtual component setups are compared to the ground truth reference simulation.

5.4.2 Models and Configurations

Similar to the controller model, the OLTC and transformer models were provided by Ormazabal [73] as a monolithic Simulink model. The model uses components from the Simscape Power Systems library [83] which are configured to use phasors to solve the circuit. One three-phase programmable voltage source block is used to represent the medium voltage grid. In order to simulate voltage changes and to trigger switching actions, multiple voltage steps are applied via the voltage source. A static and fixed load is used at the low voltage side of the transformer. The entire model exports the absolute Medium Voltage (MV) levels for debugging, the LV measurement, the current tap position as well as a ready signal. Two input control signals, one for switching the current tap towards a higher LV and one for decreasing the LV, are required to drive the virtual OLTC transformer model. Note that the MV profile is statically entered in the model and cannot be changed via model inputs. Consequently, no MV profile has to be generated to drive a virtual component, simplifying its usage.

Beside various detailed electrical characteristics, the model also covers several mechanical properties of the OLTC. In particular, the time required by the OLTC to switch taps is covered. As soon as a positive edge on one of the control signals is encountered, a switching action is started. The OLTC indicates that it is currently switching tap by deasserting the ready signal. As soon as the switching action is finished, ready is asserted again by the OLTC hardware and the tap position output changes. The partly asynchronous nature of the control and ready signals is exploited by the OLTC controller implementations to switch multiple taps in a row unless the low voltage value is in the desired voltage band. The controller model will instantly deassert any action output in case the ready input evaluates to false. As soon as the ready input is asserted again, the control output can be applied without awaiting any artificial delay. Consequently, the transformer model can initiate the next switching operation after the previous one has

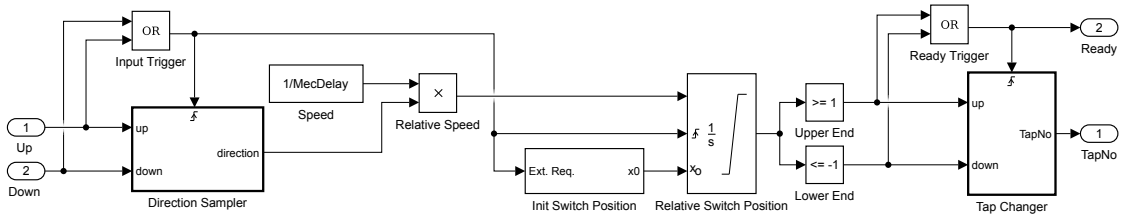


Figure 5.14: Adapted Simulink tap control implementation

been completed. Via the feedback loop formed by the ready signal, a controller does not have to consider any (worst case) switching delay of an OLTC but adapts to the delay of the hardware. Nevertheless, the implementation is not fully delay-insensitive because the rising edge of the ready signal does not depend on any falling edge of a control signal. Such a behavior is considered as a part of the system properties and is therefore preserved.

Similar to the controller model described in Section 5.3.2, mechanical delays in the OLTC were initially modeled by transport delay blocks. To increase the accuracy of a prediction-based approach, the tap control implementation was changed to an implementation which uses an integrator block to represent the current status of the mechanics. Figure 5.14 shows the adapted implementation of the tap control model. The relative switch position integrator thereby reflects the mechanical position and consequently the delay until the next winding is switched. It does not encode an absolute switch position. The direction sampler simply outputs the domain of one of the input signals as soon as a rising edge is detected. Simultaneously, the integrator is reset to zero. As soon as the integrator reaches the border position of plus or minus one, it is stopped and the final position is indicated via the upper and lower end comparisons. To initially assert the ready signal and to avoid an initial switching operation, a dedicated block initially resets the integrator to a border position. As soon as one control signal is asserted for a control action request, the integrator is reset to zero instead. The tap changer block maintains the actual absolute tap position and updates its state as soon as a switching operation is completed and the ready signal rises again.

To create a reference simulation, the closed-loop setup, which includes the controller as well as the transformer models, was first simulated as a monolithic Simulink model. Due to the complexity and condition of the model, only a variable step size solver was able to satisfyingly solve the embedded ODEs. Solvers which feature a fixed step size either showed divergent results or suffered from an unacceptable execution time. As the controller model, the OLTC transformer model is exported as a version 2.0 FMU. In addition to the previously described model in- and outputs, the FMU also exposes several internal signals and states. Nevertheless, none of these are used in the virtual component configuration which only accesses the two control inputs as well as the tap position, ready, and voltage outputs. The entire continuous dynamic state of the model is encapsulated into nine state variables, and 13 state event indicators are used to indicate

Property	Predictive Config.	Periodic Config.
Direct Output Dependency	Yes	-
Variable Synchronization Step Size	Yes (Always)	No
Event Search Precision	1 ms	1 ms
Integration Method	Adams-Bashforth-Moulton	Adams-Bashforth-Moulton
Integrator Library	Sundials	Sundials
Initial Integration Step Size	Default (10 ms)	Default (10 ms)
Absolute Integrator Tolerance	10^{-4}	10^{-4}
Relative Integrator Tolerance	10^{-4}	10^{-4}
Look-ahead/Synchronization Time	1 s	100 ms
Look-ahead Step Size	100 ms	-

Table 5.5: FMITerminalBlock simulation settings in Test Case 2

model events such as tap switching operations. On creation, the FMU is configured such that S functions are loaded from binary MEX files. This is necessary to successfully export and run the model.

Two distinct configurations are used to create virtual components. Table 5.5 summarizes the main configuration settings for both of them. The same interface settings, such as the order and amount of communicated model variables, are used in all configurations and are therefore not listed. Direct output dependency support is enabled at the predictive configuration because the assertion of one control signal may immediately trigger a falling edge on the ready signal. Without direct dependency support, the transition would be delayed until the next event, leading to possibly missed pulses on the ready signal. As for test case 1, variable step size support is disabled for the periodic configuration and the event search precision is set to 1 ms for both configurations.

Following the observations of the reference simulation, an adaptive integration algorithm was chosen to solve the model. In the FMI++ documentation [96], the Adams-Bashforth-Moulton integrator is recommended to solve models which require a high effort for computing the model equations. While the generated FMU only exports a few model variables, considerable computation effort was observed for evaluating the derivative functions. Consequently, an Adams-Bashforth-Moulton integrator was chosen. It turned out that the Sundials implementation [49] of the integration algorithm performs significantly better than the Boost Odeint implementation [1] which both can be accessed via FMI++. Preliminary tests showed that only the Sundials implementation is able to solve the model without accumulating delay values. Consequently, the faster implementation is selected for all configurations. To increase the comparability of gained results, both configurations select the default initial step size of 10 ms and the same tolerance values.

The look-ahead horizon, synchronization time and look-ahead step size are chosen such that the models can be solved in nearly real time and that delay values accumulated

Experiment	Stage	Clean	Samples	Delay Statistics			
				Mean [s]	Variance [s^2]	Min. [s]	Max. [s]
PLC Predictive	Reg.	no	379	-0.218	0.1291	-0.896	0.187
	Reg.	yes	361	-0.211	0.1206	-0.837	0.172
	Begin Dist.	no	282	0.0903	0.003331	0	0.187
	Begin Dist.	yes	268	0.0904	0.003101	0.00596	0.172
	End Dist.	no	282	0.0909	0.003335	0	0.187
	End Dist.	yes	268	0.091	0.00311	0.00617	0.172
PLC Periodic	Reg.	no	842	-0.0376	0.0001798	-0.0688	0
	Reg.	yes	800	-0.0382	0.0001463	-0.0503	0
	Begin Dist.	no	771	0.0262	2.176e-05	0.0156	0.0336
	Begin Dist.	yes	733	0.0262	1.933e-05	0.0183	0.0332
	End Dist.	no	771	0.0275	4.919e-05	0.0181	0.0468
	End Dist.	yes	733	0.0273	3.98e-05	0.0185	0.0468
HW Predictive	Reg.	no	293	-0.271	0.12	-0.837	0.187
	Reg.	yes	279	-0.268	0.1136	-0.83	0.0156
	Begin Dist.	no	222	0.0219	0.001354	0	0.187
	Begin Dist.	yes	212	0.0185	0.0007501	0	0.172
	End Dist.	no	222	0.0224	0.001381	0	0.187
	End Dist.	yes	212	0.0191	0.0007828	0	0.172

Table 5.6: FMITerminalBlock real-time performance parameters (closed-loop experiments)

during a short period of time quickly decay to their minimum value. It turned out that, in most cases in the periodic configuration, due to the limited precision, simulated LV readings vary slightly from one steady-state period to another. The resulting fully distributed output event at nearly every synchronization point limits the performance and further leads towards the synchronization period of 100 ms.

5.4.3 Results

Again, the real-time performance parameters are evaluated prior to the simulation results. Table 5.6 summarizes the main timing statistics of every evaluated experiment run. The first two blocks cover the software PLC implementation with a predictively and periodically synchronized virtual component, respectively. The last block corresponds to the experiment which interfaces the OLTC controller hardware. Observed absolute event delays first indicate that the model requires much more computational effort than the OLTC controller model of test case 1. Especially for the predictive configurations, delay values of up to 187 ms, even at registration, are observed. Unlike test case 1, the maximum values are not dominated by few outliers, but several predictive samples with

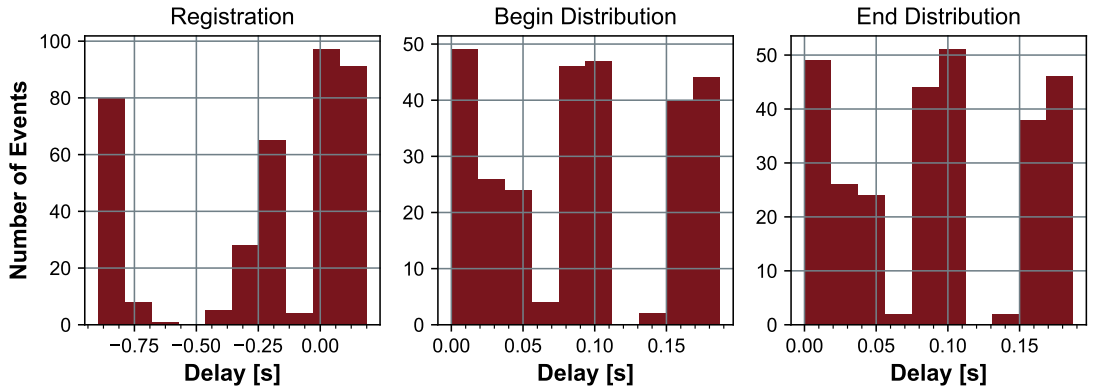


Figure 5.15: Histogram of observed execution delays (predictive, PLC controller)

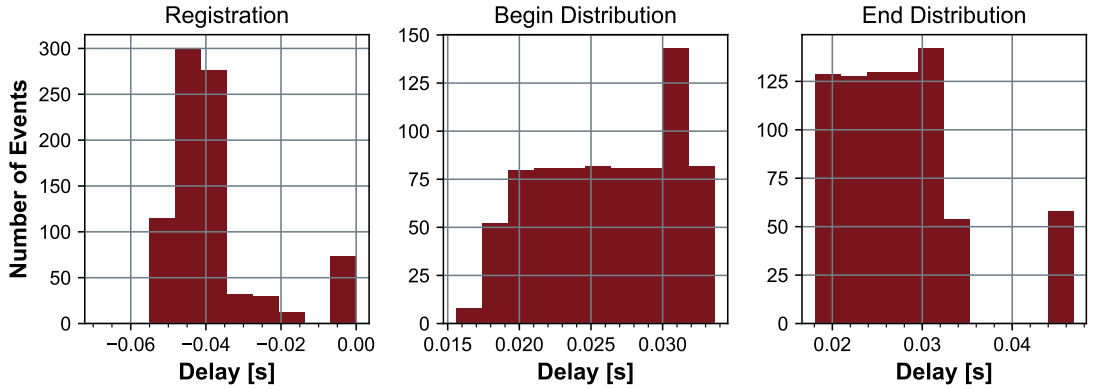


Figure 5.16: Histogram of observed execution delays (periodic, PLC controller)

distribution delay values in the order of 180 ms are observed.

Figures 5.15 to 5.17 show the delay distributions for each experiment and give further insights into the structure of observed delay values. For the predictive PLC controller experiment run, multiple samples yield distribution delay values above 150 ms. In the hardware controller experiment, much fewer samples above that threshold are observed, although the same FMITerminalBlock configuration is used. Due to polled, external hardware, fewer events are triggered and processed in the hardware controller experiment than in the PLC version. In particular, the timing analysis of the hardware controller experiment covers 292 events, while 378 events are processed for the PLC controller. Since both experiments cover approximately 70 s of simulation time, the average time between two events is proportionally larger in the hardware controller experiment. Figure 5.18 plots the histograms of inter-event time spans for both predictive experiments. The histograms show that the number of events triggered shortly after another event is much higher for the PLC experiment. Due to the high computational effort to solve the model,

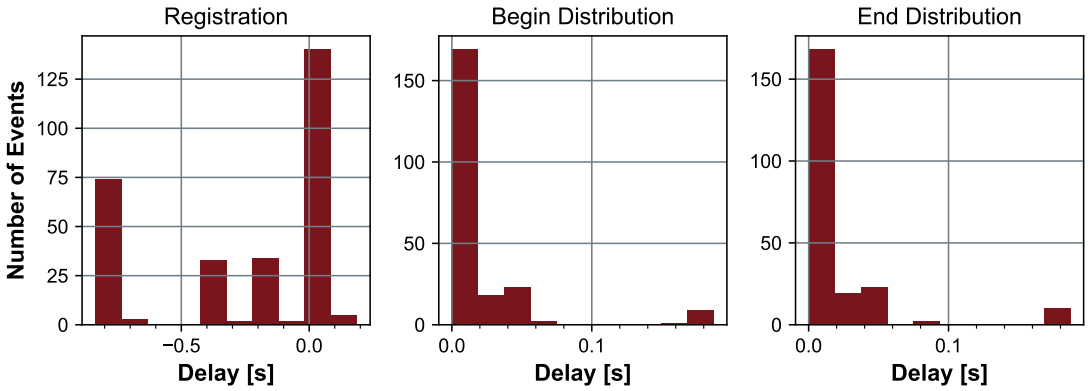
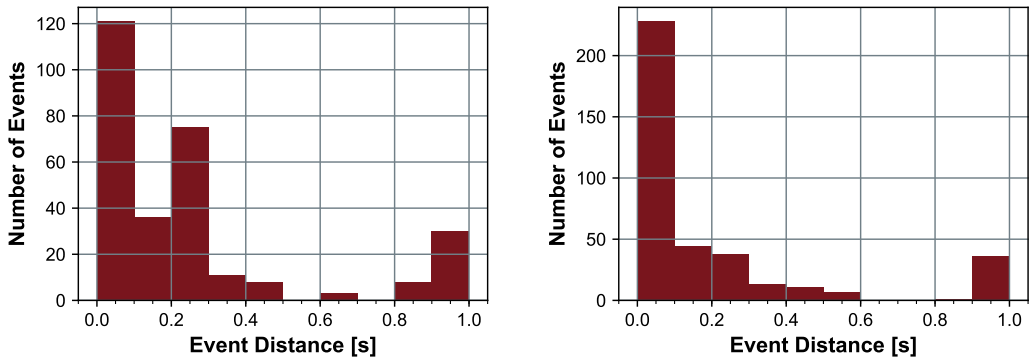


Figure 5.17: Histogram of observed execution delays (predictive, HW controller)



(a) HW controller

(b) PLC controller

Figure 5.18: Event distance distribution (predictive, closed-loop experiments)

such events often cannot be predicted and delivered in real time and contribute to high real-time delay values.

For the periodic configuration, observed maximum delay values of up to 47 ms are a lot smaller than for the predictive configuration. Note that the synchronization step size of the periodic approach is one order of magnitude lower than the look-ahead time. Consequently, a smaller simulation-time span needs to be processed in order to compute a single event. Since the computational effort of computing a single event is lower, also absolute delay values of the periodic experiment are reduced. In addition, the update procedure of applying external inputs to the model does not indirectly cause a computationally expensive prediction step and the real-time evaluation shows less accumulated delay of multiple late events. Due to the high frequency of emitted events, most of the 841 processed events are nominally delayed by the synchronization time of 100 ms.

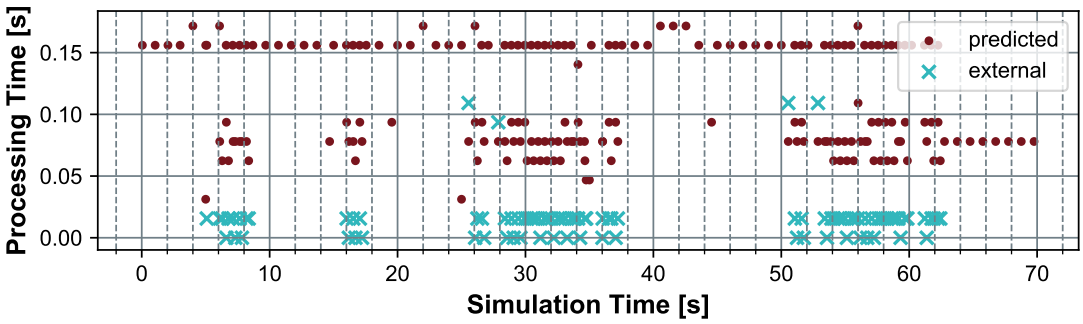


Figure 5.19: Duration of subsequent event predictions (predictive, PLC controller)

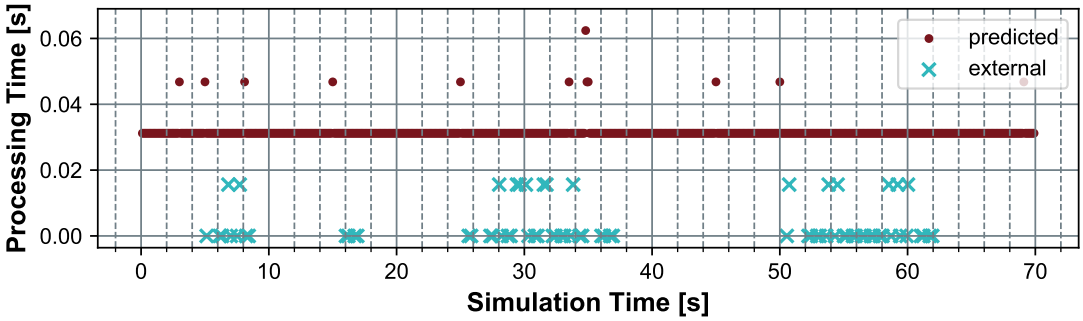


Figure 5.20: Duration of subsequent event predictions (periodic, PLC controller)

Figures 5.19 and 5.20 show the duration of the event prediction phase which follows the distribution of a given event. Points are separated by the type of the event which triggers the event prediction. Results of the periodic synchronization experiment in Figure 5.20 support the observation of an inexpensive model update process. For all external events, the time for applying changed inputs is lower than any observed prediction time. Most synchronization steps are solved within 32 ms and every step is solved faster than the real-time duration of 100 ms with a maximum prediction time of 62 ms.

In the predictive PLC controller experiment, a larger variation and higher prediction time values of up to 171 ms are observed. Both approaches show average processing time values for predicting a full period of 31 ms and 111 ms, respectively. Both average and maximum processing time of a single period are much higher for the predictive experiment but they stay well below the expectation of one order of magnitude. Processing time results indicate an increased overhead due to frequent event emission.

An external event in the predictive approach may be a trigger for processing a whole prediction interval but observed processing times mostly stayed well below the times for predicted events. Since direct dependency support is enabled for both experiments, each external event triggers an immediate reply. Such an event is generated by resetting the

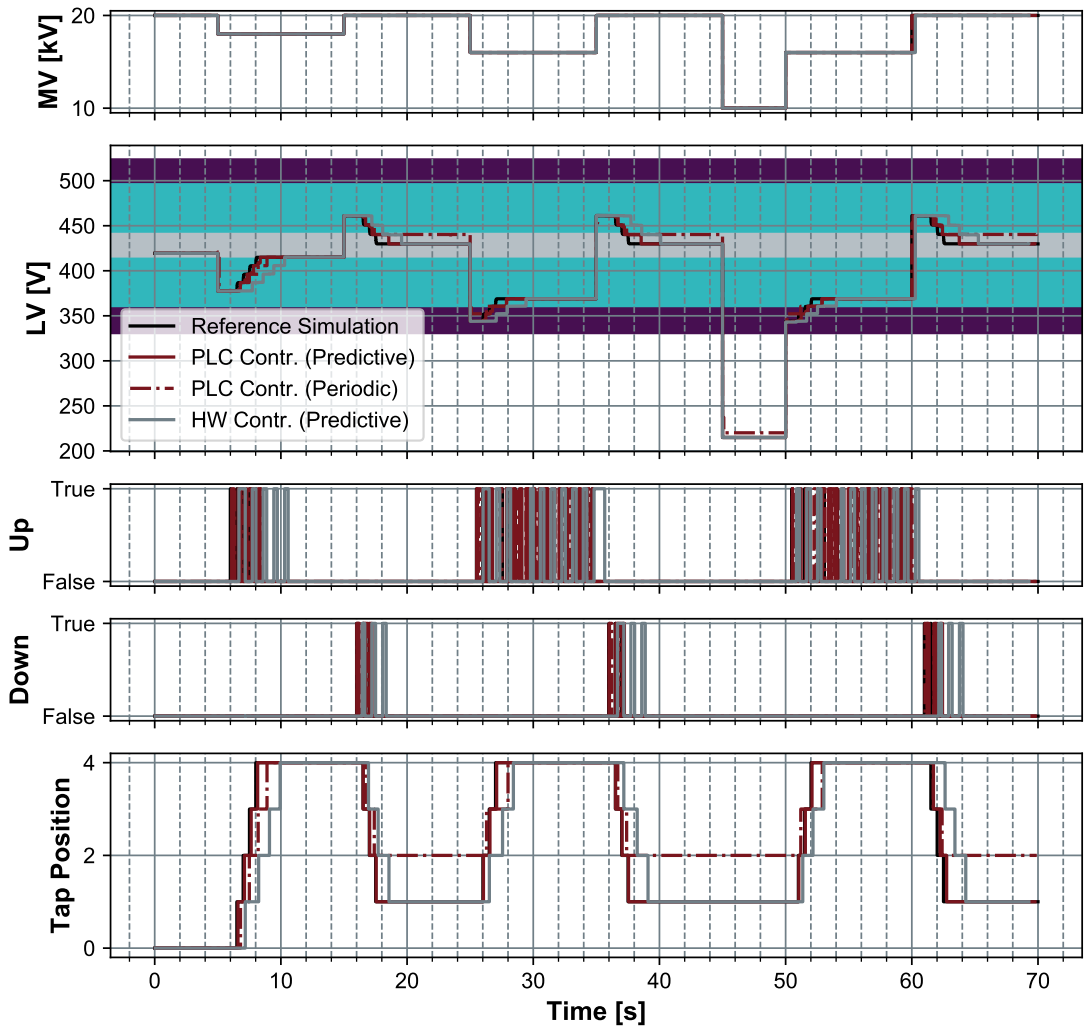


Figure 5.21: Overview of the closed-loop experiment

state of the model, applying external inputs, executing any events, and calculating the outputs. Note that it is not necessary to solve the continuous equations until the next event is detected, and, consequently, lower processing time values for external events are observed. The solution of the next prediction interval is postponed to the subsequent model event and the full processing time does not account for the external one.

Figure 5.21 shows the resulting signals of all three experiment runs and the reference simulation. For all experiments, the same MV pattern is applied in the model. Since the Simulink implementation does not directly trigger events on MV changes, slight deviations in the signal output are observed. Note that only the debugging output and not the internal representation of the MV value is affected. Each MV step, except for

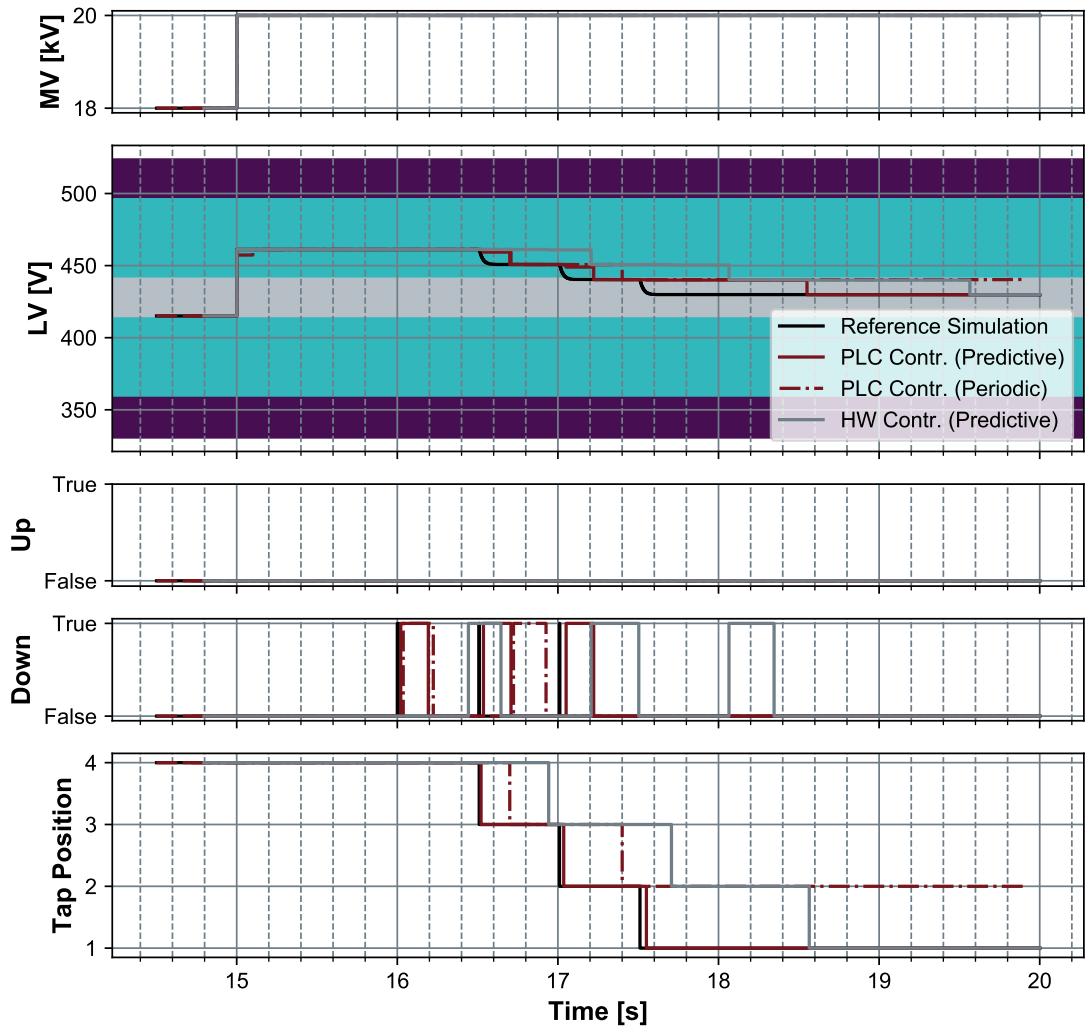


Figure 5.22: Exemplary switching operation

the undervoltage condition, causes multiple subsequent tap switching operations. A tap switching operation is initiated by a positive pulse on one control line. Blocks of overlapping pulses on the control line indicate transition phases in the experiment at which the controllers signal actions. Due to the implementation of the reference model and controller, commands may still be issued in case the minimum or maximum tap position is reached.

To further study the outcome of the experiments, one exemplary set of tap switches is plotted in Figure 5.22. Each experiment differs in duration of the positive pulses. While the reference simulation only shows short spikes of exactly one integration step, all other experiments show pulse widths in the order of a few hundred milliseconds. The width of

such a control pulse is directly influenced by communication and processing delays. As soon as a rising control edge is encountered, the model deasserts the ready signal and a controller ideally outputs a falling edge on the control line. To break algebraic loops, unit delays are inserted and the falling edge is issued one integration time step later than the rising one. All other experiments include communication steps, one which signals changed control commands and one which transports altered ready signal and voltage readings. Due to the asynchronous nature of subsequent control actions, communication and processing delays are directly reflected in the signal timing.

As expected, tap switches are issued first in the reference simulation, and the interval between single switching actions is solely influenced by the unit delay and the mechanical delay of 500 ms. In the exemplary switching operation, the first control action in the predictive PLC controller experiment arrives 21 ms after the reference signal and the model instantly issues a falling edge on the ready signal. Since previous events are still being processed, the falling edge of the ready signal is distributed late by up to 156 ms, which directly affects the reply of the first falling control signal arriving 193 ms after the reference simulation. Observed delays for falling control signals do not add up at multiple switching commands because the mechanical delay of the transformer is independent from the falling edge of that signal. Still, communication delays of each rising command edge add up between multiple switching commands.

When including a hardware controller in the closed-loop, communication delays in both directions drastically influence the rising edges on each control command. The effects of these delays can be directly seen in Figure 5.22 where the delays of the hardware correspond to the worst observed delays but still correspond to the expectations from a polling cycle of 200 ms.

Pulse width and delays are in the order of the other signals but slightly other delay mechanisms apply to the experiment with periodic configuration. The initial control pulse, for instance, is delayed by 37 ms, which is in the same time range as for the predictive counterpart and can be directly related to a communication delay between the PLC and the virtual controller. Due to the mode of operation, control inputs have to be delayed until the next synchronization point. In case of the first control action, the total input delay accounts for 100 ms until applied to the model. Since the periodic synchronization approach requires all outputs readily calculated before inputs are applied, the directly reset ready signal is issued at the subsequent synchronization point only and results in a total delay of 200 ms. Further 24 ms later, the cleared control signal from the software PLC arrives. Similarly, due to synchronization, the rising edge of the ready signal is delayed until the next synchronization point which manifests in 200 ms total delay compared to the reference simulation.

The last rows of Figures 5.22 and 5.21 indicate that, although no opposite control pulse is issued, a deviating final tap position is reached for the periodic virtual component. Both final voltage levels reside in the voltage band which does not indicate an action but, due to the different tap position, other final voltage levels are reached. Since the transformer model also covers the electrical characteristics, on switching taps, voltage

levels do not instantly decrease but are subject to continuous change. In most cases, no intermittent voltage reading is communicated due to the fast transition time and a reduced timing accuracy in comparison to the reference simulation.

In the reference simulation, the controller uses the voltage reading at the time instant when the ready signal is up and the tap is switched. Although the updated discrete tap position is still available, the electrical system has not reached a new steady state yet and the next control output is based on non-steady state voltage readings. A predictive virtual component uses the switching event to synchronize data which allows to issue the outputs at the same simulation time instant as the switching operation is conducted. Consequently, the correct (with respect to the reference simulation) transient states can be transferred, even if data exchange is late. Periodic synchronization delays the switching operation to the beginning of a synchronization period and outputs the results at the end. Due to the small electrical time constants, the steady state at the end of the synchronization period is (nearly) reached and the upcoming control decision is based on different states of the transformer. As a manifestation of the varying sampled state, the OLTC controller, which is connected to the periodic virtual component, issues one tap switch operation less.

On solely examining the real-time performance of the periodic approaches without any reference simulation, one has to add up to 187 ms per switching cycle. Since transformer model outputs often directly depend on its inputs, significant dynamics are introduced by the simulation method. Without an in-depth discussion, the detailed dynamics of a switching operation cannot be directly mapped between the reference simulation and the other experiments. Nevertheless, results indicate that system dynamics in the range of few seconds can be accurately represented.

5.5 Combined Results

To limit the impact of periodic synchronization, it is demonstrated that the synchronization intervals have to be a lot smaller than the look-ahead horizons of the predictive configurations. Consequently, the periodic configurations in both test cases trigger more events than the predictive configurations. Whether such an increased event emergence results in a higher number of actually distributed events, depends on the particular model. In test case 1, where all model outputs are discrete outputs, event filtering of FMITerminalBlock marks most of the additional events as redundant and avoids their distribution to remote devices. Contrary, the transformer model of test case 2 features a continuous output which rarely stays constant between two given synchronization points. Hence, the implemented simple event filtering mechanism is not able to drastically reduce emitted events and the periodic approach results in an increased workload of connected devices.

Both test cases showed different real-time delay distributions and even for a single test case, observed event delays depend on the particular configuration. In the first test case, the largest delay values observed are dominated by few outliers while in the second test

case, a higher share of large delay values is observed. In particular, the second test case demonstrated, that even for the same model, delays in different configurations can vary drastically. Due to decreased effort of solving a smaller synchronization interval, less absolute real-time delay is found in the periodic configuration of test case 2. In general, various real-time delay phenomena are observed and no synchronization approach clearly yields a better real-time performance.

Except for delays which are introduced by the real-time operation and small numerical errors, both test cases show matching results between the predictive configurations and the reference simulation. In particular, correct steady-state solutions of all predictive configurations are observed and no bias is introduced. Also, conventional periodic synchronization in test case 1 shows no deviating steady-state results, but test case 2 reveals a systematic bias due to periodic synchronization and closed-loop operation. In case results depend on sampling and communicating the model outputs at the point of time when a particular event occurs, delaying the communication until the next synchronization point can lead to a significant error. Especially, when generating an event at a transient state, outputs at the next synchronization point can deviate strongly from the outputs at the event. It is demonstrated that the predictive approach provides a viable alternative by communicating outputs at the event, instead of sampling the outputs at fixed intervals.

Access of external hardware is successfully demonstrated in both test cases. Due to the involved polling mechanism and the need of an additional protocol translation, a substantial increase in signal delay is observed. Since delays which are introduced by accessing connected hardware are accounted as external quantities, the delays cannot be identified by FMITerminalBlock itself. Hence, the deviations of control signals are evaluated with respect to a reference simulation. Detailed analysis of test case 1 reveals the occurrence of transiently inconsistent states, which are caused by non-atomic read and update operations of the protocol bridge and the controller, respectively. Due to the experimental design, the inconsistent states have no impact on the steady-state results and do not influence the outcomes of the methodological synchronization assessment.

CHAPTER 6

Conclusion and Outlook

Prior to any implementation, the research questions regarding potential ways of linking IEC 61499 and the FMI were tackled. In principle, a broad spectrum of coupling strategies exists. For instance, a controller may be encapsulated into an FMU or can utilize models which are exported via the FMI. Four principal ways of interaction were identified, each covering a general coupling strategy. For every principal way, different strategies to implement the general paradigm can be utilized. For example, a controller may synchronize an included FMI-based model periodically or it may use extrapolation to predict future states and may reset them, if necessary. For each principal way, multiple exemplary strategies for various aspects are given and discussed. Nevertheless, only an excerpt of relevant properties and strategies can be given in the scope of the thesis. A final qualitative comparison of the strategies indicates that no single coupling approach is able to serve all classes of use cases. One approach, for instance, may yield good real-time properties while another approach may be effectively used to model detailed controller timings in an offline simulation. A suitable coupling strategy must always be selected according to the targeted use case.

One potential way of creating virtual components via FMI-based models was successfully implemented by extending a dedicated software component. Although it was feasible to follow the initial software design of this component, several major modifications and amendments were necessary to implement a closed-loop operation. It is demonstrated that the software design tackling a predictive synchronization approach can also be used to flexibly implement periodic synchronization. Nevertheless, some limitations regarding the abstract network stack implementation were encountered. The implementation of protocols which require shared low-level connections may create considerable development overhead because the current design manages network entities separately. Nevertheless, the currently used ASN.1-based protocol implementation does not directly suffer from this limitation. Special attention has to be put on the queue management of exchanged events. It is shown that in particular late and equally timed events which prohibit a hard real-time

operation require dedicated processing steps. An event queuing and processing scheme is introduced which supports the implementation of the synchronization approaches and maintains several assumptions on the state of the event queue.

A proposed general timing and result assessment methodology is successfully applied to two dedicated test cases in the context of a smart transformer. Results from both test cases indicate that real-time operation of the virtual components is feasible in principle, but the accuracy of results strongly depends on timing properties of the processes. While it is not feasible to investigate transient electrical phenomena in a sub-milliseconds range, presented components can be used well to study effects in a seconds and sub-seconds range. Such processes may include electrical systems in steady states, thermal processes in the building sector, some mechanical, and a variety of chemical processes. Although the chosen test cases are specifically selected to challenge the approaches and to study the limits thereof, both test cases successfully demonstrate their capabilities in assessing the operation of controller hardware. On accessing external hardware, it is shown that special attention has to be put on the properties of the communication protocol which highly influences the timing.

The test cases also indicate that the real-time performance of open- and closed-loop setups strongly depend on the instantiated models and the configuration of the virtual components. For instance, the numerical integration method and the deployed synchronization strategies strongly affect delays and delay variances. In case the model cannot be solved accurately within the limits of real time, delays may accumulate and invalidate any result. Since accessed models and solvers, in general, do not provide any guarantees, a real-time evaluation after each experiment run is necessary. Presented test cases not only use the internal real-time assessment methodology of the developed interface component but also successfully validate timings based on external measurements. A detailed analysis of observed timing deviations and delays reveals major delay sources and effects which influence the outcome of the experiments. For instance, delays in the predictive approach are mainly caused by a violation of the idealistic hard real-time assumption that two events must not be triggered close to each other.

The test cases further present some ways of improving the performance of exported Simulink models by reducing the need for internal data buffers which decrease the simulation performance. Monolithic reference simulations are used to highlight the effects of the evaluated coupling approaches on the outcome of the experiment. Except for unavoidable timing deviations for transitive states, one experiment also showed varying steady-state results. The deviation can be tracked down to the operation principle of the conventional periodic synchronization approach and further reveals the relevance in properly selecting the simulation methodology.

Although the thesis successfully demonstrated the application of virtual components, further research and engineering effort may be conducted to broaden the applicability of the presented approaches. For instance, the networking stack may be refactored and extended to support a more versatile spectrum of communication protocols. Detailed performance benchmarks may be performed to further reveal and reduce processing

delays and to increase simulation performance. In particular, the influence of accessed libraries and operating system calls on the real-time performance is not well understood, yet. Network stacks and the scheduling mechanism may be improved, such that less delay and delay jitter is observed.

In order to reduce the effort in using FMITerminalBlock, the user interface can be substantially extended. Currently, it is necessary to configure a simulation run via command-line arguments. Dedicated configuration files or an interactive graphical user interface may guide an engineer in executing a simulation. Some efforts in testing the interoperability of FMUs are still required, too, to improve the confidence in the interface program. So far, FMITerminalBlock was mainly tested with dedicated test and Simulink FMUs. Testing the interface with FMUs by other vendors may reveal some details, which are not yet tackled. Further attention may be put to the description of use cases which do not have an adequate HIL library support and strongly benefit from FMI-based coupling.

The implementation of an adaptive event triggering mechanism, which automatically generates model events on continuous outputs, may further increase the accuracy of simulations. Similarly, the optional state retrieval feature of FMI 2.0 may be used to improve reset capabilities of the predictive approach. Since state retrieval would allow to extend the prediction beyond a single event, it is still to discuss, how the optional feature can be utilized best. A detailed study of influencing simulation parameters towards an automatic model parameter optimization may additionally advance the understanding of the described coupling strategies and ease the practical implementation. Another feature, which is not yet implemented, is automatic model transformation from FMUs to IEC 61499-based applications. Currently, coupling needs to be manually implemented by instantiating network SIFBs. Automatic model transformation may reduce the need for configuring the network connection and may tighten the integration of virtual components into automation systems.

Accessing external hardware via the presented methodologies may also receive further attention. For event-based systems, atomic network access and protocol implementations need to be studied and featured. Additionally, the reduction of communication delays and hardware access by other means than conventional polling may broaden the applicability of virtual components. Since FMITerminalBlock focuses on standard-based interfaces, it may be directly used to create virtual lab components inside an existing laboratory infrastructure which exceeds one controller under test. Nevertheless, practical implementation of such a setup is still open. Similarly, several other coupling strategies, which are described in Chapter 3, have not yet been implemented and may improve traditional assessment methodologies.

List of Figures

Bibliography

[1] Karsten Ahnert and Mario Mulansky. *Boost.Numeric.Odeint*. 2015. URL: http://www.boost.org/doc/libs/1_61_0/libs/numeric/odeint/doc/html/index.html (visited on 12/18/2017).

[2] Karsten Ahnert and Mario Mulansky. *ODEINT*. 2012. URL: http://headmyshoulder.github.io/odeint-v2/ (visited on 02/17/2018).

[3] Filip Andrén, Sawsan Henein, and Matthias Stifter. "Development and validation of a coordinated voltage controller using real-time simulation". In: *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*. Nov. 2011, pp. 3713–3718. DOI: 10.1109/IECON.2011.6119913.

[4] Filip Andrén, Felix Lehfuß, and Thomas Strasser. "A Development and Validation Environment for Real-time Controller-hardware-in-the-loop Experiments in Smart Grids". In: *International Journal of Distributed Energy Resources and Smart Grids* 9.1 (Hardware-in-the-loop Testing July 2013), pp. 27–50. ISSN: 1614-7138.

[5] Filip Andrén, Matthias Stifter, and Thomas Strasser. "Towards a Semantic Driven Framework for Smart Grid Applications: Model-Driven Development Using CIM, IEC 61850 and IEC 61499". English. In: *Informatik-Spektrum* 36.1 (2013), pp. 58–68. ISSN: 0170-6012. DOI: 10.1007/s00287-012-0663-y.

[6] Michêle Arnold and Göran Andersson. "Model Predictive Control of Energy Storage including Uncertain Forecasts". In: *17th Power Systems Computation Conference*. Stockholm, Sweden, 2011.

[7] *Atmel ATmega640/V–1280/V–1281/V–2560/V–2561/V*. Atmel Corporation. 2014. URL: http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf (visited on 12/19/2017).

[8] Muhammad Usman Awais. "Distributed hybrid co-simulation". Dissertation. Technische Universität Wien, 2015.

[9] Muhammad Usman Awais, Peter Palensky, Wolfgang Mueller, Edmund Widl, and Atiyah Elsheikh. "Distributed hybrid simulation using the HLA and the Functional Mock-up Interface". In: *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*. 2013, pp. 7564–7569. DOI: 10.1109/IECON.2013.6700393.

[10] Muhammad Usman Awais, Peter Palensky, Atiyah Elsheikh, Edmund Widl, and Matthias Stifter. “The high level architecture RTI as a master to the functional mock-up interface components”. In: *Computing, Networking and Communications (ICNC), 2013 International Conference on*. Jan. 2013, pp. 315–320. DOI: 10.1109/ICCNC.2013.6504102.

[11] Marko Bacic. “On hardware-in-the-loop simulation”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. 2005, pp. 3194–3198. DOI: 10.1109/CDC.2005.1582653.

[12] Stefan Biffl, Alexander Schatten, and Alois Zoitl. “Integration of heterogeneous engineering environments for the automation systems lifecycle”. In: *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009, pp. 576–581. DOI: 10.1109/INDIN.2009.5195867.

[13] Torsten Blochwitz, Martin Otter, Martin Arnold, C Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, M Monteiro, T Neidhold, et al. “The functional mockup interface for tool independent exchange of simulation models”. In: *8th International Modelica Conference, Dresden*. 2011, pp. 20–22. URL: https://www.modelica.org/events/modelica2011/Proceedings/pages/papers/05_1_ID_173_a_fv.pdf (visited on 07/04/2017).

[14] David Broman, Christopher Brooks, Edward A. Lee, Thierry Stephane Nouidui, Stavros Tripakis, and Michael Wetter. *JFMI - A Java Wrapper for the Functional Mock-up Interface*. Apr. 2013. URL: http://ptolemy.eecs.berkeley.edu/java/jfmi/ (visited on 02/17/2018).

[15] *Building Controls Virtual Test Bed*. Lawrence Berkeley National Laboratory. 2016. URL: http://simulationresearch.lbl.gov/bcvtb (visited on 02/05/2018).

[16] *CarMaker. Virtual Testing of Automobiles and Light-Duty Vehicles*. IPG Automotive GmbH. URL: https://ipg-automotive.com/?id=266 (visited on 07/07/2017).

[17] *ControlBuild. Designing Automation and Embedded Control Systems*. Dassault Systèmes. 2017. URL: https://www.3ds.com/products-services/catia/products/controlbuild/.

[18] Fabio Cremona, Marten Lohstroh, Stavros Tripakis, Christopher Brooks, and Edward A. Lee. “FIDE: An FMI Integrated Development Environment”. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. SAC ’16. Pisa, Italy: ACM, 2016, pp. 1759–1766. ISBN: 978-1-4503-3739-7. DOI: 10.1145/2851613.2851677.

[19] *DIN EN 61131-3 Speicherprogrammierbare Steuerungen. Teil 3: Programmiersprachen (IEC 61131-3:2013)*. DIN/VDE-DKE Deutsche Kommission Elektrotechnik Elektronik Informationstechnik, June 2014.

[20] Gerhard Doblinger. *Zeitdiskrete Signale und Systeme; eine Einführung in die grundlegenden Methoden der digitalen Signalverarbeitung*. 2., überarb. Aufl. Wilburgstetten: Schlembach, 2010. ISBN: 978-3-935340-66-3.

[21] Michael Drmota, Bernhard Gittenberger, Günther Karigl, and Alois Panholzer. *Mathematik für Informatik*. 2nd ed. Vol. 17. Berliner Studienreihe zur Mathematik. Lemgo: Heldermann, 2008. ISBN: 978-3-88538-117-4.

[22] *Eclipse 4diac - The Open Source Environment for Distributed Industrial Automation and Control Systems*. Eclipse Foundation. URL: http://www.eclipse.org/4diac/ (visited on 07/03/2017).

[23] Johan Eker, Jörn W. Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. "Taming heterogeneity – the Ptolemy approach". In: *Proceedings of the IEEE* 91 (1 Jan. 2003), pp. 127–144. ISSN: 0018-9219. DOI: 10.1109/JPROC.2002.805829.

[24] Gisela Engeln-Müllges, Klaus Niederdrenk, and Reinhard Wodicka. *Numerik-Algorithmen. Verfahren, Beispiele, Anwendungen*. 9th ed. Xpert.press. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2005. ISBN: 9783540263531.

[25] *ERIGrid – H2020 Research Infrastructure Project*. 2017. URL: https://erigrid.eu/ (visited on 12/15/2017).

[26] Charlie Erwall and Oscar Mårtensson. "Model-based design of industrial automation solutions using FMI". eng. master. Sweden: Department of Automatic Control, Lund University, 2016. URL: http://lup.lub.lu.se/student-papers/record/8894129 (visited on 07/07/2017).

[27] *Factories 4.0 and Beyond. Recommendations for the work programme 18-19-20 of the FoF PPP under Horizon 2020*. European Factories of the Future Research Association. Sept. 2016. URL: http://www.effra.eu/sites/default/files/factories40_beyond_v31_public.pdf (visited on 01/23/2018).

[28] *Factories of the Future. Multi-annual roadmap for the contractual PPP under Horizon 2020*. 2013. ISBN: 978-92-79-31238-0. DOI: 10.2777/29815. URL: http://www.effra.eu/sites/default/files/factories_of_the_future_2020_roadmap.pdf (visited on 01/23/2018).

[29] *FBDK - The Function Block Development Kit*. Holobloc Inc. Mar. 2011. URL: http://www.holobloc.com/doc/fbdk/index.htm (visited on 07/03/2017).

[30] *FBDK 2.6 - The Function Block Development Kit*. Holobloc Inc. Mar. 2017. URL: http://www.holobloc.com/fbdk2/index.htm (visited on 07/03/2017).

[31] *FMI Kit for Simulink. version 2.4.0*. Dassault Systèmes. 2017. URL: https://www.3ds.com/fileadmin/PRODUCTS/CATIA/DYMOLA/PDF/FMI_Kit_for_Simulink.pdf (visited on 12/18/2017).

[32] *FMI Library*. Modelon AB. 2014. URL: http://www.jmodelica.org/FMILibrary (visited on 02/17/2018).

[33] *FMI Support in Tools*. Modelica Association Project. 2017. URL: https://www.fmi-standard.org/tools (visited on 03/24/2017).

[34] *FMI Use Case - dSpace. Integrating Functional Mock-up Units for HIL Simulation*. dSPACE GmbH. 2017. URL: https://www.dspace.com/en/pub/home/products/hw/simulator_hardware/scalexio/scalexio_fmi/hil_fmi.cfm (visited on 07/07/2017).

[35] *FMITerminalBlock: FMI - Fieldbus Interface*. AIT Austrian Institute of Technology GmbH. 2017. URL: https://github.com/AIT-IES/FMITerminalBlock (visited on 12/13/2017).

[36] *FMU SDK 2.0.5*. QTronic GmbH. 2018. URL: http://www.qtronic.de/en/fmusdk.html (visited on 02/17/2018).

[37] *FORTE - 4diac runtime environment*. Eclipse Foundation. URL: http://www.eclipse.org/4diac/en_rte.php (visited on 07/03/2017).

[38] *Functional Mock-up Interface for Co-Simulation*. Version 1.0. Modelica Association Project. MODELISAR consortium, Oct. 2010. URL: https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_CoSimulation_v1.0.pdf (visited on 02/17/2018).

[39] *Functional Mock-up Interface for Model Exchange*. Version 1.0. Modelica Association Project. MODELISAR consortium, Jan. 2010. URL: https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_ModelExchange_v1.0.pdf (visited on 02/17/2018).

[40] *Functional Mock-up Interface for Model Exchange and Co-Simulation*. Version 2.0. Modelica Association Project. MODELISAR consortium, July 2014. URL: https://svn.modelica.org/fmi/branches/public/specifications/v2.0/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf (visited on 03/28/2017).

[41] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, and Fabrice Rossi. *GNU Scientific Library Reference Manual (3rd Ed.)* Ed. by Brian Gough. 2009. URL: http://www.gnu.org/software/gsl/ (visited on 02/17/2018).

[42] Sara Gunnarsson. "Evaluation of FMI-based workflow for simulation and testing of industrial automation applications". eng. master. Sweden: Department of Automatic Control, Lund University, 2016. URL: http://lup.lub.lu.se/student-papers/record/8776878 (visited on 07/07/2017).

[43] Feng Guo, Luis Herrera, Robert Murawski, Ernesto Inoa, Chih-Lun Wang, Philippe Beauchamp, Eylem Ekici, and Jin Wang. "Comprehensive Real-Time Simulation of the Smart Grid". In: *Industry Applications, IEEE Transactions on* 49.2 (Mar. 2013), pp. 899–908. ISSN: 0093-9994. DOI: 10.1109/TIA.2013.2240642.

[44] Reinhard Hametner. "Modellierung von Regelungsstrategien in einer event-basierten Echtzeitsteuerungsumgebung". Technische Universität Wien, 2008.

[45] Ingo Hegny. “Development and simulation framework for industrial production systems”. Technische Universität Wien, 2014.

[46] Ingo Hegny, Monika Wenger, and Alois Zoitl. “IEC 61499 based simulation framework for model-driven production systems development”. In: *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. Sept. 2010, pp. 1–8. DOI: 10.1109/ETFA.2010.5641364.

[47] Ingo Hegny, Alois Zoitl, and Wilfried Lepuschitz. “Integration of simulation in the development process of distributed IEC 61499 control applications”. In: *Industrial Technology, 2009. ICIT 2009. IEEE International Conference on*. Feb. 2009, pp. 1–6. DOI: 10.1109/ICIT.2009.4939681.

[48] José Juan Hernández-Cabrera, José Évora Gómez, and Octavio Roncal-Andrés. *javaFMI*. Sept. 2016. URL: https://bitbucket.org/siani/javafmi/wiki/Home (visited on 02/17/2018).

[49] Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. “SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers”. In: *ACM Transactions on Mathematical Software (TOMS)* 31.3 (2005), pp. 363–396.

[50] *IEC 61499-1/Ed.2: Function blocks - Part 1: Architecture*. International Electrotechnical Commission, IEC, Nov. 2012.

[51] *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules*. Aug. 2010, pp. 1–38. DOI: 10.1109/IEEESTD.2010.5553440.

[52] *Jenkins*. 2017. URL: https://jenkins.io/ (visited on 12/13/2017).

[53] Simon Josefsson and Nikos Mavrogiannopoulos. *GNU Libtasn1*. Free Software Foundation, Inc. Jan. 2018. URL: https://www.gnu.org/software/libtasn1/ (visited on 02/17/2018).

[54] Andi Kleen and Michael Kerrisk. *TCP (7) Linux Programmer's Manual*. Linux man-pages project. Sept. 2017. URL: http://man7.org/linux/man-pages/man7/tcp.7.html (visited on 02/17/2018).

[55] Hermann Kopetz. *Real-Time Systems; Design Principles for Distributed Embedded Applications*. Real-Time Systems Series. Boston, MA: Springer Science+Business Media, LLC, 2011. ISBN: 9781441982377. DOI: 10.1007/978-1-4419-8237-7.

[56] Henning Kroll, Giacomo Copani, Els Van de Velde, Magnus Simons, Djerdj Horvat, Angela Jäger, Annelies Wastyn, Golboo PourAbdollahian, and Mika Naumanen. *An analysis of drivers, barriers and readiness factors of EU companies for adopting advanced manufacturing products and technologies*. Sept. 2016. ISBN: 978-92-79-64467-2. DOI: 10.2873/715340. URL: https://ec.europa.eu/docsroom/documents/20926/attachments/1/translations/en/renditions/native (visited on 01/19/2018).

[57] *LABCAR-OPERATOR V5.4.2. User's Guide*. ETAS GmbH. 2016. URL: https://www.etas.com/download-center-files/products_LABCAR_Software_Products/LABCAR-OPERATOR_V5.4.2_UsersGuide.pdf (visited on 07/07/2017).

[58] *Layered Model-View-Control Design Pattern*. Holobloc Inc. Jan. 2011. URL: http://www.holobloc.com/doc/despats/mvc/ (visited on 07/03/2017).

[59] Edward A. Lee. "Cyber Physical Systems: Design Challenges". In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[60] *libmodbus*. URL: http://libmodbus.org/ (visited on 02/17/2018).

[61] Per Lindgren, Marcus Lindner, Andreas Lindner, Valeriy Vyatkin, David J. Pereira, and Luis Miguel Pinho. "A Real-Time Semantics for the IEC 61499 standard". In: Luxembourg, Sept. 2015. ISBN: 978-1-4673-7929-8. URL: http://ltu.diva-portal.org/smash/get/diva2:1010414/FULLTEXT01.pdf (visited on 07/03/2017).

[62] Jan Lunze. *Regelungstechnik 1; Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. 9., überarbeitete Aufl. 2013. Springer-Lehrbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-29533-1.

[63] Jan Lunze. *Regelungstechnik 2; Mehrgrößensysteme, Digitale Regelung*. Springer-Lehrbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN: 978-3-642-10197-7. DOI: 10.1007/978-3-642-10198-4.

[64] *Matplotlib: Python plotting – Matplotlib 2.1.1 documentation*. 2017. URL: http://matplotlib.org/ (visited on 12/18/2017).

[65] *Modbus Technical Resources*. Modbus Organization, Inc. URL: http://www.modbus.org/tech.php (visited on 12/19/2017).

[66] *Model.CONNECT^{TM}. Integration of virtual and real components*. AVL LIST GmbH. 2017. URL: https://www.avl.com/en/iodp/-/asset_publisher/MQahPiTr3eTp/content/model-connect- (visited on 07/07/2017).

[67] Sven Christian Müller, Hanno Georg, Markus Küch, and Christian Wietfeld. "INSPIRE - Co-Simulation of Power and ICT Systems for Evaluation of Smart Grid Applications". In: *At-Automatisierungstechnik* 62(5) (Apr. 2014), 315–324. ISSN: 0178-2312. DOI: 10.1515/auto-2014-1086.

[68] Wolfgang Müller and Edmund Widl. "Linking FMI-based components with discrete event systems". In: *Systems Conference (SysCon), 2013 IEEE International*. Apr. 2013, pp. 676–680. DOI: 10.1109/SysCon.2013.6549955.

[69] Heikki Nikula, Eero Vesaoja, Seppo Sierla, Tommi Karhela, Paul G. Flikkema, Antti Aikala, Tuomas Miettinen, and Chen-Wei Yang. "Co-simulation of a dynamic process simulator and an event-based control system: Case district heating system". In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. Sept. 2014, pp. 1–7. DOI: 10.1109/ETFA.2014.7005184.

[70] *NumPy v1.13 Manual*. The Scipy community. 2017. URL: https://docs.scipy.org/doc/numpy/ (visited on 12/18/2017).

[71] *nxtControl – nxtSTUDIO*. nxtControl GmbH. 2017. URL: http://www.nxtcontrol.com/engineering/ (visited on 07/03/2017).

[72] *Open Virtual Platforms^{TM} (OVP^{TM}) portal*. Imperas Software Limited. URL: http://www.ovpworld.org/ (visited on 02/17/2018).

[73] *Ormazabal | Reliable innovation. Personal solutions.* 2017. URL: https://www.ormazabal.com/ (visited on 12/15/2017).

[74] *pandas: powerful Python data analysis toolkit*. 2017. URL: http://pandas.pydata.org/pandas-docs/stable/ (visited on 12/18/2017).

[75] Xiufeng Pang, Thierry S. Nouidui, Michael Wetter, Daniel Fuller, Anna Liao, and Philip Haves. "Building energy simulation in real time through an open standard interface". In: *Energy and Buildings* 117 (Apr. 2016), pp. 282–289. ISSN: 0378-7788.

[76] J.S.C. Prentice. "Numerical differentiation – a general purpose algorithm". In: *International Journal of Mathematical Education in Science and Technology* 44.1 (2013), pp. 116–122. DOI: 10.1080/0020739X.2012.662296.

[77] Claudius Ptolemaeus, ed. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. ISBN: 978-1-304-42106-7. URL: http://ptolemy.org/books/Systems (visited on 02/05/2018).

[78] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. *SESC simulator*. 2005. URL: http://sesc.sourceforge.net (visited on 10/06/2014).

[79] Sebastian Rohjans, Edmund Widl, Wolfgang Müller, Steffen Schütte, and Sebastian Lehnhoff. "Gekoppelte simulation komplexer energiesysteme mittels mosaik und FMI". In: *At-Automatisierungstechnik* 62 (5 May 2014), pp. 325–336. ISSN: 01782312. DOI: 10.1515/auto-2014-1087.

[80] *Rolling Plan for ICT Standardisation*. European Commission. 2017. URL: https://ec.europa.eu/docsroom/documents/24846/attachments/1/translations/en/renditions/native (visited on 01/19/2018).

[81] Bernhard Rumpe. *Modellierung mit UML. Sprache, Konzepte und Methodik*. 2nd ed. Xpert.press. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2011. ISBN: 9783642224133. DOI: 10.1007/978-3-642-22413-3.

[82] Martin Schlager. "Interface Design for Hardware-in-the-Loop Simulation of Real-Time Systems". Disertation. Technischen Universität Wien, Fakultät für Informatik, Sept. 2007.

[83] *Simscape Power Systems. Model and simulate electrical power systems.* The MathWorks, Inc. 2018. URL: https://www.mathworks.com/products/simpower.html (visited on 01/11/2018).

[84] *SIMulation Workbench.* SIMulation Workbench. URL: http://wiki.simwb.com/wiki/Main_Page (visited on 07/07/2017).

[85] Michael H. Spiegel. *Integrating the Functional Mockup Interface into IEC 61499-based components.* Tech. rep. 183/1-175. A-Lab @ Automation Systems Group, TU Vienna, Nov. 2015, p. 85. URL: http://www.auto.tuwien.ac.at/bib/pdf_TR/TR0175.pdf (visited on 03/24/2017).

[86] Michael H. Spiegel, Fabian Leimgruber, Edmund Widl, and Günther Gridling. "On using FMI-based models in IEC 61499 control applications". In: *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2015 Workshop on.* Apr. 2015, pp. 1–6. DOI: 10.1109/MSCPES.2015.7115407.

[87] Stefan Stattelmann, Oliver Bringmann, and Wolfgang Rosenstiel. "Fast and Accurate Source-level Simulation of Software Timing Considering Complex Code Optimizations". In: *Proceedings of the 48th Design Automation Conference.* DAC '11. San Diego, California: ACM, 2011, pp. 486–491. ISBN: 978-1-4503-0636-2. DOI: 10.1145/2024724.2024838.

[88] Matthias Stifter, Edmund Widl, Filip Andrén, Atiyah Elsheikh, Thomas Strasser, and Peter Palensky. "Co-simulation of components, controls and power systems based on open source software". In: *Power and Energy Society General Meeting (PES), 2013 IEEE.* July 2013, pp. 1–5. DOI: 10.1109/PESMG.2013.6672388.

[89] Thomas Strasser, F. Auinger, and Alois Zoitl. "Development, implementation and use of an IEC 61499 function block library for embedded closed loop control". In: *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on.* June 2004, pp. 594–599. DOI: 10.1109/INDIN.2004.1417415.

[90] Thomas Strasser, Matthias Stifter, Filip Andrén, Daniel Burnier de Castro, and Wolfgang Hribernik. "Applying open standards and open source software for smart grid applications: Simulation of distributed intelligent control of power systems". In: *Power and Energy Society General Meeting, 2011 IEEE.* July 2011, pp. 1–8. DOI: 10.1109/PES.2011.6039314.

[91] Thomas Strasser, Matthias Stifter, Filip Andrén, and Peter Palensky. "Co-Simulation Training Platform for Smart Grids". In: *Power Systems, IEEE Transactions on* 29.4 (July 2014), pp. 1989–1997. ISSN: 0885-8950. DOI: 10.1109/TPWRS.2014.2305740.

[92] Thomas Strasser, Alois Zoitl, James H. Christensen, and Christoph Sünder. "Design and Execution Issues in IEC 61499 Distributed Automation and Control Systems". In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 41.1 (Jan. 2011), pp. 41–51. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2010.2067210.

[93] *Technology Roadmap. Smart Grids*. International Energy Agency. 2011. URL: http://www.iea.org/publications/freepublications/publication/smartgrids_roadmap.pdf (visited on 01/23/2018).

[94] *Technology Roadmap. Solar Photovoltaic Energy*. International Energy Agency. 2014. URL: http://www.iea.org/media/freepublications/technologyroadmaps/solar/TechnologyRoadmapSolarPhotovoltaicEnergy_2014edition.pdf (visited on 01/23/2018).

[95] *The FMI++ Library – Version 1.0. Documentation*. AIT Austrian Institute of Technology GmbH. Feb. 2015. URL: https://sourceforge.net/projects/fmipp/files/fmipp_doc_v2015-02-11-17-29.pdf/download (visited on 02/17/2018).

[96] *The FMI++ Library / Code / [b81b84] /import/integrators/include/IntegratorStepper.h*. AIT Austrian Institute of Technology GmbH. 2017. URL: https://sourceforge.net/p/fmipp/code/ci/master/tree/import/integrators/include/IntegratorStepper.h (visited on 12/13/2017).

[97] *The FMI++ Library / Code / [b81b84] /import/integrators/include/IntegratorType.h*. AIT Austrian Institute of Technology GmbH. 2017. URL: https://sourceforge.net/p/fmipp/code/ci/master/tree/import/integrators/include/IntegratorType.h (visited on 12/12/2017).

[98] *TISC Suite — Software zur Kopplung mehrerer Simulationswerkzeuge*. TLK-Thermo GmbH. 2016. URL: https://www.tlk-thermo.com/index.php/de/softwareprodukte/tisc-suite (visited on 07/06/2017).

[99] *Transport Delay*. MathWorks, Inc. 2017. URL: https://de.mathworks.com/help/simulink/slref/transportdelay.html (visited on 12/20/2017).

[100] *Typhoon HIL604*. Typhoon HIL GmbH. 2018. URL: https://www.typhoon-hil.com/doc/products/Typhoon-HIL604-brochure.pdf (visited on 01/26/2018).

[101] Alexander Viehweider, Georg Lauss, and Felix Lehfuss. "Stabilization of Power Hardware-in-the-Loop simulations of electric energy systems". In: *Simulation Modelling Practice and Theory* 19.7 (2011), pp. 1699 –1708. ISSN: 1569-190X. DOI: 10.1016/j.simpat.2011.04.001.

[102] Valeriy Vyatkin. "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review". In: *Industrial Informatics, IEEE Transactions on* 7.4 (Nov. 2011), pp. 768–781. ISSN: 1551-3203. DOI: 10.1109/TII.2011.2166785.

[103] Valeriy Vyatkin. "The IEC 61499 standard and its semantics". In: *Industrial Electronics Magazine, IEEE* 3.4 (Dec. 2009), pp. 40–48. ISSN: 1932-4529. DOI: 10.1109/MIE.2009.934796.

[104] *W5100 Datasheet*. Version 1.2.7. WIZnet Co., Inc. 2011. URL: http://www.wiznet.io/wp-content/uploads/wiznethome/Chip/W5100/Document/W5100_Datasheet_v1.2.7.pdf (visited on 12/19/2017).

[105] Lev Walkin. *Open Source ASN.1 Compiler*. 2017. URL: http://lionet.info/asn1c/compiler.html (visited on 02/17/2018).

[106] Christian Walter. *FreeMODBUS - A Modbus ASCII/RTU and TCP implementation*. embedded solutions. June 2010. URL: http://www.freemodbus.org/ (visited on 02/17/2018).

[107] Michael Wetter and Philip Haves. "A Modular Building Controls Virtual Test Bed for the Integrations of Heterogeneous Systems". In: *SimBuild 2008*. June 2008. URL: http://www.osti.gov/scitech/servlets/purl/936246 (visited on 02/05/2018).

[108] Edmund Widl, Wolfgang Müller, Atiyah Elsheikh, Matthias Hörtenhuber, and Peter Palensky. "The FMI++ library: A high-level utility package for FMI for model exchange". In: *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*. May 2013, pp. 1–6. DOI: 10.1109/MSCPES.2013.6623316.

[109] Chia-Han Yang, Valeriy Vyatkin, and Valeriy Cheng Pang. "Model-Driven Development of Control Software for Distributed Automation: A Survey and an Approach". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44(3) (May 2014), pp. 292–305. ISSN: 2168-2216.

[110] Chia-Han Yang, Gulnara Zhabelova, Chen-Wei Yang, and Valeriy Vyatkin. "Cosimulation Environment for Event-Driven Distributed Controls of Smart Grid". In: *Industrial Informatics, IEEE Transactions on* 9.3 (Aug. 2013), pp. 1423–1435. ISSN: 1551-3203. DOI: 10.1109/TII.2013.2256791.

[111] Sherali Zeadally, Liqiang Zhang, Zhaoming Zhu, and Jian Lu. "Network application programming interfaces (APIs) performance on commodity operating systems". In: *Information and Software Technology* 46.6 (2004), pp. 397–402. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2003.08.005.

[112] Josef Zehetner, Georg Stettinger, Helmut Kokal, and Bart Toye. "Echtzeit-Co-Simulation für die Regelung eines Motorprüfstands". In: *ATZ – Automobiltechnische Zeitschrift* 116.2 (2014), pp. 40–45. ISSN: 2192-8800. DOI: 10.1007/s35148-014-0042-x.