DIPLOMARBEIT

Applying Model-Integrated Computing on Time-Triggered Application Development

ausgeführt zum Zwecke der Erlangung des
akademischen Grades eines

Diplom - Ingenieurs

am

Institut für Technische Informatik 182/1

der

Technischen Universität Wien

unter der Leitung von

O. Univ.-Prof. Dr. phil Hermann Kopetz
und
Univ. Ass. Dipl. - Ing. Dr. Wilfried Elmenreich

durch

Bakk. techn. Christian Paukovits
Matr. - Nr. 0127145
Linzer Straße 429/5/5204, 1140 Wien

Wien, im Mai 2006 .....

Applying Model-Integrated Computing on Time-Triggered Application Development

Designing and implementing a distributed embedded real-time system is a challenging task. A possible approach to cope with the complexity is to employ tools in the design and implementation process, that relieve the designer from lower level issues like matching system parameters and determining communication message schedules.
This thesis presents a variation of the model-based tool suite *Generic Modeling Environment* (GME) for the time-triggered real-time communication system TTP/A.
The tool builds on the *Generic Modeling Environment* (GME) an integrated modelling tool that supports a meta-level design technology. Due to the generic nature of the tool suite, the work presented in this thesis also applies, in general, to other real-time communication systems with minor modifications.
Besides this, this work presents a comprehensive definition of the conceptual model of TTP/A applications, which makes up the major theoretical part.

Model-Integrated Computing für die Entwicklung zeitgesteuerter Applikationen in verteilten Echtzeitsystemen

Der Entwurf und die Implementierung von verteilten, eingebetteten Echtzeitsystemen stellt eine besondere Herausforderung für Konstrukteure dar. Ein möglicher Ansatz, um die Komplexität beherrschen zu können, ist der Einsatz von Software-Tools, die den Prozess des Entwurfs und der Implementierung unterstützen. Folglich wird der Entwickler durch die Automatisierung der zugrunde liegenden Eigenschaften des Zielsystems entlastet, wie beispielsweise die Berechnung von Konfigurationsparametern für das System oder die Generierung von Nachrichten- und Jobfahrplänen.
Diese Diplomarbeit präsentiert eine Variation der Modell-basierten Tool-Suite *Generic Modeling Environment* (GME) auf Basis des zeitgesteuerten Echtzeit-Kommunikationssystems TTP/A.
Aufbauend auf der *Generic Modeling Environment* (GME) wird ein neues, integriertes Modellierungswerkzeug geschaffen, das Meta-Level Modellierung bietet. Auf Grund der Generizität der Tool-Suite ist die hier vorgestellte Arbeit auch für beliebige Echtzeit-Kommunikationssysteme mit geringfügigen Anpassungen anwendbar.
Darüber hinaus beinhaltet diese Arbeit eine umfassende Abhandlung über das konzeptuelle Model von TTP/A Applikationen, was den theoretischen Hauptteil dieser Arbeit ausmacht.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software Engineering is a relatively young field in technical sciences, which has evolved during the past few decades. This is a short time compared to the mature, well-established engineering fields like architecture or engine construction, which have undergone centuries or even millennia of evolution. Over the time those engineering fields fostered a huge knowledge base and several design principles, and industrial standards like the DIN manifested.

Albeit notable exceptions such as UML, software engineering lacks of uniform and standardized methodologies of designing software components. Due to fast technology life cycles and different problem solution approaches, we cannot expect the validity of any know-how over some time. Thus, the details of implementation are subject to rapid changes, and techniques become out-dated.

It is impossible for a human being, in that case the software engineer, to keep pace with all developments in his or her field. There is one possibility to come across that limitation of human cognitive performance: **simplification** or **abstraction**. A given system is not described in its details of implementation and specific technology, but in a higher-level abstract formalism.

For instance, the software engineering community produced a unified modelling language, which explains structure, interrelation, use cases etc. for software components: *UML* [UML04]. This notational style abstracts from concrete programming languages, used middle-ware or target technologies as well as programming paradigms. Conversely, it is a formal base, from which all these implementation specific details can be *derived*. For example, the computer aided software engineering tools (CASE tools) are able to produce software components from a higher-level specification like UML.

According to its origin, UML had been mainly designed for software development projects. As this thesis deals with distributed embedded real-time systems, we would like to mention more "hardware-level" approaches. For instance, the field of chip design provoked such higher-level description formalism by means of the programming language C++, namely *SystemC* [Sys05]. The design of some circuitry is not expressed in a hardware description language like VHDL [VHD00], but in C++, which entails a more "software-engineering-like" point of view on hardware design.

Before we proceed, we should think about the reasons, **why** we would like to use abstraction respectively modelling in the development of distributed embedded systems.

1.1 The struggle of designing

Setting up a distributed embedded system may often be a sumptuous task. In distributed embedded real-time systems the sources of possibles errors are manifold. Accordingly, debugging is a challenge for system integrators as well as programmers.

In a field test [Dej05] with about 100 students implementing and setting up a time-triggered real-time system on a distributed 8-bit microcontroller platform, we gathered experience about the many possible sources of malfunctions. We shall briefly outline the major issues.

- hardware failures, e. g., broken cables etc.
- firmware configuration errors, e. g., incorrectly set fuse-bits
- programming errors
- configuration error regarding compiler options as well as linker settings
- inconsistent communication schedules.

Even though, this field test was conducted within an under-graduate course in the bachelor studies of computer engineering, we may assume, that the professional programmers in the industrial field will come across the same concerns, as they deal with the same matter.

While each of those (sub-)problems can be handled, the actual system becomes very complex, indeed.

1.2 Motivation for a tool

We identified the need for an integrated development tool, which assists the designer from the first draft of the application's model to the definition of the global communication schedule as well as the local job specification and the compilation and linking of the resulting source code. The intentions are

1. to accelerate the design and implementation phase of an embedded system's software
2. to reduce the opportunities of errors creeping in during the whole development process

The first goal might especially appeal to the academic research, where it is the attention to have shorter development cycles, so that we can implement several diversitive approaches to a problem fastly and efficiently, when conducting research among the field of embedded systems. Nonetheless, this matter is of interest in the industrial field, too. However, here the second goal is of upmost importance. The less errors we have to remove in source code, the less resources the debugging consumes. Thus, the application of a integrated tool will relieve a project's budget and other resources.

1.3 Requirements

Due to the great variety of embedded system hardware, communication technologies and fields of applications, such a tool has to be highly flexible concerning the configuration and the extensibility in order to be useful in different domains, e. g., for a time-triggered real-time field-bus system like TTP/A [Kop01] as well as CAN [Bos91] and LIN [vdW00].

Preferably, that tool should entail

- an appealing graphical user interface
- expressiveness in meta-modelling and modelling, e. g., in UML-related notation
- a mechanism for formal validation of meta-models
- a validation mechanism for domain-specific constraints imposed by the meta-model

- support for extensibility by means of plug-ins, which consequently gives some degree of flexibility when changing between different plug-ins

In our point of view all those requirements are mandatory for the "ideal" modelling tool. When full-filling those requirements, we would be able *to elevate a generic tool to an integrated development environment (IDE) according to our needs* for distributed time-triggered real-time embedded systems.

1.4 Outline

In chapter 2 we introduce the concept of model integrated computing (MIC), and how it contributes to the software development projects of distributed embedded real-time systems.

Chapter 3 deals with the major part of that work. It evolves the *conceptual model* of TTP/A applications. The contained sections discuss the fundamental design, functional and temporal requirements, as well as the relation and the restrictions between the defined entities.

The following chapter 4 focuses the implementation of that conceptual model called *MetaTTPA* by means of the generic modelling environment GME. Moreover, it explains, how such generic modelling tool can be transformed into an integrated environment (IDE) for TTP/A applications following the concept of MIC. Additionally, it introduces two software components integrated into GME.

Finally, chapter 5 gives a short summary and mentions ideas for future work concerning the field of meta-modelling and modelling of TTP/A applications.

Chapter 2

Model Integrated Computing

2.1 Domain-specific modelling

When we look at the market of modelling tools, we find several prominent products from well-known vendors. For instance, **Rational Rose** (`http://www.rational.com/`) is a visual modelling tool especially for software development projects; **Simulink** (`http://www.mathworks.com`) is a hierarchical block-diagram design and simulation tool with its main application in signal processing; and **LabVIEW** (`http://www.ni.com`) is a graphical programming development environment.

Even though, these tools have been designed for different fields of application and contain other terminology, they share one common property. Each tool is an integrated set of tools for modelling, model analysis, simulation, and code-generation, that helps to design and implement any target system for its own specific, well-definied engineering field.

In other words, these tools *capture specifications of target systems in the form of domain-specific models*. Furthermore, they support the design process by automating analysis and simulating system behavior. In addition, they can automatically generate, configure, and integrate target system components, such as source code, glue code, or database schemas.

Despite their enormous power and feature set, domain-specific modelling tools suffer from one significant block, which detains a broad acceptance: the **high costs**. Consequently, such tools are available only for domains with large markets and high volume, so that the initial investment costs can be balanced.

2.2 Generic notational formalisms

In some sense, modelling among different domains incorporates the same concepts. No matter, if we deal with traditional software engineering or embedded systems, we find concepts like

entities / objects atomic entities or objects, which embody a given property or object of a system.

containment hierarchical relations among entities, which state, that any entity A is part of another entity B.

associations non-hierarchical relations among entities, which model some kinds of real-world relations, e. g., communication.

inheritance a concept imported from the object-oriented programming paradigm; used to express derivation and similarity among different types of entities, or to re-use existing entities.

multiplicity a numerical attribute for containment and association relations, which gives information how many entities may take part in some kind of relation.

attributes each concepts above can be equipped with textual or numerical attributes in order to include additional information.

Consequently, we could take into consideration, that we do not use domain-specific modelling and its tools, but we use a *generic notational formalism* like UML to specify a *rule set* according to the domain for the models of a target system. However, would this be enough?

For instance, if we define in the rule set, that an object of type A is allowed to contain objects of type B (but we have no such rule regarding objects of type C), then a generic notational formalism like UML is able to express this relation by means of the concepts of the containment. In the model of the target system we would not be able to let an object of type A have an object of type C. The same applies to connective rules. Objects must not be connected by a given type of connection, unless explicitly specified in the rule-set.

While we can even express such conditions with "ordinary" UML, what about more complex relations between objects according to a domain-specific point of view? Let's say, we have objects of the type "Task" and connection relations "Phase" and "Dataflow" between "Task"s. But a "Phase" must not be

declared between a pair of "Task"s, if and only if those "Task"s are already connected by a "Dataflow". How could we model this *constraint* in UML? UML does not care about "Task"s, "Phase"s and "Dataflow"s, hence it does not include such a domain-specific point of view, but it provides the standard notation of "Class"es, "Connection"s etc.

In short, the generic notational formalism's capabilities of sufficiently expressing models of domain-specific target systems are limited. The lack of the appropriate terminology of the given field must be bypassed by generic concepts, which impose an unnecessary degree of complexity. Moreover, it is hard (or even unfeasible) to express specific constraints, as this is not within the reach of the generic notation.

2.3 Meta-Modelling

Contrary to domain-specific modelling as well as generic notations, the *model integrated computing (MIC)* deals with *meta-models*. In that case, the modelling formalism is not tied down to any specific domain, but it is so generic, that we can easily produce a *meta-model*, which describes the entities, relations and constraints of the given domain. Figure 2.1 illustrates the functioning of meta-modelling.

As we can see, meta-modelling allows some kind of "boot-strapping" or recursion of models. Firstly, we have one generic model – the *meta-meta-model*, which models the *meta-model* of a given domain. Additionally to the concepts stated in section 2.2, such meta-model includes the familiar *terminology* of a given domain as well as *integrity constraints*, which a concrete domain model has to full-fill. Finally, we use that meta-model in order to specify a concrete target system with a domain-specific point of view.

It must be mentioned, that the meta-meta-model and the meta-model use the same notational language. This is a generic language, which does not involve any domain-specific concepts. But this generic language builds the meta-model, which defines the domain-specific notational language. The model of the target system is just expressed in the domain-specific language formed by the meta-model. Even though, all these languages might use different notation and terminology, in most modelling environments they use a **closely related style**, e. g., UML-like diagrams.

All these steps take place within the same generic integrated modelling tool. Thus, when defining a meta-model for a given domain or engineering field, that generic modelling tool transforms into a modelling tool for that domain.

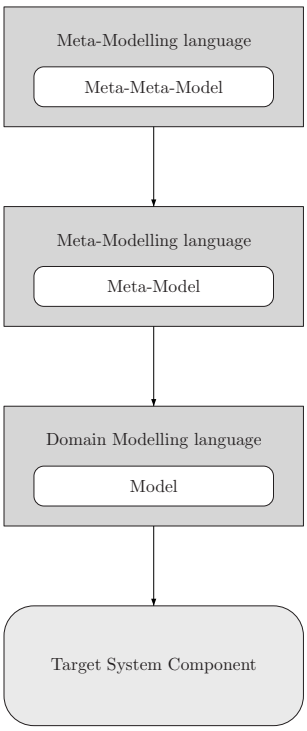

Figure 2.1: Structure of MIC systems

However, the tool can be re-transformed for applications in another field, if we define another meta-model for that field. So, a generic modelling tool is highly flexible and extensible due to the meta-modelling approach of MIC.

2.4 Meta-Modelling for embedded systems

Finally, when we have defined a model for a target system, most modelling tools support a further processing of the model, which produces the software component, i. e., some software part of the overall system, for usage in the target system. Such a component would be a piece of source code, if we dealt with modelling of software engineering projects. Probably, we define the layout of an electronic circuit in the domain-specific model, while the meta-model consists of electronic components, e. g., AND- and OR-Gates etc. Then, the target system component is a Intel-HEX-File, which can be loaded onto an FPGA.

To sum up, a generic modelling tool turns into an *integrated development environment* for a specific domain or engineering field due to the mechanism of meta-modelling in MIC and the further processing.

Consequently, it is obvious, that we can adapt any given generic modelling tool into such an IDE for embedded systems, moreover for distributed embedded real-time systems. All we need to do is to define the meta-model for the design of such embedded systems. Additionally, we would like to extract some useful source code, communication schedules, or other data from the model, so that this product can be used in the overall development process. For that purpose, we would create the *model processor* or *model transformer* or *model parser* and integrate it into the tool.

2.5 Related Work

Recent summits like the *MoDELS/UML 2005 conference on model driven design* [MoD05] in general and the *MARTES 2005 workshop on modeling and analysis of real-time and embedded systems* [MAR05] testify the high interest of industry and academic researchers in the field of modelling. In this section we will shortly examine a few up-to-date promising approaches.

2.5.1 Modeling UML Architectures with Coloured Petri Nets

Pettit and Gomma [Rob05] propose a method for early design analysis, in which hierarchically oriented Coloured Petri Nets (CPN) [Jen97] are used for capturing dynamic aspects of a given object-oriented architecture expressed in UML. Therefore, the UML model is transformed based on appropriate stereotypes and templates into CPN. On the resulting model existing analysis tools can be used. For Petri nets there exist syntax level analysis methods for validating control invariant and deadlock freedom, but the main interest is the usage of powerful simulation based state space and performance analysis provided by the tool named DesignCPN (`http://www.daimi.au.dk/designCPN/`).

Pettit and Gomma argue for the motivation of their work that object-oriented architecture design can be augmented, so that the confidence that the design will cover functional and performance requirements can be increased. Using this technique, object-oriented artifacts by means of the Unified Modeling Language [UML04] are still captured. Furthermore, this native modelling

language is then enhanced by seamlessly integrating an underlying formal representation capable of providing the necessary analytical tools. Therefore, the particular method used in Pettit's and Gomma's research is the integration of CPNs with object-oriented architecture designs captured in terms of UML behavioural models.

The basis of the model is a communication diagram containing a collection of active and passive objects along with the message communication that occurs between the objects. In order to generate a CPN from the UML representation some information must be captured (by means of UML).

Stereotype : Each object must be stereotyped to indicate its fundamental behaviour. The stereotypes also determine the structure of the CPN template used to model the object.

Execution Type : Objects must be declared as either passive or active by means of a tagged value. This distinction is used to determine whether an object maintains its own thread of control within the system. Within the CPN model, each active object is given its own control token to model its thread of control and to represent its concurrent execution within the system.

state chart : State-dependent objects are assigned a UML state chart in order to capture the state behaviour of that object. This information is even used in the CPN representation to simulate state behaviour of the respective object.

Processing and Activation Time : Some information about timing are also including, namely the estimated processing time of an object as well as the period of the activation.

Communication issues : As objects might correspond by means of messages, the UML representation has to capture appropriate information. For instance, this is an I/O Mapping, operation types (reader or writer in the communication), communication type (asynchronous or synchronous) etc.

For each stereotype and tagged value pairs a pre-defined CPN template is specified corresponding to the object's behavioural roles. In the transformation process each UML item is replaced by the corresponding CPN template. Such CPN templates have consistent interfaces and can be inserted into the resulting CPN model of the software architecture and connected in

a component-based fashion. The templates are then customized using the information from the UML specification items.

Once the underlying CPN model has been constructed, test cases are created based on use case scenarios from the UML model. These test cases are used to exercise the CPN model for the purpose of simulating the execution of the architecture and for analyzing the resulting behaviour. Results from the simulation are used to assess the object architecture in terms of desired output and desired performance. For this purpose, the tool DesignCPN (`http://www.daimi.au.dk/designCPN/`) is used.

Additionally, someone could conduct performance analysis with respect to timing and concurrency on the CPN model with DesginCPN's performance tool [LW99]. For instance, we could conduct end-to-end timing analysis of a system with multiple concurrent objects. These results can then be compared with performance requirements to determine if the system in fact satisfies the necessary throughput and timing requirements. By being able to conduct this form of analysis from the concurrent software design and adjust the design accordingly, someone gains confidence that the final design will satisfy the necessary performance requirements of the system.

2.5.2 Pure UML based modelling

Similar to this work, where it's the intention to introduce an integrated development environment for distributed embedded real-time systems based on meta-modelling, Kovacs, Pinter and Majzik propose another platform-specific development tool that can be extended by or interfaced to a *purely UML-based* visual design toolkit [KPM05]. As a case study, they present their approach for time-triggered systems (TTA) [KB03]. On the basis of the design language (respectively the conceptual model) of those systems, meta-model extensions of UML class diagrams are proposed and the necessary transformations are specified and implemented. The authors show that the design flow of time-triggered systems immediately benefits from the UML-based approach, existing methods of automatic code generation from UML activity diagrams can easily be tailored to support the platform-specific operating system TTP-OS and communication layer FT-COM [TTT05] used in TTA applications.

Overview of the Time-Triggered Architecture (TTA)

The Time-Triggered Architecture (TTA) embodies a computing paradigm for the design and implementation of distributed real-time embedded systems. It defines a hardware and software architecture and an appropriate design methodology. As its name suggests, the main characteristic of a TTA system is that communication and task execution are initiated at pre-defined points in time.

From the hardware point of view, a system is decomposed into clusters with individual hosts in each cluster. The hosts are interconnected by either a redundant bus or a redundant star topology. Hosts consist of stand-alone computers that are attached to an independent communication controller (CC).

The communication in a cluster applies the time-triggered protocol TTP/C [TTA03]. The communication controllers decide autonomously based on the Message Description List (MEDL) when a message is transmitted to or received from other hosts.

Software Development with TTA

In the TTA the perspectives of system-level, which deals with integration of the hosts within a cluster, and the level of node development, which only concerns a host locally, are strictly seperated.

- The cluster design tool TTPplan [TTT06b] is used to define the parameters of the cluster, namely the hosts, subsystems, message, message types and cluster modes. Relation among those entities are stated, so that a MEDL can be generated.
- The node design tool TTPbuild [TTT06a] is used to construct the internal behaviour of a host according to the MEDL generated at cluster level. Tasks are identified that make up the subsystems, process incoming and outgoing messages. The configuration for TTP-OS and FT-COM can be extracted. The output of TTPbuild is available as program code skeletons that can be completed by the application specific program code of the tasks.

Figure 2.2 depicts the cooperation of both design tools.

Instead of supporting a visual design, the user interface of both tools consists of a sophisticated set of tables, that can be filled in to supply all information. Therefore, a visual design language by means of UML might be desirable.

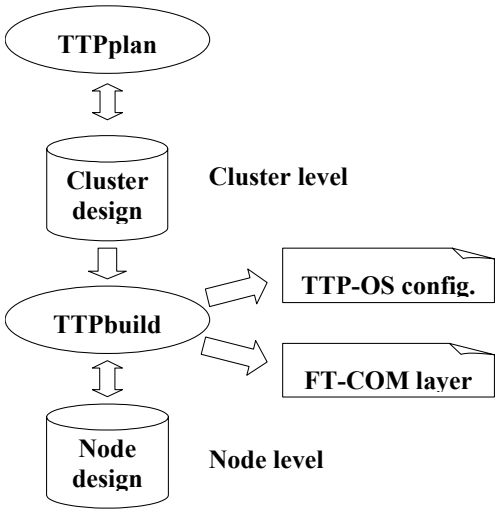

Figure 2.2: The work-flow in the TTA software development environment (TTTech)

UML Profile for TTA

The terminology and the concepts of TTA are reproduced by means of UML notation. The native mechanism of UML in order to extend the meta-model is the concept of the *stereotype*, that provides a facility of defining subclasses of UML meta-classes – a stereotyped model element is considered as instance of a meta-class that does not exist in UML's built-in meta-model.

The new stereotypes together with the necessary constraints form the *UML Profile for TTA*. These constraints force the accordance of a model to TTA-specific conditions and are expressed in OCL [OCL04] – UML's native constraint definition language. Certainly, it's beyond the scope of this thesis to mention all the details of that UML Profile for TTA. At this point we will refer to the original literature [KPM05].

Integration of the tools

In Kovacs' case study, a UML CASE tool – Rational Rose (`http://www.rational.com/`) – was used as the visual modelling tool in order to produce models based on the UML Profile for TTA. The UML CASE tool and the TTA design tools (TTPbuild and TTPplan) are not compatible by default, but they can exchange information by means of the XMI (XML Metadata In-

terchange) [XMI02]. Thus, a transformation is defined and implemented that processes the XMI output of the CASE tool, extracts the relevant information and feeds the model repository of TTPplan and TTPbuild by using their respective programming interfaces. The cooperation of all tools is depicted in figure 2.3.

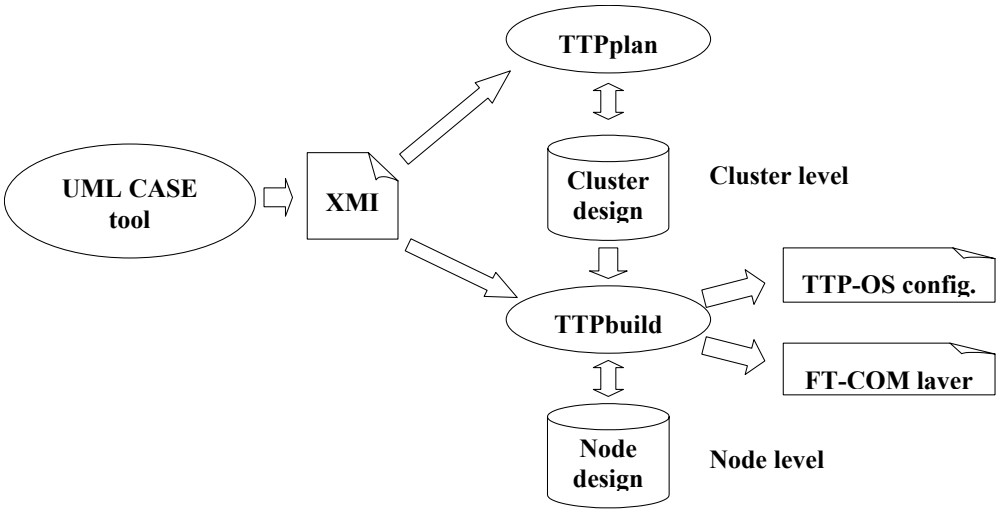

Figure 2.3: Cooperation of visual modelling and TTA software development tools

2.5.3 OMG's efforts

Not only academic researchers consider application development supported by modelling, but even popular organization like the OMG (Object Management Group). The more recent *Model Driven Architecture (MDA)* [MDA05] initiative puts forward the idea, that future software development in general will focus on models, thus keeping application development and underlying platform technology as separate as possible.

The MDA is a collection of paradigms combined with appropriate modelling languages and meta-models. At the core of MDA are the concepts of models, of meta-models defining the abstract languages in which the models are captured, and of transformations that take one or more models and produce one or more other models from them. The means of representation are design languages like the Unified Modeling Language (UML) [UML04] or the Meta-Object Facility (MOF) [MOF02].

In this work we deal with distributed embedded real-time systems. Certainly, the range of MDA is very broad and concerns several fields of software development. However, there exist several UML profiles, which taylor the languages to the needs of the specified domain. Hence, even for the domain of real-time systems the MDA offers an approach for modelling, namely the *UML profile for Schedulability, Performance and Time (UML-SPT)* [SPT05] – or the more recent improvement and enlargement of UML-SPT, the *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms* [QoS04].

2.6 Meta-Modelling Environments

In this section we will shortly examine a few meta-modelling tools as candidates for a model-integrated approach in creating embedded applications.

2.6.1 AToM3

AToM3 means "A Tool for Multi-formalism and Meta-modelling" and is under development at the Modelling Simulation and Design Lab (MSDL) in the School of Computer Science of McGill University [dLV02]. AToM3 is written in Python and is thus available under the General Public License (GPL) for all platforms supporting that language. The main component is the *Processor*, which is responsible for loading, saving, creating, and manipulating models, as well as for code generation. Its meta-meta-model is formulated in Entity Relationship (ER) and allows the modelling of meta-models in a graphical notation. The ER formalism can be extended with constraints, which can be specified using Python functions or (in future releases) OCL.

AToM3 uses Graph Grammar to implement formalism transformation and simulation [BH04], so the users can specify a simulator in a graphical way. Alternatively, it supports the specification of the simulator by means of Python code. AToM3 provides code generation based on the graphical specifications.

2.6.2 MetaEdit+

MetaEdit+ is a commercial Meta-CASE tool developed by the company "MetaCase Consulting", Finland (`http://www.metacase.com`).

Working with MetaEdit+ involves two parts:

- specifying the meta-model using *Method Workbench* — in their definition, the *method* takes the role of the meta-model; generally, it is more complicated than a meta-model formalism and may include additional features like code generators.

- designing the actual model with the CASE tool of MetaEdit+, which follows the previously defined *method* (meta-model).

Even the meta-meta-modelling language in MetaEdit+ named GOPRR (Graph, Object, Property, Relationship and Role) is proprietary. The name implies the main entities in the meta-meta-modelling language, namely graphs, objects, relationships, roles. Specification of a *method* invokes separate tools in order to declare and modify these main entities and to set up their interrelations. Code generation on the basis of the actual model following the *method* is also supported by MetaEdit+.

MetaEdit+ can look back on a successful series of applications, such as development of mobile phone software for NOKIA, and development of e-commerce platforms for PECUNET. It is available on all major platforms. However, it is commercial and consequently not desirable for this work.

2.6.3 DOME

The DOME (Domain Modelling Environment) (`http://www.htc.honeywell.com/dome/`) is an open-source tool, although developed and maintained by a commercial company: Honeywell. DOME is implemented in Smalltalk and available on different platforms including Windows and Linux. It consists of a set of tools, each tailored to a specific aspect of meta-modelling.

DOME Tool Specification : This is used in order to construct a new meta-model with the DOME Specification Language – DOME's built in meta-meta-modelling language.

ProtoDOME : This is the tool, where the user edits the actual model based on the meta-model specification.

Projector and Alter : Projector is a data-flow-based graphical language and Alter is its textual counterpart. They provide functionality needed to write complex model transformation, for instance code generation, simulation and test execution.

2.6.4 DiaGen

Contrary to the other tools introduced so far, DiaGen [Dia05] is not a tool by its own, but an open-source Java-based framework in order to rapidly develop a *diagram editor* for a specific domain. The diagrams will take the role of a model. DiaGen consists of two parts:

- A framework of java classes which provides generic functionalities for editing and analyzing diagrams.
- The *Generator*: it produces Java Source code for most of the functionalities according to the specification of the diagram language.

In order to design a new diagram editor, the user follows these procedures:

- Define a formal syntax for the diagram using the specification grammar provided by DiaGen. Generally, the users defines, what components will appear in the diagram and how they will be connected.
- Generating Java source code according to the meta-model specified in the diagram specification language by means of the *Generator*.
- Adopt the generated source code to the needs of the target domain by manually editing the code.
- Compile the source code and let the diagram editor for the given domain be constructed.

The approach taken by DiaGen seems to be innovative, however it lacks of other features compared to other modelling tools, for instance a built-in constraint definition language. Nevertheless, constraints could be implemented directly into the diagram editor by adding the constraint checking function in the Java source code.

2.6.5 GME

We will shortly argue, why GME is the tool suite of choice when it comes to the "optimal" generic modelling tool. All in all, GME suits all the needs listed in section 1.3.

The Generic Modeling Environment (GME) [LaBK+01] was developed at the ISIS group at the University of Vanderbilt. GME supports modelling

and meta-modelling on the basis of a UML-style notation [LBM+01]. The concepts are realized in an open-source tool running on the MS Windows platform and providing a graphical user interface and model editor.

Practically, designing an application with GME involves two steps:

1. A meta-model of the intended **application domain** is created.
2. This (domain-specific) meta-model is used in order to express **application-specific models** of target applications.

At that point we must mention, that GME does not incorporate any other notation than its own to produce domain-specific meta-models as well as application-specific models, which are "instances" of the according meta-model. GME allows some kind of "boot-strapping" or recursion of models – we have one basic GME model, which models GME models, which model other GME models by themselves.

Additionally, GME provides *aspects*. With aspects it is possible to mask specific parts of a model. As a result, we can concentrate on that information, which is only relevant at the moment. We will not be confused by an overloaded workspace. Moreover, aspects are not pre-configured. Conversely, they are defined as part of a meta-model, thus enabling to exactly specifying the masking operation according to the given domain.

Finally, GME is a highly flexible and extensible architecture, because it supports the integration of customized plug-ins or - in its own terminology - *interpreters*.

2.6.6 Comparison

Table 2.1 lists the features of the meta-modelling tools introduced in previous sections.

In the end, we choose GME as the tool to build an integrated development environment for distributed embedded real-time system on basis of the TTP/A communication subsystem.

The most important reason for our decision for GME is its built-in support of a validation mechanism by implementing OCL constraints [OCL04]. With GME we can impose domain-specific constraints on a meta-model. As GME provides built-in support of OCL, there is even no need to use another external OCL constraint checking tool, but we can remain in one environment. In short, GME allows us to write a domain-specific meta-model, define

Aspects	AToM3	MetaEdit+	DOME	GME	DiaGen
Platform	Windows Unix	Windows Unix Solaris	Windows Linux Solaris	Windows	Platform independent
Implementation language	Python	unknown	SmallTalk	C++	Java
Meta-modelling language	ER	GOPRR	DOME tool specification language	UML	own specification language
Graphical specification	yes	no	partly	yes	no
Hierarchy	partly	decomposition	subdiagram	yes	no
Inheritance	no	yes	yes	yes	yes
Constraint Language	Python or OCL	none	Alter language	OCL	none
Simulation	yes	yes	yes	yes	yes
Simulation method	Graph Grammar	Report definition language	Alter function	C++ plug-in	Java program
Report generation	no	yes	no	no	no

Table 2.1: Summary of the meta-modelling tool's features

domain-specific constraints, and eventually we produce application-specific models from the meta-model and let GME ensure the compliance with those constraints.

Moreover, the extensibility mechanism is of interest, too. In chapter 4 we will understand, why this is so important, and how it is used to transform GME into an integrated development environment (IDE) for TTP/A application development.

Chapter 3

The conceptual model

This section covers the conceptual model, which is the theoretical basis of TTP/A applications. It defines a terminology and "design patterns", how a TTP/A application on distributed embedded real-time systems should be modelled. These rules have been transformed into a GME model, which serves as meta-model for TTP/A application models – *MetaTTPA*.

The conceptual model has undergone several revisions ([EPS04] and [Pau04]). With the usage of GME it has reached a new level of maturity, and finally it encapsulates all special features supported by the TTP/A communication protocol [Kop01].

First, we will give an overview of the fieldbus communication system TTP/A, which is the underlaying communication protocol of TTP/A applications. Further information can be found in the original documentation [Kop01].

3.1 Overview of TTP/A

TTP/A is a time-triggered fieldbus protocol, which meets the given timing requirements due to its real-time character. TTP/A is standardized as part of the Object Management Group Smart Transducer Interface Standard [OMG02].

3.1.1 Communication System

Generally, each communication partner – often called *host* or *node* – on the communication system allocates the bus **a-priori** following a *static communication schedule*. The time line is divided into slots according to the Time Division Multiple Access (TDMA) paradigm. During each slot one communication partner can have assigned the bus exclusively. Such slot is the unit for transmission of one byte of data. Figure 3.1 depicts the sequence of some slots, which make up a *round*, in that case a multipartner round (see below).

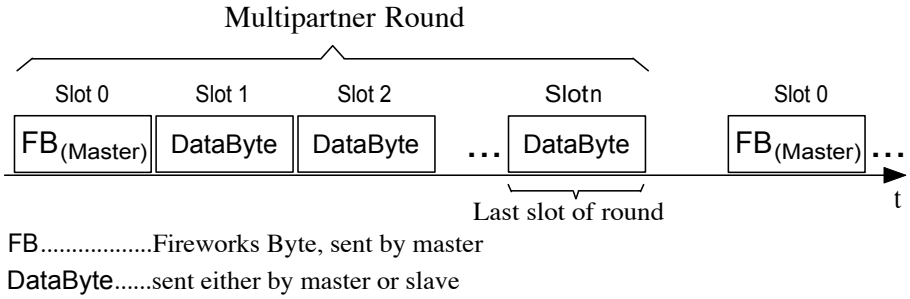

Figure 3.1: TTP/A Multi-Parnter Round

The beginning of a round is indicated by the *fireworks byte*, which is sent by the **master** on the bus. Obviously, TTP/A is a *master/slave architecture*. The collection of at least one master and several slaves build a *TTP/A cluster*.

The fireworks byte determines the type of the round and is a reference signal for clock synchronization. The protocol provides 8 different firework bytes encoded in a single byte message using a redundant bit code for error detection. Generally, there are 2 types of rounds.

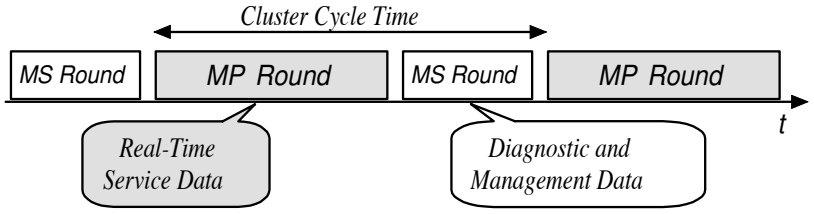

Figure 3.2: TTP/A Round Sequence

multipartner round Such round is made up of a a-priori given number of consecutive slots, whereas the bus arbitration is already arranged. This configuration is defined in a data structure called *Round Descriptor List (RODL)*, which is hosted on every communication partner from its own point of view. The RODLs determine which node transmits and which nodes receive during a certain slot, or which service is to be executed. Figure 3.1 illustrates a multipartner round.

master/slave rounds These rounds serve in order to access specific data from any slave in the cluster. The layout of those rounds is fixed. Firstly, during the *master/slave address round* the master addresses one slave and determines a specific chunk in the slave's memory, which should be read (sent back to the master) or written (stored at the chunk). Secondly, that slave answers in the following *master/slave data round* with the appropriate data or acknowledge.

Figure 3.2 shows an example of a sequence of different TTP/A rounds. Moreover, this specific sequence is the recommended schedule for TTP/A applications. That schedule – called *Rounds Sequence (ROSE)* – is only present in the master, as the master is responsible for announcing the type of round with the fireworks byte.

3.1.2 Interface File System

In TTP/A, a node's memory area is logically structured, either completely or parts of it. This structure on every node is called the local *Interface File System (IFS)*. Consequently, a TTP/A cluster's IFS is made up of the IFS of all nodes in it. The IFS is the source and destination of the communication data conveyed over the bus. All relevant application-specific data consumed or produced by a node are mapped into its IFS.

The IFS of a single node may contain up to 64 files with a maximum of 256 records of 4 byte size for each file. The current revision of TTP/A reserves 2 files in slaves, 4 files in the master for special purpose. Moreover, the RODL of each host and the ROSE are IFS files themselves.

The layout of a IFS file is statically defined, but all files can have arbitrary lengths. However, the first record in every file is the *header record*, which includes protocol specific data.

3.2 Decomposition

In TTP/A, a real-time application is decomposed into a **set of jobs** $\mathcal{S}$. Each job s is a unit of distribution, and is made up of one or more *tasks*, thus a job $s = \{s'_1, s'_2, \ldots\} \in \mathcal{S}$ is a *representative* for its set of consisting tasks, and should be understood as single entity.

The set of distributed *nodes* $\mathcal{H}$ represents the hosts in the TTP/A cluster hosting the jobs. Each job is made up of one or more executing tasks. However, a job with its physically executing tasks must only be run on exactly one node. Note, that a job is not physically present, it is just an abstraction of a set of tasks.

Usually, in TTP/A we find a 1-to-1 mapping between jobs and tasks; in other words: a job is (in most cases) only implemented by one single task. The hosts are connected via a real-time communication system running TTP/A.

Formally, we denote a job $s \in \mathcal{S}$ hosted on host $h \in \mathcal{H}$ as $s \multimap h$. Obviously, as a job s is made up of several tasks s'_i, all those tasks are hosted on the same host h: $s \multimap h \Leftrightarrow \forall s'_i \in s,\ s'_i \multimap h$. Please keep in mind, that the term "job" is an abstraction of a set of tasks. *A job is not physically executed by a host's processors, but its consisting tasks are.*

Furthermore, we name the decomposition for a whole TTP/A cluster $\mathcal{Q} \subseteq \mathcal{S} \times \mathcal{H}$ and the jobs hosted on one specific host h as $\mathcal{Q}(h)$. Then, we define the decomposition more formally as

$$\mathcal{Q}(h) = \{s = \{s'_1, s'_2, \ldots\} \in \mathcal{S} \mid s \multimap h\} \qquad \mathcal{Q} = \bigcup_{h \in \mathcal{H}} \mathcal{Q}(h)$$
$$\text{with } [\bigcap_{h \in \mathcal{H}} \mathcal{Q}(h) = \emptyset] \wedge [\forall h \in \mathcal{H} \mid \mathcal{Q}(h) \neq \emptyset] \tag{3.1}$$

Please note, that in our current model a host must execute at least one job. "Empty nodes", which are present in the cluster, though, but not executing any jobs and reserved for later usage are not supported in the conceptual model, yet. This restriction is arbitrary, indeed. However, it leads to the following condition: because a node must host at least one job, the number of hosts is less equal the number of jobs

$$\forall h \in \mathcal{H} \mid \mathcal{Q}(h) \neq \emptyset \Rightarrow |\mathcal{H}| \leq |\mathcal{S}| \tag{3.2}$$

Every host is equipped with an Interface File System (IFS). The host's communication subsystem sends data from the IFS through the communication

network interface (CNI) over the communication system, while other host's communication subsystems are listening and place the received data in their IFS. These activities take place at pre-defined points of time, which are defined in the Round Description List (RODL).

The communication subsystem with its temporally controlled behavior operates independently from the overlaying application software, thus forming a temporal firewall [Kop97]. The application software consisting of the job's executing tasks uses the Interface File System to obtain and distribute application-specific data, i. e., to communicate with other tasks, while disregarding communication issues of the underlaying communication system.

3.3 Interfaces

From the point of view of the tasks, each job defines *interfaces* to its host's IFS in order to access application-specific data, and all tasks of that job contribute to the interfaces. We will call the entities *ports*, which an interface is made up of. We distinguish the following interfaces (see fig. 3.3):

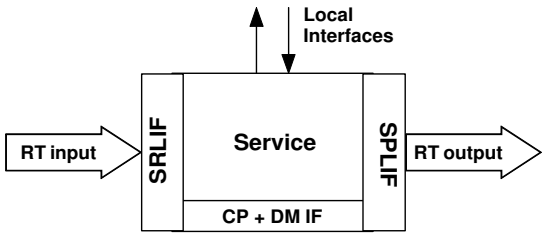

Figure 3.3: Interfaces of a job (from [EPS04])

Service Providing Linking Interface (SPLIF): This interface provides the real-time services to other jobs [JKK+02].

Service Requesting Linking Interface (SRLIF): A job that requires real-time input requests these data via the SRLIF [JKK+02].

Diagnostic and Management (DM): This interface is used to set parameters and to retrieve information about intermediate and debugging data, e. g., for the purpose of fault diagnosis. Access of the DM interface does not change the (a-priori specified) timing behavior of the service.

Configuration and Planning (CP): This interface is used during the integration phase to generate the "glue" between the nearly autonomous services, e. g., communication schedules. The CP interface is not time critical.

Local interfaces: The term local interfaces subsumes all kinds of devices, such as sensors, actuators, displays, and input devices, for which the job creates a unified access via the SPLIF or SRLIF services. For example, the job may instrument a physical sensor element by reading the measurement, calibrating the value, and ex- porting the measurement via its SPLIF.

Concerning the TTP/A protocol, the DM and CP are implemented in the master/slave rounds. Complementary, the multipartner rounds form the SPLIF and SRLIF, which together are also called *real-time service (RS)*. Local interfaces are created by the local job's tasks, which handle the physical sensor, actuators etc.

3.4 IFS Management

The ports of all interfaces are mapped into the Interface File System, thus forming an own Interface File for each task, the so called *task file*. At this point it must be mentioned, that task files belong to their appropriate tasks, but not to jobs as a whole.

Additionally, each host possesses one outstanding file, where all ports of task files are mapped – the *I/O file* [EHK+02]. The host's CNI performs sending and receiving operations only from and to this special I/O file.

Through the mapping each task gathers the data **indirectly** from the I/O file, but not from his own task file directly. As a result, we have one central chunk in the IFS, where all communication from all tasks goes in and out. The ports in the task files are just references to the "real" ports in the I/O file. We can think of some kind of *2-tier access*. Nonetheless, the access to the ports in the I/O file is transparent for the local task, as the mapping is realized through references.

Additionally, we can conduct communication between tasks hosted on the same node without using the real-time communication system (see fig. 3.4). Due to the 2-tier access to all real-time data, two (and even more) tasks (no matter, whether they belong to the same job or not) hosted on the same node can direct their local reference to the same port in the I/O file. As a

result, this implements some kind of *shared memory* between the involved tasks, though, due to the transparency of the 2-tier access each tasks is not aware of that fact.

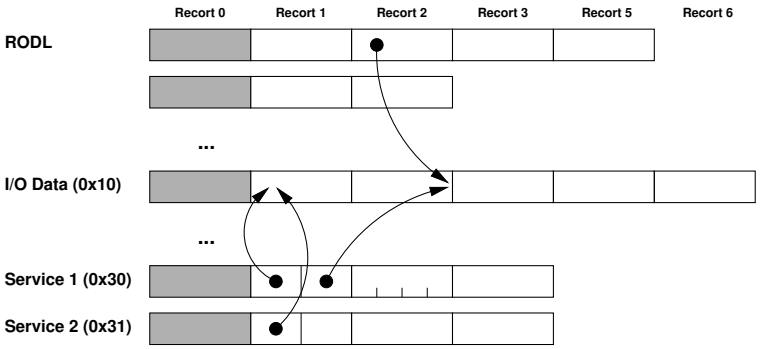

Figure 3.4: I/O file referenced by task files

3.5 Functional and Temporal Requirements

The *specification* of a distributed embedded real-time system does not only involve the description of its functional "behavior". Temporal requirements like service execution deadlines, periodicity have to be considered, too [Kop97].

That distinction was a major factor in the design of **MetaTTPA**, thus it found its manifestation in the conceptual model.

3.5.1 Scopes

As mentioned before, the conceptual model defines *functional* and *temporal* requirements, each TTP/A application has to full-fill. Those requirements appear in different scopes of observation within a TTP/A application.

job level Such functional and temporal requirements only concern jobs for themselves.

application level Beyond the scope of jobs we define the application level. On the one hand, here we find requirements, which reveal their utility only on global scope, but not on any single job. On the other

hand "application level" follows from the *composition* (cf. "composability" [Kop97]) of job level requirements.

Table 3.1 gives a summarisation of functional and temporal requirements among job and application level.

level	**functional**	**temporal**
job	local algorithms	job execution deadline
application	dataflow among jobs	deadlines between jobs phases between jobs

Table 3.1: Functional and temporal requirements in job level and application level scope.

3.5.2 Job Level

Functionality of jobs

By now, the conceptual model of TTP/A handles jobs as **black boxes**, except the fact that they are made up of several tasks. Unlike [KPM05], it does not concern the single jobs respectively their tasks, yet. The local algorithms executed by the job's tasks are neither subject of the meta-model nor the application-specific model. There is no modelling formalism for the interiors of tasks present in the conceptual model, so far.

The reason is, that the functionality of a job is hidden behind its interfaces [Kop97]. Consequently, we do not need to care about **what** (algorithm) a job is doing, but **when** (temporal) it provides respectively requires **which kind** (functional) of data.

Deadlines with jobs

Concerning the job level scope it is only necessary to specify, **when** a job is to be finished with its execution, i. e., the execution of all its consisting tasks – this is the *job execution deadline*. Obviously, a job will have an "ordinary" *execution time*, which says how long the job, i. e., all its tasks, actually takes to execute, whereas the deadline specifies the instant, when the job is to be finished. In order to full-fill the deadline, the job has to finish before the deadline, hence the job execution time must be lower than the deadline.

Generally, in the conceptual model a *deadline* may assume two different types.

periodic A periodic deadline d of value $\upsilon_s(d)$ for a job s means, that s has to be finished with the execution at most **every** $\upsilon_s(d)$ time units. So, this introduces some sense of periodicity, whereas it does not reference the beginning of a period. Usually, the period incorporates the duration between two consecutive multipartner rounds with the same fireworks byte code. Thus, the equation of a periodic deadline is defined as

Requirement 1 (periodic job deadlines)

$$\Delta(MP_c, MP_c') + T_c(s) \leq \upsilon_s(d)$$

whereas $\Delta(MP_c, MP_c')$ means the duration between two consecutive multipartner rounds with fireworks byte code c, $T_c(s)$ is the instant when job s has finished relatively to the beginning of the actual multipartner round of type c, and $\upsilon_s(d)$ marks the value of the deadline d of job s.

offset Contrary to periodic deadlines, an offset deadline d determines the time $\upsilon_s(d)$, that a job s has to be finished, taking the beginning of one multipartner round c as the reference. We describe such a requirement as

Requirement 2 (offset job deadlines)

$$T_c(s) \leq \upsilon_s(d)$$

with the symbols having the same meaning as above.

3.5.3 Application Level

Deadlines on Application Level

Nonetheless, deadlines occur on application level scope, too. On global application level scope a deadline d *between a pair of jobs* (s, r), whereas s sends data to r, specifies the critical instant of time $\upsilon_{(s,r)}(d)$, until s has completely sent all data to r – from the other point of view, r has received all data from s. Certainly, such deadlines can assume the same kinds – *periodic* and *offset* like their job level counterparts with appropriate meanings (see sec. 3.5.2).

With the familiar symbols from the previous section we write the equations for periodic (req. 3) and offset (req. 4) deadlines in application-level scope. The symbol $\triangleright(s, r)$ embodies the transmission activity from any service s to service r.

Requirement 3 (periodic transmission deadlines)

$$\Delta(MP_c, MP'_c) + T_c(\triangleright(s, r)) \leq \upsilon_{(s,r)}(d)$$

Requirement 4 (offset transmission deadlines)

$$T_c(\triangleright(s, r)) \leq \upsilon_{(s,r)}(d)$$

Phases

Another temporal requirement in application level scope are *phases*.

A phase p between a pair of jobs (a, b) defines a time span, that the start of execution of both tasks has to deviate approximately, no matter which job is started first – so, phases are non-directional / non-hierachical relations. Due to the TDMA schema of TTP/A, we can not achieve a fine-grained resolution of the difference of starting points. The resolution is dependent on the slot length, which in turn is dependent on the transmission speed, because starting points can only be assigned to the beginning of a whole slot. Therefore, we can only express phases "approximately" (in slot duration quanta). In the following we will define the concept of phases in formal way.

We name the starting point of a job $S_c(x)$ of any job x in the multipartner round c. As a phase p is defined as the duration between two job's starting points, we express a phase p between two jobs a and b in a given schedule as $\Delta_p(S_c(a), S_c(b))$ or in a more symbolic way $a||_p b$, concerning the actual value of that phase within a given schedule $\upsilon(a||_p b)$. With the specified time value $\phi(p)$ of a phase p, whereas ϕ has the context of the temporal requirement like deadlines imposed by the target application's specification (not the actual schedule), we can set up an equation of full-filling a phase relation. Concerning starting points this is

$$\Delta_p(S_c(a), S_c(b)) \approxeq \phi(p) \tag{3.3}$$

and with regard to the more symbolic way of notation we have

$$\upsilon_c(a||_p b) \approxeq \phi(p) \tag{3.4}$$

Unlike earlier revisions of the conceptual model ([EPS04] and [Pau04]), now the *transitivity* among jobs in phase does not apply any more (by default).

For instance, given a set of three (and even more) jobs $\{x, y, z, \ldots\}$, transitivity according to any phase p says that

$$x||_p y \wedge y||_p z \Rightarrow x||_p z \tag{3.5}$$

If we consider a value for the phase p, we will discover, that the rule of transitivity leads into a contradiction. Let's say, that $\phi(p) = 1$, and we assume any combination of starting points $\{S_c(x) = 0, S_c(y) = 1, S_c(z) = 2\}$. Then the phases according to that schedule evaluate with respect to equation 3.3

$$\left.\begin{array}{l|l} \Delta_p(S_c(x), S_c(y)) = 1 & x||_p y \\ \Delta_p(S_c(y), S_c(z)) = 1 & y||_p z \\ \Delta_p(S_c(x), S_c(z)) = 2 & x \nparallel_p z \end{array}\right\} x||_p y \wedge y||_p z \nRightarrow x||_p z$$

As we can see, the rule of transitivity in equation 3.5 does not hold any more, because $x \nparallel_p z$. Maybe we could try to avoid this effect, when we let service c start earlier than before. Let's say, $\{S_c(x) = 0, S_c(y) = 1, S_c(z) = 0\}$. Then we have the following situation

$$\left.\begin{array}{l|l} \Delta_p(S_c(x), S_c(y)) = 1 & x||_p y \\ \Delta_p(S_c(y), S_c(z)) = 1 & y||_p z \\ \Delta_p(S_c(x), S_c(z)) = 0 & x \nparallel_p z \end{array}\right\} x||_p y \wedge y||_p z \nRightarrow x||_p z$$

Again, we can not satisfy the rule of transitivity. No matter what combination of starting points we choose in our schedule for all jobs, the **transitivity among jobs in phase does not hold** for more than two jobs.

If we want to provide transitivity for phases, we have to weaken the requirement from equation 3.3 resp. 3.4, so that a phase can still hold, even though the starting points of all involved jobs (the *transitive closure* of that phase) are not that ideally distributed. For that purpose, we re-define the requirement of a phase.

Requirement 5 (phases between tasks)

$$\Delta_p(S_c(a), S_c(b)) = \phi(p) + \delta_{(a,b)} \qquad with \ -\phi(p) \leq \delta_{(a,b)} \leq \phi(p)$$

As can be seen, we substitute the "approximately" by "equal", but we introduce a variation in the specified phase value by transforming $\phi(p) \rightarrow \phi(p) + \delta_{(a,b)}$. Thus, in the worst case the difference between the starting

points of a pair of jobs might either vanish to 0, or it might take the double amount than originally intended. For instance, with that modification the transitivity rule defined in equation 3.5 will hold for our previous examples with $\{S_c(x) = 0, S_c(y) = 1, S_c(z) = 2\}$ and $\{S_c(x) = 0, S_c(y) = 1, S_c(z) = 0\}$. It must be mentioned, that δ is depended on a pair of specific jobs (a, b), but not on the phase in general. In other words, δ can be different for every pair of jobs. Otherwise, the modification would have been abolished.

Thus, with that modification of the phase requirement we create more possible combinations, when assigning the starting points in the schedule for a given set of jobs in phase. So far, the problem of transitivity of phases has not been solved "properly" in the conceptual model, yet. We just introduced that little work-around in order to force the full-filling of the transitivity among jobs in phase.

Dataflow

Finally, *dataflow* is the major concern in TTP/A application modelling. The dataflow gives information, **how application-specific data passes along the jobs respectively their executing tasks**. Consequently, this implies a *causality* or *precedence*, even *ordering* between jobs. Additionally, dataflow will manifest in the communication schedule, as the carried information of the dataflow will be conveyed over the real-time communication system during some specific TTP/A slots.

A dataflow relation $f_{(\mathbf{src},\mathbf{dst})}$ between a pair of jobs is a *directed* connection. This says, that the originating job, i. e., one specific task of that job, sends data to the sinking task of some job. For instance, if we let service a send to b, we write $f_{(a,b)}$, whereas a takes the role of the sender **src** and b embodies the receiver **dst**.

However, concerning the interfaces (sec. 3.3) of job, a job obtains its real-time data through the SRLIF respectively distributes its data via the SPLIF. Additionally, interfaces are made up of ports. So, if we say, that a job sends to another one, this implies that the data passes a port of the sender's SPLIF and enters an opposing port at the SRLIF of the receiver.

Consequently, the definition above, which simply says that a job sends to another one, is not precise enough, because ports and interfaces are involved. Nevertheless, we will accept that imprecise formulation in order to reduce complexity in the notation.

In the current revision of the conceptual model, dataflow features two directions.

forward feed All dataflows with the direction "forward feed" contribute to the application's *main stream*. Forward feed is the type of dataflow, which appears most. Such dataflow relations build a precedence or ordering among jobs, and therefore contribute to automatic scheduling generation with TTP/A scheduling algorithms.

A job s can not be executed, until it has received all **incoming forward feed** $N^+(s)$ from opposing, sending jobs.

Requirement 6 (ordering of main stream (incoming))

$$\forall f \in N^+(s) : \; T_c(f) < S_c(s)$$

Consequently, forward feed can be time-critical. A deadline between to jobs (a, b) is only reasonable, if there also exists forward feed dataflow between (a, b).

After execution, job s sends its **outgoing forward feed** $N^-(s)$, whereas each dataflow in $N^-(s)$ is part of the incoming forward feed of some other receiving jobs R_s.

Requirement 7 (ordering of main stream (outgoing))

$$\forall [f_{(s,r)} \in N^-(s)] \subseteq N^+(r), \; r \in R_s : \; T_c(s) < S_c(f_{(s,r)})$$

backward feed Typically backward feed occurs relatively sparsely. It has been introduced in order to solve the problem of *cycle resolution*, when it comes to automatic scheduling using the TTP/A specific scheduling algorithms.

The most likely usage for a backward feed is a dataflow between two jobs, so that the backward dataflow embodies the **feedback of a control loop**. Backward feed is not relevant to automatic scheduling generation, though, it must be included in the schedule. It is not time-critical.

We will denote backward feed dataflow like $f^B_{(s,r)}$, with (s, r) being a pair of jobs involved in that dataflow.

It must be mentioned, that incoming and outgoing feed N^+, N^- are made up of dataflow relations, which originate and end in ports. So, we denote $\pi \in N^+(s)$ respectively $\pi \in N^-(s)$ for a port π of a job s, which is part of the SRLIF respectively SPLIF of that job. Alternatively, we could describe

a dataflow $f_{a,b}$ as the pair of ports (π_a, π_b) of the involved jobs a and b, whereas $\pi_a \in N^-(a)$ and $\pi_b \in N^+(b)$. According to the term of the "degree of a vertex" in graph theory, we define the number of incoming dataflow for one specific port $d^+(\pi)$, respectively $d^-(\pi)$ for the number of outgoing dataflow of the port π. To sum up, we could even express the number of incoming dataflow feed of a given job s like

$$|N^+(s)| = \sum_{\pi \in s} d^+(\pi) \tag{3.6}$$

Concerning the number of outgoing dataflow feed we might write

$$|N^-(s)| = \sum_{\pi \in s} d^-(\pi) \tag{3.7}$$

Moreover, the TTP/A protocol supports even more features like *speed-up* and *multiplexed virtual channels*, which have been integrated in the concept of dataflow in the conceptual model.

With speed-ups we can increase the throughput of data conveyed over the real-time communication system per TTP/A slot. A speed-up is a factor, that the transmission speed will be multiplied for the assigned slots occupied by a given dataflow. TTP/A provides several factors: $\{4, 8, 16, 32\}$ times the original transmission speed of the communication system. Actually, speed-ups have no relevance for the conceptual models, as it is handled by the TTP/A protocol stack. Consequently, the speed-up factor is simply an attribute of the dataflow. Dataflow with speed-up does not behave differently towards the conceptual model, than dataflow without (factor = 1).

Finally, multiplexed virtual channels entail an additional degree of complexity to the conceptual model, even though rarely used. Multiplexing provides *slots in slots*, furthermore it can be combined with speed-ups.
Given a specific TTP/A slot, this slot can be used differently in consecutive multiparnter rounds. Currently, TTP/A enables a periodicity λ of 4 or 8 for a multiplexed slot. For instance, if we have a multiplexing period of 4, that slot is available for 4 different configurations. Maybe in the first iteration service s' sends to r', in the second iteration s'' occupies the slot with a sending operation and r''_1 and r''_2 read the bus, etc. Regarding the communication schedule, the overall structure of bus utilization remains the same. However, that outstanding slot may be used differently in every iteration of the multipartner round, which virtually entails some sort of "sub-schedule" for that specific slot.
From the point of view of the conceptual model, multiplexed virtual channels

$\mathcal{S}$	set of jobs
$\mathcal{F}^F$	set of forward feed dataflow
$\mathcal{F}^B$	set of backward feed dataflow
$\mathcal{F}^M$	set of multiplexed dataflow
$\mathcal{F}$	$\mathcal{F}^F \cup \mathcal{F}^B \cup \mathcal{F}^M$
$\mathcal{D}$	set of deadlines (periodic and offset)
$\mathcal{P}$	set of phases
V	set of vertices in the precedence graph
E	set of directed vertices in the precedence graph
A	set of undirected vertices in the precedence graph

Table 3.2: sets of dependencies

incorporate a specific type of dataflow. All in all, they behave the same way as ordinary dataflow, however multiplexed virtual channels are not time-critical. They are defined like dataflow between a pair of jobs, but they include additional attributes for the periodicity and impose a few additional constraints on the conceptual model (see sec. 3.6.2). Another difference appears with automatic scheduling generation, because several dataflow relationships have to be compressed into one slot, but this is not the matter of the conceptual model.

3.6 Interoperation of the entities

The previous section listed all types of entities, which occur in the conceptual model. Now, we want to examine their interoperation within TTP/A applications.

3.6.1 Precedence Graph

We will introduce a graphical representation of TTP/A applications based on the conceptual model. For that purpose, we introduce some terminology from *graph theory* in the modelling.

A TTP/A application can be represented by means of graphs – the so called *precedence graphs* [Pau04]. Generally, in precedence graphs a job is represented by a *vertex*, and an *edge* embodies a specific relationship between the two adjacent vertices / jobs. In table 3.2 we name sets, which include the entities of the conceptual model and the precedence graph in each application

model.

So, we denote the transformation from conceptual model to precedence graph in a symbolic way.

$$\begin{aligned} V &\vDash \mathcal{S} \\ E &\vDash \mathcal{F} \cup \mathcal{D} \\ A &\vDash \mathcal{P} \end{aligned} \tag{3.8}$$

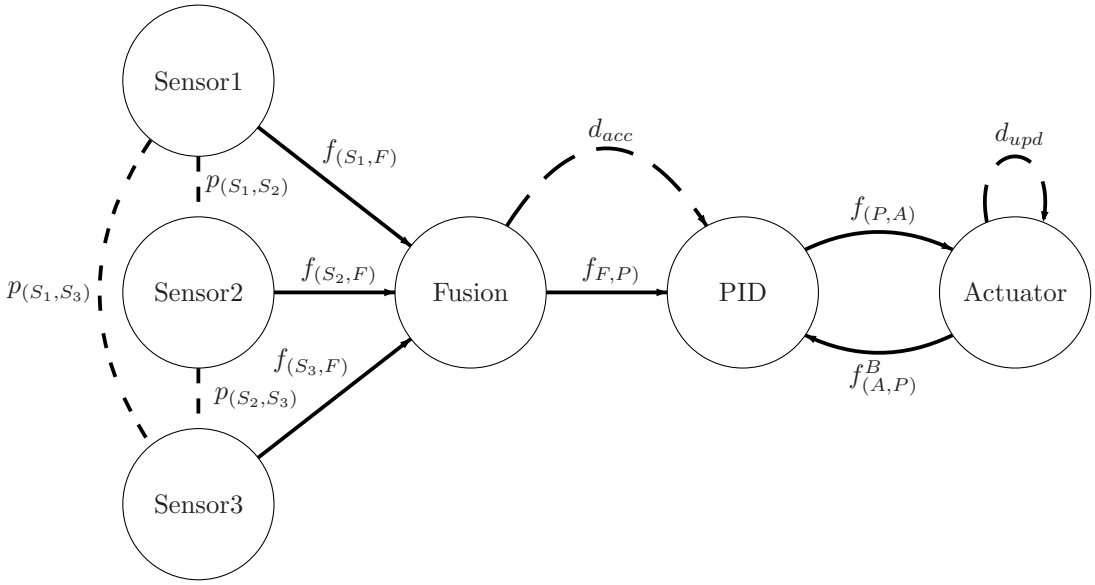

Figure 3.5: Illustration of Smart Fusion in the conceptual model

At this point we introduce an example target application for better illustration. Figure 3.5 shows a precedence graph of a **Smart Fusion** [Elm02] application with control elements.

As we can see from figure 3.5, this application runs 6 jobs, and these jobs are related by dataflow, deadline and phase dependencies.

The jobs $Sensor\{1,2,3\} \in \mathcal{S}$ read some sensor value from their local interfaces and afterwards ship that data over the communication system to the job $Fusion \in \mathcal{S}$ (in different time slots, not at the same time), which is modelled by the three dataflows $f_{(S_{\{1,2,3\},F})} \in \mathcal{F}^F$ from the "sensor" jobs to the

"fusion" job. Thus, these three jobs start the *main stream* of forward feed dataflows. Additionally, the three sensor jobs are "in phase", depicted by the undirected edges between the job vertices. When we introduced the weaker requirement of transitivity (req. 5), we enabled distinct transitivity between more than two jobs. As we have phases between ($Sensor1$, $Sensor2$) and ($Sensor2$, $Sensor3$), this implies a phase between ($Sensor1$, $Sensor3$) according to the rule of transitivity in equation 3.5. Therefore, $p_{(S1,S3)} \in \mathcal{P}$ need not to be included in the precedence graph in order to reduce complexity, which is depicted by the slashed edge.

Afterwards, the "fusion" job continuous the main stream with the dataflow $f_{(F,P)} \in \mathcal{F}^F$ to "PID", which in turn goes ahead with the dataflow $f_{(P,A)} \in \mathcal{F}^F$. Eventually, the main stream ends in the job "Actuator". Moreover, we find one backward feed dataflow $f^B_{(A,P)} \in \mathcal{F}^B$ between the "PID" and "Actuator" jobs. Consequently, that application contains a control loop, and $f^B_{(A,P)}$ embodies the feed back from the actuator job "Actuator" to the PID controller job "PID".

The remaining two dependencies are the deadlines $d_{acc} \in \mathcal{D}$ and $d_{upd} \in \mathcal{D}$. The first expresses, that the dataflow $f_{(F,P)}$ has to be transmitted from "Fusion" to "PID" **within** $\upsilon_{(F,P)}(d_{acc})$ time units, taking the beginning of the multipartner round as reference – in other words, this is an offset deadline. On the other hand, d_{upd} is a periodic deadline involving only one job, meaning that the service "Actuator" has to be executed **every** $\upsilon_A(d_{upd})$ time units.

3.6.2 Restrictions

There exist several restrictions or so called *constraints*, which assure the *integrity* of the modelled target application. Integrity means, that the application expressed by means of jobs and dependency relations (for instance with a precedence graph) is actually feasible in a real target system, and can further be transformed into some reasonable target system component like source code or the communication schedule according to the philosophy of MIC (see sec. 2.4).

Notational issues

If we define the integrity rules of dataflows, phases and deadlines among services, we prepare a suitable notation, first.

$$\left.\begin{array}{c} e \in E \vDash f \in \mathcal{F} \\ e \in E \vDash d \in \mathcal{D} \\ u, w \in V \vDash a, b \in \mathcal{S} \end{array}\right\} e = (u, w) \vDash f_{(a,b)}$$

$$f_{(a,b)} = (\pi_a, \pi_b) \text{ with } \pi_a \in N^-(a), \pi_b \in N^+(b)$$

Obviously, we combine the notation of precedence graphs with the conceptual model formalism, due to the coherence between the conceptual model and precedence graph representation (defined in equations 3.8).

Dataflows and Deadlines

Constraint 1

$$u \neq w \Rightarrow \nexists f_{(a,b)} \in \mathcal{F} \text{ with } a = b$$

Constraint 2

$$\exists d_{(a,b)} \in \mathcal{D} \wedge a \neq b \Rightarrow \exists f_{(a,b)} \in ((\mathcal{F} \cup \mathcal{F}^B) \text{ with } a \neq b$$

First of all, in constraint 1 we determine, that there can never be dataflow of any type (forward / backward feed, multiplexed) from one single job to itself. In terms of graph terminology we can say, that a precedence graph can not have *loops* made of "dataflow edges" attached to its vertices.

However, that need not apply to deadlines. As we learn from figure 3.5, there may exist deadlines originating and ending in the same job like d_{upd}. This circumstance is even expressed in constraint 2. Furthermore, it says that whenever we find a deadline between two different jobs, there must also be at least one dataflow of the forward or backward feed type between that two jobs. Otherwise, as a 2-job deadline concerns the communication between them, and without communication that deadline would be useless. Multiplexed dataflow does not contribute, because multiplexed virtual channels are not time-critical in gernal.

Ports

Constraint 3

$$\exists f_{(a,b)} \mid a \neq b \Rightarrow [(d^-(\pi_a) > 0) \wedge (d^+(\pi_b) = 1)]$$

Concerning the utilization of ports, which are the terminating points of every dataflow and part of a job's interfaces, constraint 3 assumes, that a port is either used for output or for input. The general distinction between incoming or outgoing ports is manifested in constraint 4.

Constraint 4

$$\pi \in N^-(s) \otimes \pi \in N^+(s)$$
$$d^-(\pi) > 0 \otimes d^+(\pi) = 1$$

In addition to this it must be mentioned, that a given port π can have exactly one incoming dataflow. Thus, the "incoming degree" of that port (or the number of incoming dataflows) is exactly $d^+(\pi) = 1$.

Contrary to incoming ports, an outgoing port could have several outgoing dataflows: $d^-(\pi) > 0$. Because TTP/A performs a TDMA schema, several hosts could read the data at the same time from the bus, while always one host transmits exclusively. Accordingly, the port of the sending job must allow several dataflow to other services in order to model that multiple reading.

Finally, if there exists a dataflow, then the two attached ports must be belong to different jobs. This coheres with constraint 1, which claims, that a job can not possess dataflow from itself to itself. Even though, we can have several dataflows between two jobs, we can only define one dataflow for each specific pair of ports (constraint 5).

Constraint 5

$$\mid \{f_{(a,b)} \in [\mathcal{F} \cup \mathcal{F}^B] \mid f_{(a,b)} = (\pi_a, \pi_b), \pi_a \in N^-(a), \pi_b \in N^+(b)\} \mid = 1$$

Phases

In the conceptual model there occurs one important constraint (7), which is expressed best with graph terminology of the precedence graph representation (constraint 6).

For that purpose, we need the definition of the *transitive closure* of phases, first.

$$\varepsilon(p) = \{s \in \mathcal{S} \mid p_{(s,s')} \in \mathcal{P}; s, s' \in \mathcal{S}\} \tag{3.9}$$

The transitive closure $\varepsilon(p)$ of a phase p includes all jobs, so that each job s takes part in the same phase relationship p. As we can see from figure 3.5,

we can optionally model redundant phases like $p_{(S1,S3)}$, which follow from the transitivity of $p_{(S1,S2)}$ and $p_{(S2,S3)}$. Despite the redundancy imposed by transitivity, the transitive closure is stripped down to single occurrences of jobs involved in the phase. For instance, in the smart fusion application depicted in figure 3.5 the transitive closure of the phase p is the set $\varepsilon(p) = \{Sensor1, Sensor2, Sensor3\}$, even though p might span over all sensor jobs redundantly by $\{p_{(S1,S2)}, p_{(S2,S3)}, p_{(S1,S3)}\}$.

Constraint 6

$$\nexists\{u, w \in V \mid [(u \rightsquigarrow w \in E) \vee (w \rightsquigarrow u \in E)] \wedge u \nleftrightsquigarrow w \in A\}$$

From the point of view of precedence graphs, constraint 6 describes a condition, that avoids a **path** of directed edges (in both directions) between two jobs, if that jobs are also reachable by another path of undirected edges. Regarding the mapping of precedence graph terminology and conceptual model notation (see eq. 3.8), this constraint determines, that there must not be a dataflow feed either directly or indirectly between two jobs a, b, if that jobs belong to the transitive closure of the same phase p: $a \in \varepsilon(p) \wedge b \in \varepsilon(p)$ with $a, b \in \mathcal{S}$ and $p \in \mathcal{P}$. To sum up, we re-write constraint 6 in the context of the conceptual model in constraint 7.

Constraint 7

$$\nexists\{a, b \in \mathcal{S} \mid a, b \in \varepsilon(p) \Rightarrow [f_{(a,b)} \vee f_{(a,s_1)}, f_{(s_2,s_3)}, \ldots, f_{(s_n,b)}] \in \mathcal{F}\}$$
$$\text{with } p \in \mathcal{P},\ s_i \in S,\ i = 1 \ldots n$$

Multiplexed virtual channels

Multiplexed virtual channels are made up of multiplexed dataflows, which behave the same way like their forward and backward feed counterparts. Consequently, the constraints 1 and 2 are valid, too. Moreover, the constraints 3 and 4 have to hold. However, as multiplexed virtual channels allow some kind of "sub-schedule" (see sec. 3.5.3), a multiplexed dataflow belongs to one specific iteration of the multiplexed slot, but a port can be used differently in each iteration. Thus, constraints 3 and 4 have to apply just **per iteration**. For instance, among several consecutive iterations each port involved in the multiplexed virtual channel could be used for sending operation, in the next iteration for receive operation, then idle, then receiving again etc.

We will give a more formal notation of multiplexed virtual channels.

$$\begin{aligned} \gamma[i] &= \{f_{(a,b)}[i] \in \mathcal{F}^M\} \\ \Gamma &= \bigcup_{0 \leq i < \lambda_\Gamma} \gamma[i] \\ \text{with} \bigcap_{0 \leq i < \lambda_\Gamma} \gamma[i] &= \emptyset \end{aligned} \tag{3.10}$$

As we can see from that definitions, a multiplexed virtual channel Γ is made up of the virtual channels $\gamma[i]$ of the specific iteration $0 \leq i < \lambda_\Gamma$. Each virtual channel $\gamma[i]$ contains a set of multiplexed dataflow. Moreover, a multiplexed dataflow must only belong to one virtual channel, therefore, the intersection of all virtual channels is the empty set.

However, there is one constraint, which has to apply to the "sub-schedule" of multiplexed virtual channel. As well as in the "first-order" schedule, the slot in every iteration must be used for sending operations **exclusively** by one job. Otherwise, we would have collisions on the bus, even with multiplexed virtual channels. Consequently, there can only be one sending job with one sending port in every iteration, but several other receiving jobs. We manifest that fact in constraint 8.

Constraint 8

$$\begin{aligned} \forall \gamma[i]\ [\exists s \in \mathcal{S}, \pi_s \in N^-(s) \mid (\forall f \in \mathcal{F}^M) \in \gamma[i] = (\pi_s, \pi_r)] \\ \text{with } r \in R_s, \pi_r \in N^+(r) \end{aligned}$$

3.7 Validity & Feasibility

In this section we deal with the matter of validity of model specifications against the constraints as well as the feasibility of a target application's schedule according to the functional and temporal requirements.

3.7.1 Summary of restrictions and dependencies

For better notation in the following sections, we sum up all integrity constraints and requirements to sets.

We denote the **set of constraints** as $\mathcal{C}$.

$$\mathcal{C} = \{\zeta_i \mid 1 \leq i \leq 8\}$$

This embodies the collection of all the 8 constraints defined along the section 3.6.2, whereas a single constraint is called ζ_i and i is the index.

Accordingly, we introduce the **set of requirements** $\mathcal{R}$, which is the collection of all the 7 functional and temporal requirements explained along section 3.5.

$$\mathcal{R} = \{\rho_i \mid 1 \leq i \leq 7\}$$

Here, ρ_i denotes a single requirement with index i.

3.7.2 ASPEC and validity

So far, we have dealt with the model of a target application. This model embodies the functional and temporal specification of target application, i. e., TTP/A applications, according to the conceptual model. Thus, we will introduce the term *application specification* or *ASPEC*. In formal notation we write an application specification $\mathbb{A}$ as a triple

$$\mathbb{A} = (\mathbb{S}, \mathbb{F}, \mathbb{T})$$

whereas $\mathbb{S}$ is the same as the set of jobs $\mathcal{S}$, $\mathbb{F}$ is the set of sets of functional requirements $\mathcal{F}$, and $\mathbb{T}$ takes the role of the set of sets of temporal requirements. According to the terminology of the conceptual model we substitute the placeholders $\mathbb{S}, \mathbb{F}, \mathbb{T}$ with the sets of table 3.2.

$$\mathbb{A} = (\mathbb{S}, \mathbb{F}, \mathbb{T}) = \left| \begin{array}{l} \mathbb{S} = \mathcal{S} \\ \mathbb{F} = \mathcal{F} \\ \mathbb{T} = \mathcal{P} \cup \mathcal{D} \end{array} \right| = (\mathcal{S}, \mathcal{F}, [\mathcal{P} \cup \mathcal{D}]) \tag{3.11}$$

At that point, we have integrated all entities of the conceptual model into one term of "application specification $\mathbb{A}$". However, we have not pointed out, how the integrity constraints $\mathcal{C}$ contribute to a valid application specification.

Obviously, each constraint $\zeta \in \mathcal{C}$ has to evaluate to "true" $(= \mathbf{t})$ concerning the model of a given target application, hence the application specification $\mathbb{A}$ of the target application. Therefore, we will give a formal definition of such "validity checking" or *semantic constraint checking function* $\mathcal{M}_\mathcal{C}$ regarding the set of model integrity constraints $\mathcal{C}$.

$$\mathcal{M}_\mathcal{C} : \mathbb{A} \rightarrow \mathbb{B} = \{\mathbf{t}, \mathbf{f}\} \tag{3.12}$$

In other words, the semantic constraint checking function $\mathcal{M}_\mathcal{C}$ determines, whether a given application specification $\mathbb{A}$ is valid against the set of constraints $\mathcal{C}$, if and only if that function evaluates to boolean true $\mathbf{t}$. Of course,

that function can even be applied for the evaluation of a single constraint $\zeta \in \mathcal{C}$ – we will call it $\mathcal{M}'_{\mathcal{C}}$, then. To sum up, we formulate validity of an application specification $\mathbb{A}$ against the set of integrity constraints $\mathcal{C}$ in equation 3.13.

$$\mathcal{M}_{\mathcal{C}}(\mathbb{A}) = \mathbf{t} \Longleftrightarrow \forall \zeta \in \mathcal{C} \mid \mathcal{M}'_{\mathcal{C}}(\zeta, \mathbb{A}) = \mathbf{t} \tag{3.13}$$

3.7.3 CCONF and feasibility

By now, we have defined the application specification with the conceptual model as a modelling formalism for distributed real-time embedded systems. However, model integrated computing (MIC) is far more powerful. We could even produce additional information from the target model on basis of the target model, for instance, source code, communication schedules, and cluster configurations as a whole.

Eventually, we will deal with communication schedules and *cluster configurations* or *CCONF* as part of the conceptual model. In the following sections we will give a formal definition of that entities of the conceptual model.

Schedules

The communication schedule Ψ or simply *schedule* is a function, which assigns a pair of a *start point* S_c and *ending point* T_c in the given multipartner round c for each job and functional requirement. So, we write a definition of the schedule function.

$$\Psi : (\mathbb{S} \cup \mathbb{F}) \rightarrow (\mathbb{N} \times \mathbb{N}) \tag{3.14}$$

The structure of such an entry is defined as follows:

$$[\chi : (S_c(\chi), T_c(\chi))] \qquad \text{with } \chi \in \mathbb{S} \cup \mathbb{F} \tag{3.15}$$

For example, we list a part of a possible schedule for the Smart-Fusion Application in figure 3.5.

$$\Psi(\mathbb{A}) = \{[Sensor1 : (1,1)], [Fusion : (5,7)], [f_{(S1,F)} : (2,2)], \ldots\}$$

As we can see, the job service $Sensor1$ starts at the TTP/A slot no. 1 and ends within the same slot. Thus, the execution of that job occupies only one TTP/A slot. On the contrary, the fusion job $Fusion$ starts at slot no. 5 and

ends in 7, which makes up a job execution time of 3 slots. The same applies to dataflow $f_{(S1,F)}$, which is transmitted within one slot. Consequently, we learn that the execution respectively transmission time $|\chi|$ (measured in TTP/A slots) of any $\chi \in (\mathbb{S} \cup \mathbb{F})$ in a given mutlipartner round c is calculated

$$|\chi| = \Delta_c(S_c(\chi), T_c(\chi)) + 1 \qquad \text{with } \chi \in (\mathbb{S} \cup \mathbb{F}) \tag{3.16}$$

Cluster configurations

Meanwhile, we have finalized all terminology in order to constitute the *cluster configuration* of a target application. The cluster configuration $\mathbb{C}$ of a given application specification $\mathbb{A}$ in a target system platform, which is embodied by the *decomposition* $\mathcal{Q}$, is a pair of the decomposition and a schedule $\Psi(\mathbb{A})$ of that application.

$$\mathbb{C}(\mathbb{A}) = (\mathcal{Q}, \Psi(\mathbb{A})) \tag{3.17}$$

As we can see in equation 3.17, the definition of the cluster configuration is related to that of the application specification in equation 3.11. In addition to this, we have to assure the *feasibility* of a cluster configuration against the **requirements** $\mathcal{R}$ mentioned along section 3.5.

Feasibility

Similar to the semantic constraint checking function $\mathcal{M}_{\mathcal{C}}$ we introduce a *semantic CCONF checking function* $\mathcal{M}_{\mathcal{R}}$. Furthermore, the CCONF checking function applies to each functional and temporal requirement $\rho \in \mathcal{R}$ – at this scope we will call the "requirement-specific" evaluation function $\mathcal{M}'_{\mathcal{R}}$.

$$\mathcal{M}_{\mathcal{R}} : \mathbb{C}(\mathbb{A}) \rightarrow \mathbb{B} = \{\mathbf{t}, \mathbf{f}\} \tag{3.18}$$

The purpose of these functions is to evaluate the compliance of the schedule $\Psi(\mathbb{A})$ in the cluster configuration $\mathbb{C}(\mathbb{A})$ to the functional and temporal requirements $\tau \in \mathbb{T}$ specified in the application specification.

In other words, $\mathcal{M}_{\mathcal{R}}$ examines the schedule in that way, whether all jobs complete execution before the expiration of some deadline, and whether transmission deadlines hold, and jobs in phase deviate with their starting points properly.

These requirements $\tau \in \mathbb{T}$ obey those functional temporal requirements $\rho_\tau \in \mathcal{R}$ defined in the conceptual model, whereas the index $_\tau$ denotes, that each type of requirement, i. e., deadline, phase, has to comply to the appropriate requirement.

Thus, a cluster configuration is *feasible*, if and only if each functional and temporal requirement is full-filled for the given cluster configuration and its application specification.

$$\mathcal{M}_{\mathcal{R}}(\mathbb{C}(\mathbb{A})) = \mathbf{t} \iff \forall \tau \in \mathbb{T}, \rho_\tau \in \mathcal{R} \mid \mathcal{M}'_{\mathcal{R}}(\tau, \rho_\tau, \Psi(\mathbb{A})) = \mathbf{t} \tag{3.19}$$

Chapter 4

Setting up a model-based tool suite

In the previous chapters we have outlined, how model integrated computing (MIC) can be used to elevate a generic modelling tool to an integrated development environment (IDE) for a specific domain in the engineering field of distributed embedded real-time systems. Furthermore, we have given a comprehensive definition of the conceptual model, which is the theoretical basis and the "template" for a *meta-model* of TTP/A applications.

Now, in this chapter we deal with a concrete implementation of the conceptual model as meta-model, and we describe an IDE for TTP/A Application Development. This is provided by the Generic Modeling Environment (GME).

It is beyond the scope of that work to appreciate every feature of GME. For that purpose we shall reference to the original documentation of GME ([LaBK+01]) and to its development project (`http://www.isis.vanderbilt.edu/projects/gme/`). We will just deal with the matter of GME, as far as it is needed for our intentions. It must be mentioned, that most content of that chapter has already been covered in earlier sections, especially when we come to the conceptual model's implementation in GME called *MetaTTPA*. We will not repeat every detail there, but we will just take a look into the "new shape" of the conceptual model and its constraints, which is entailed by the usage of GME.

4.1 The role of GME

In section 2.3 we proposed the structure of MIC systems in figure 2.1. At that place we declared a "work-flow" when using model integrated computing, beginning with meta-meta-models, coming to meta-models and target system models and eventually producing some target system component, which is of interest in the development process of the embedded system.

With the application of GME we present an implementation of that structure according to GME's features. In figure 4.1 we learn, how the placeholders of the MIC structure is substituted by concrete entities regarding TTP/A.

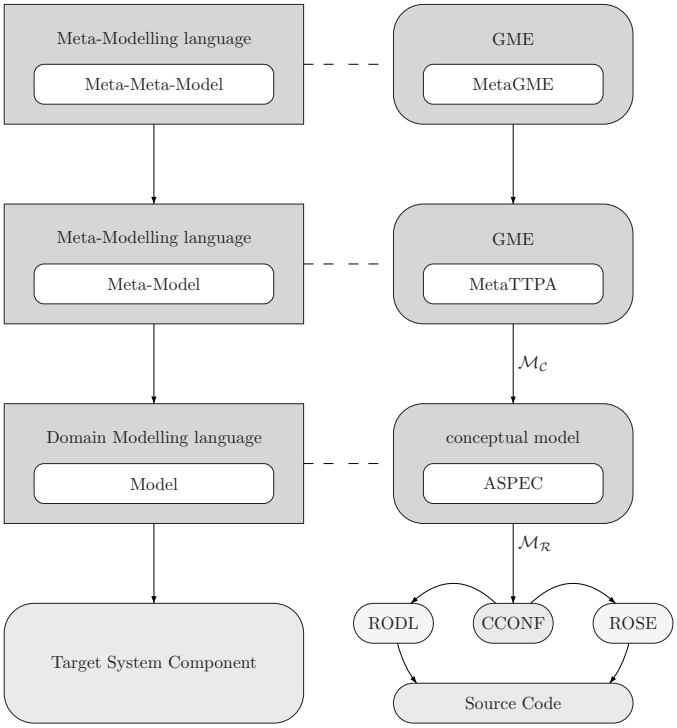

Figure 4.1: MIC with GME

As wee can see, the *meta-modelling language* is the built-in notational style of GME, which is a UML-related formalism. This language is used to express the *meta-meta-model* MetaGME, which is delivered as part of the tool suite and used to model all domain-specific meta-models. In addition, the actual *meta-model* is the implementation of the conceptual model and is called *MetaTTPA*. It is defined by means of the GME notation language and

checked against validity constraints of MetaGME. Finally, every TTP/A application is modelled in the *application specification*, which is part of the conceptual model and thus checked against the integrity constraints of the conceptual model $\mathcal{C}$ introduced in the previous chapters. The constraint checking function $\mathcal{M}_{\mathcal{C}}$ is realized through GME's built in constraint checking mechanism, although the set of integrity constraints is stated in MetaTTPA.

However, it must be mentioned that functional and temporal requirements are ensured separately by a GME extension, though part of the GME tool suite – in the next section we learn, that this is done by "schedulers". Despite GME's powerful constraint specification and validation mechanism, that feature is not sufficient for the complex semantic CCONF checking function $\mathcal{M}_{\mathcal{R}}$. Thus, it must be outsourced and realized by an own module or "plug-in", which hooks into GME's modular architecture – a so called *GME interpreter*.

The target system components are produced within the GME tool suite. They consist of the *cluster configuration* including decomposition and a schedule cohering the application specification. From the cluster configuration we can extract the RODLs for every TTP/A node and the ROSE for the master, and last but not least translate that specifications into source code. All this work is realized by means of *GME intepreters* within the GME tool suite.

4.2 Working with GME

As GME full-fills all the requirements as generic modelling tool (cf. section 1.3), it is even extensible and flexible by means of a modular architecture, which supports the integration of customized modules or – in GME's terminology – *interpreters*. Such interpreters may implement any useful behavior. In our case we designed interpreters so, that they contribute to the MIC system's work-flow. Figure 4.2 illustrates the work-flow of modelling target applications with GME.

As we learn from figure 4.2, the user begins the modelling process with the *application specification (ASPEC)* in the GME tool suite. This is the domain-specific model of any target application, i. e., the Smart-Fusion Application or an obstacle collision avoidance application.

Before the user runs the *scheduler*, he or she triggers the constraint checking function $\mathcal{M}_{\mathcal{C}}$ for the current application specification $\mathbb{A}$. Although these constraints are declared in MetaTTPA, it is GME's built-in constraint checking mechanism that performs that task.

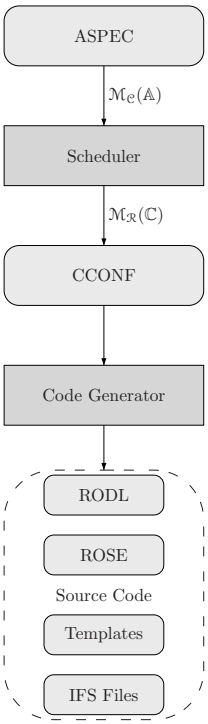

Figure 4.2: modelling work-flow with GME

If the ASPEC is valid according to the integrity constraints $\mathcal{C}$, the user will proceed by starting the scheduler. That scheduler is a GME interpreter itself. Currently, we implemented the *Straight Forward Scheduler*, which realizes the Straight Forward Scheduling Algorithm (see sec. 4.4.2). Each TTP/A scheduler is a GME interpreter and produces the cluster configuration (CCONF) $\mathbb{C}$, which is described in MetaTTPA and even operated within GME. Moreover, the scheduler is that interpreter, which conducts the semantic CCONF checking function $\mathcal{M}_{\mathcal{R}}$. After it has executed its built-in scheduling algorithm and wrote the model of the CCONF into the GME workspace, it evaluates that cluster configuration against the temporal and functional requirements imposed by the application specification (see sec. 3.7). Afterwards, the scheduler reports the result of $\mathcal{M}_{\mathcal{R}}$ to the user.

Finally, the user invokes the *code generator*, which is an interpreter, too. Currently, there exists one code generator, the *TTP/A C-Code Generator*. It takes a "feasible" CCONF from the GME workspace (the momentarily open GME project) as input and transforms that model into C source code. As a result, the system creates C source code files for each target host. Such

code file contains definitions of IFS Files, the host-specific RODL, the ROSE (only in the master's source code file), as well as "code stubs" for each job, i. e., C function headers for the tasks of each job. In other words, the source code includes everything we need to compile and link the target application, except the code of the jobs themselves.

Even though the generation of ASPEC, CCONF etc. is uni-directional, some steps in the whole process can be skipped in order to avoid the automatic generation features of the interpreters, and use the IDE in a more manual way. For example, the user could have skipped the definition of an ASPEC and begun directly with a CCONF. In that case, the user has to design the RODL for each node by himself. Another example, the user is enabled to generate a CCONF with a scheduler, edit the CCONF manually and eventually generate the source code.

As mentioned before, we have already implemented one scheduling interpreter and one code generator. However, it would be possible to develop another scheduling interpreter, which implements another scheduling algorithm compatible to the conceptual model (MetaTTPA). The user would have the choice between several schedulers to have the CCONF generated, unless the user models the CCONF by himself.
The same applies to code generators. For instance, if we intend not to produce C source code, but Assembler code, we need to use an alternate GME interpreter working as the code generator.

The user is enabled to **combine pairs of schedulers and code generators** in order to easily achieve **diversitive results**. The whole work-flow is highly flexible due to GME's architecture of extensibility via "plug-ins".

It must be mentioned, that all these steps take place within the GME tool suite. There is no other tool involved from the modelling entry when specifying the ASPEC until the extraction of source code. Therefore, this is what makes GME to an integrated development environment (IDE) for distributed embedded real-time systems, in that case for TTP/A target systems.

4.3 MetaTTPA

In this section we will take a look, how the conceptual model of TTP/A including functional and temporal requirements plus integrity constraints is realized by means of GME's notational formalism.

Generally, MetaTTPA includes 3 GME root models, which embody entities from the conceptual model or other TTP/A-specific entities.

- the model of the application specification (ASPEC)
- the model of the cluster configuration (CCONF)
- the model of the ROSE

The purpose of each GME model is obvious, they comply to the counterparts in the conceptual model. As models in GME are hierarchical, each root model covers the GME-style definition of the conceptual model, thus forming a container for the corresponding rule set in the conceptual model.

4.3.1 ASPEC in MetaTTPA

Figure 4.3 shows the definition of the application specification (ASPEC) in MetaTTPA, which makes up a part of the meta-model. According to the MIC structure introduced in figure 2.1, the meta-model is defined in the Meta-Modelling Language. In this case this language is the GME notational formalism [Gen05] (see figure 4.1). The similarity of GME with UML class diagrams [UML04] is remarkable.

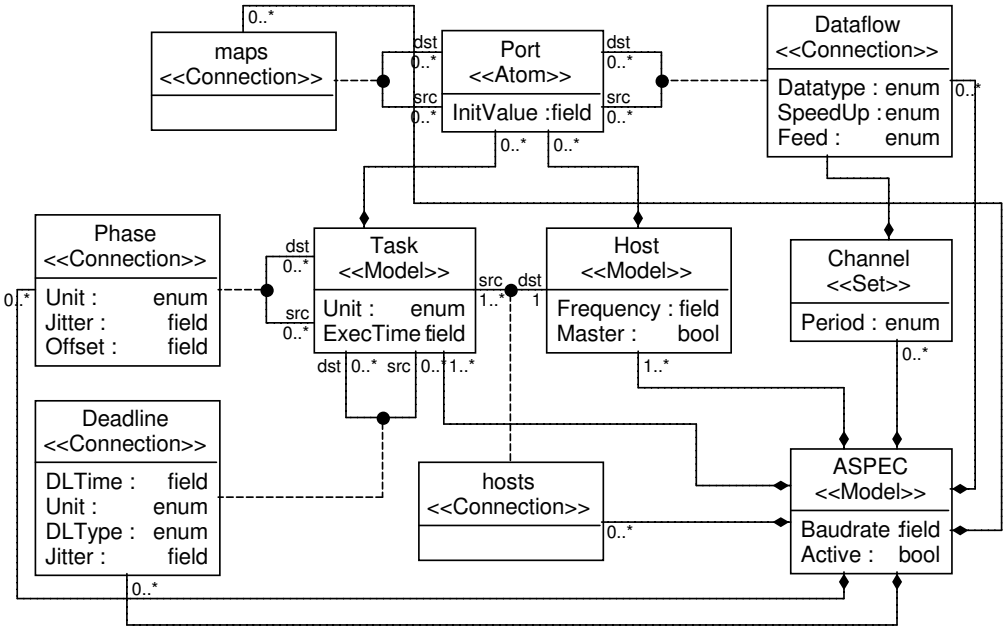

Figure 4.3: meta-definition of ASPECs in MetaTTPA

As we expect the reader to be used to UML notation, we will not particularly explain the meaning of that illustration.
This part of the meta-model defines the application specification in the conceptual model, which has already been covered extensively among chapter 3.

However, we learn from figure 4.3, that there are some irregularities in the ASPEC meta-model regarding the original conceptual model. For instance, the decomposition of a TTP/A cluster is declared in the ASPEC, but not in the CCONF as determined in the conceptual model. This fact hides in the UML-style connection "hosts" between the "Task"[1] and "Host" class symbol. Consequently, the user specifies the decomposition within the application specification and not the cluster configuration. MetaTTPA has been designed with this little discrepancy on behalf of practical reasons, although, this has no influence on the consistency of the conceptual model.

4.3.2 CCONF in MetaTTPA

Figure 4.4 depicts the meta-model excerpt of the cluster configuration (CCONF). Compared to the definition, the cluster configuration in the meta-model appears differently than the cluster configuration $\mathbb{C}$ in the conceptual model, even though they carry the same information.

The cluster configuration's implementation in GME focuses at "Slot"s, in that case this is the model of TTP/A slot. Instead of directly entering a schedule $\Psi(\mathbb{A})$ as a list of transmissions and job executions, we define, which activities take place during such a TTP/A slot and which host conducts that activity. For instance, in slot 2 host A executes service s and at the same time host B occupies the bus with a sending operation. Therefore, we have two entries in that slot model, one says that host A "executes" and B "transmits". If any activity takes longer than one slot, that operation will be present in consecutive slots, and the sequence of slots, where that activity is entered makes up the whole service execution or dataflow transmission.

So, the point of view is more locally concentrated around single slots. The global schedule results from the conjunction of the activities during each slot. Furthermore, both types of connections ("transmits", "executes") include references to the actual hosts respectively service tasks, which are involved. As a result, this maps the decomposition.

To sum up, the GME cluster configuration carries the same information of the schedule and the decomposition, hence it coheres with the conceptual

[1] Please note, that the term of the "job" is not included in MetaTTPA, as jobs are just representative for a set of physically executing tasks.

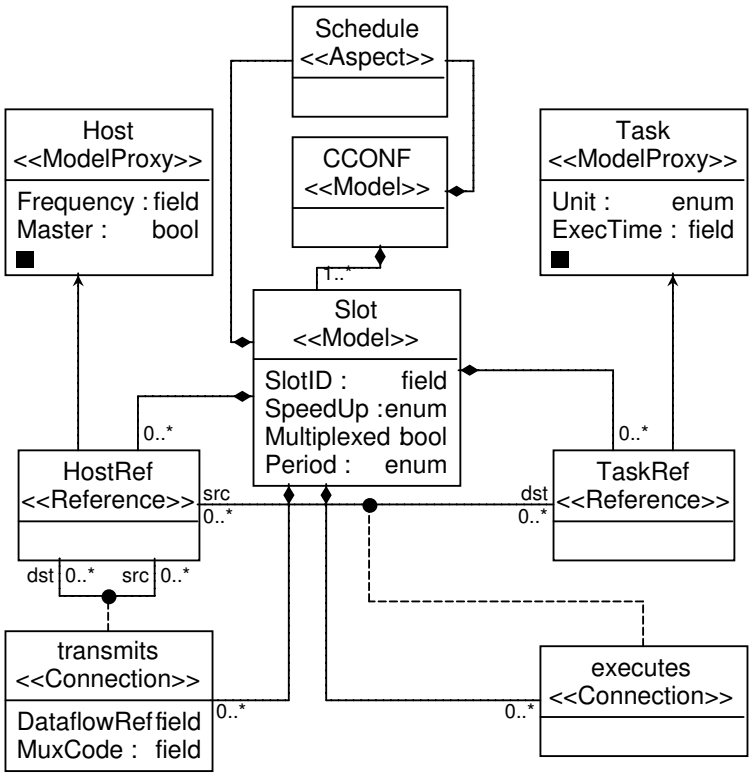

Figure 4.4: meta-definition of CCONFs in MetaTTPA

model.

4.3.3 ROSE in MetaTTPA

Figure 4.5 illustrates the meta-model of the ROSE in MetaTTPA.

Even though the ROSE does not contribute to the ASPEC or CCONF of a target application model, it is included in the meta-model and finally in target models for completeness, indeed. Actually, the structure of the ROSE will rarely deviate from the recommended TTP/A round sequence in figure 3.2. Anyhow, with this definition in MetaTTPA it is possible to describe larger ROSEs with a more complicated structure in the target model, i. e., an arbitrary sequence of multipartner rounds with different fireworks bytes combined with the MSA/MSD rounds.

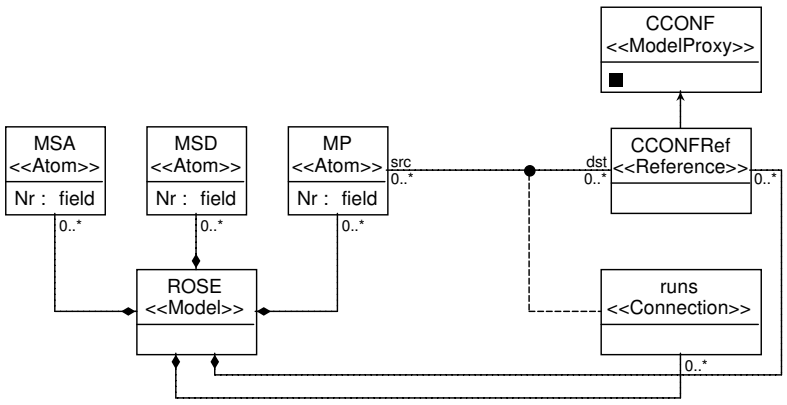

Figure 4.5: meta-definition of ROSE in MetaTTPA

4.3.4 Aspects

MetaTTPA offers 5 aspects, which differently show to and hide information from the user in order to present a manageable workspace. We will list them in the following.

- Decomposition
- Dependencies
- IFS Management
- Schedule
- ROSE View

The “Schedule” aspect and the “ROSE View” just concern the CCONF respectively the ROSE models. In no other aspect it is possible to display and edit the models of that entities of MetaTTPA. On the contrary, the other three aspects deal with the ASPEC. Obviously, it is very practical to enable a filtered view in the complex matter of the application specification.

Consequently, "Decomposition" deals with the same theme of the conceptual model, as its name suggests. In the aspect of "Dependencies" the user models the functional and temporal requirements, hence relations among the entities of the conceptual model, i. e., phases, deadlines etc. Last but not least, the "IFS Management" handles the mapping of IFS chunks within one host, if several tasks share memory (see section 3.4).

4.3.5 Attributes

Purpose

As GME is a UML-related notation formalism, it supports the concept of *attributes* for each entity in MetaTTPA. In that case, attributes are used to model information concerning the conceptual model, which can not be expressed by means of connections, multiplicity, or containment etc.

Generally, MetaTTPA offers attributes in order to determine concrete **values of functional and temporal requirements**, i. e., the value of deadlines (deadline time), service execution times, phase offsets etc. Furthermore, that meta-model includes attributes, which handle TTP/A-related specifications. For instance, a "Host" object contains a boolean attribute named "Master". If the attribute has the value "true", this host is the master node in the TTP/A cluster.

Essential attributes

Even though completely ignored in the conceptual model, two inconspicuous attributes are of upmost importance for **scheduling** and the **semantic CCONF checking function**:

- the Frequency attribute in "Host"s
- the Baudrate attribute in the ASPEC root model

These two integer values define the pace of time in the TTP/A cluster. Because TTP/A follows the TDMA paradigm, the communication is divided into slots. The slots carry a fixed size of bits (see [Kop01]). The duration of a slot is dependent on the duration of each single bit, which in turn follows from the transmission speed – the *baudrate* – of the communication system. Therefore, that attribute is mandatory, because it is a factor for calculating the number of slots, a given activity, i. e., a service execution, deadline, will

occupy.
According to this, the performance of a single node is determined by its core *frequency*. It makes a statement about the number of slots, a service task execution will last according to the specified value in the target model.

All these quantities have to be considered, when it comes to the scheduling and generation of cluster configurations as well as the feasibility checking of that schedule, because the attributes contribute to the information, when an activity will actually start and terminate, and wether this is sufficient regarding temporal requirements.

Data types

The figures 4.3, 4.4, and 4.5 also illustrate the attributes of each GME entity. In most cases, the meaning of the attributes is self-explanatory due to their names. Otherwise, the user will be assisted by useful explanations of the attributes when working with the GME tool suit, as descriptions of that entities can by included in the meta-model. It must be mentioned, that one attribute definition can be re-used among different entities. For instance, the "Unit" is contained by "Task"s, "Phase"s, and "Deadline"s.

We also learn from the pictures, what data types the attributes possess. The data type "boolean" is obvious, "field" attributes may accept values of the types "integer", "double", or textual data (strings). In the next section 4.3.6 we will learn, what value ranges such attributes may assume.

Enumerations

Finally, we come across some "enumerations". These are attributes with a pre-defined set of possible values, which has been determined in the meta-model. As a result, any faulty user-input is excluded, because only the offered values can be chosen by the user. We will quickly list the possible values for each attribute.

Datatype That attribute determines the data type of the data conveyed via dataflow as well as the connected ports. For instance, the C-code protocol stack of TTP/A declares `ifs_uint8_t`, which is an unsigned, 8-bit wide integer type. All possible entries of the enumeration are defined in the TTP/A protocol specification [Kop01].

SpeedUp This enumeration offers the speed-up factor of a transmission to the user. TTP/A currently supports factors of $\{4, 8, 16, 32\}$ and no

speed-up at all (cmp sec. 3.5.3).

Feed The user declares a dataflow as forward or backward feed with this attribute.

Period Similar to the speed-up, this attribute offers a selection of the periods of multiplexed virtual channels. TTP/A allows periods of 4 or 8 iterations in its current implementation (cmp. sec. 3.5.3).

DLType The type of a deadline. As mentioned in section 3.5.2, deadlines can be "periodic" or "offset".

Unit Because the user most likely does not want to enter time-related values like service execution times or deadlines measured in number of TTP/A slots, MetaTTPA offers the specification of such values in metric units. So, this enumeration includes timely dimension ranging from "ns" (nano second) to "sec" (second). Moreover, especially for service tasks the unit "cycles" is available. So, a task execution time can be given by the number of processor cycles the local algorithm takes for operation.

4.3.6 Constraints

As mentioned earlier, GME provides a built-in constraint checking mechanism. That constraints are defined within the GME models.

In that case MetaTTPA includes all in all about 45 OCL constraints [OCL04], whereas the major part embodies the set of integrity constraints introduced in section 3.6.2 for the application specification as well as the cluster configuration.

Besides this, that meta-model is equipped with 10 so called *trivial constraints*. Actually, these trivial constraints do not contribute to ensure validity of target models. Their purpose is to perform a **value check** on attribute values. As a result, the target model is resistant against faulty user input.

The purpose of that enlargement of MetaTTPA are practical reasons. If such value checks can be conducted within the GME tool suit by means of the constraint checking mechanism, we need not check the value of each attribute in the TTP/A interpreters in order to ensure correctly configured target system components. Therefore, the interpreters are slim and stripped down of unnecessary value checking code. The tables 4.1 and 4.2 list the trivial constraints in the GME models of the application specification respectively cluster configuration.

Name	Context	Description
VC_Frequency VC_Baudrate VC_DLTime VC_Jitter VC_Offset VC_ExecTime	Host ASPEC root Deadline Deadline, Phase Phase Task	The value of that attribute must be a positive double respectively integer.
VC_InitValue	Port	The initial value must be a hexadecimal number.
Cycle_Unit	Task	Only Tasks can use the unit "cycles".

Table 4.1: trivial constaints in the ASPEC

Name	Context	Description
VC_MuxCode	transmits	The multiplexer code must be greater than 0, but less than the period of the multiplexed virtual channel.
VC_SlotID	Slot	A Slot-ID must be between 0 and 64.

Table 4.2: trivial constaints in the CCONF

4.4 The interpreters

Eventually, we briefly discuss the TTP/A-specific GME interpreters. In earlier sections (see sec. 4.2) their purpose in the modelling work-flow has been outlined. Therefore, there is not much left to tell about them. Anyhow, it is beyond the scope of that document to go into the technical software design of that modules. This will be subject of an upcoming *developer documentation* of the TTP/A schedulers.

Even though their functionality is quite straightforward, the interpreters are realized by means of several technologies. They take advantage of the *Builder Object Network (BON)* (version 2) [Gen05], which encapsulates the COM-access to GME's interiors. Hence, they are implemented in C++ on the Windows Platform and seamlessly integrated in the GME tool suite. Consequently, the user does not notice their presence, until he triggers the scheduling or code generation.

4.4.1 The TTP/A C-Code Generator

The *TTP/A C-Code Generator* implements a "code generator" according to the GME work-flow introduced in figure 4.2. As it names suggests, it transforms a feasible cluster configuration into C-Code. That C-Code corresponds to the current protocol stack implementation of TTP/A.

For each node in the TTP/A cluster the code generator creates a source code file, where it places the following items according to the protocol stack:

- type declaration of the host's I/O file.
- a variable definition, which is an instance of the I/O file type. Moreover, that structure is initialized with the initial values given in the "InitValue" attribute of the corresponding ports.
- type declaration of all task files (one structure for each task). That structures are made up of the data type `ifs_addr_t`.
- a variable definition for each task file with their appropriate type. Each variable is initialized with the *address of the mapped port in the I/O file*. Thus, the task file ports reference the ports in the I/O file as proposed in section 3.4.
- macros `IFS_ADDAPPLFILE`, which "register" each IFS file with the associated service task function. Thus, this creates a link between a task file instance and its associated service task funktion. However, the I/O file does not belong to any function, because it is global.
- "service task templates". These are C-functions, which only include the header but no code in the body. They embody service tasks and will contain their local algorithms written by the programmer.

For further information on the TTP/A C-Code protocol implementation see [EHK+02]. It documents that "source code framework", how IFS files are declared in source code, how we link IFS files with tasks etc.

4.4.2 The Straight Forward Scheduler

The *Straight Forward Scheduler* is a GME interpreter, too, which takes the role of a scheduler in the TTP/A modelling process. It name results from the fact, that it implements the *Straight Forward Scheduling Algorithm*, formerly

known as "TTP/A Scheduling Algorithm" introduced in [Pau04]. However, since its first appearance it has evolved due to the enlargement of the conceptual model on behalf of the application of GME. Because the purpose of the scheduling interpreter has already been discussed sufficiently in earlier sections, here will just deal with that recent revision of that algorithm.

Generally, the Straight Forward Scheduling Algorithm is a *constructive greedy heuristic*, which may not always produce optimal schedules. However, a schedule $\Psi(\mathbb{A})$ calculated with that scheduling algorithm evaluates to "feasible" by means of the semantic CCONF checking function $\mathcal{M}_{\mathcal{R}}(\mathbb{C})$ in most cases. So, even though that algorithm might not produce optimal results, it guarantees the compliance of the schedule to all temporal requirements, in case it finishes successfully.

The main ideas behind the Straight Forward Scheduling Algorithm are the requirements 6 and 7 in section 3.5.3. They say, that a service can not be executed, unless it has received all its incoming dataflow. After execution it dispatches all its outgoing dataflow.

We will list the functionality of the algorithm in natural language in the following.

1. find all services with all incoming **forward-feed** dataflow already arrived or no (incoming forward-feed) dataflow at all
2. if some of that services are in phase, add the offset to the starting point of that service with the later deadline or to the service with no deadline
3. add the starting point of the services in the schedule
4. at the ending point of each services append the sending operation of all outgoing dataflow to the schedule, but concern collision on the bus (in that case a dataflow has to occupy the next free slot).
5. mark the services as done
6. unless all services are marked, go to (1)
7. append a sending operation for each backward-feed dataflow
8. reserve slots for each multiplexed virtual channel
9. create the sub-schedule for each multiplexed virtual channel

Chapter 5

Conclusion

5.1 Summary

We described the adaption of the generic modelling tool GME to an integrated development environment in order to support software development projects for distributed real-time embedded systems. TTP/A is the basic transmission protocol in this case study, which has evolved to a versatile and efficient time-triggered real-time communication subsystem.

We proposed the conceptual model in its most recent level of maturity, which describes the way TTP/A applications should be designed. Moreover, we introduced an implementation of that conceptual model in GME by means of a GME meta-model named *MetaTTPA*, which is used to describe concrete TTP/A target applications. That meta-model covers the terminology, rule set, functional and temporal requirements, as well as integrity constraints according to the conceptual model.

GME was the tool of choice, because it provides a flexible and extensible modular architecture and an additional powerful OCL constraint checking mechanism. Thus, its yet rich feature set could be extended by means of customized plug-ins or *interpreters*. In our case study we designed such additional interpreters, which contribute to the work-flow of model integrated computing (MIC) concerning our target systems equipped with TTP/A. The scheduling and code generating interpreters transform the generic modelling tool suite into a specialized integrated development environment (IDE).

We introduced one implementation of each type of TTP/A-specific interpreters. The first automatically creates schedules using the *Straight Forward Scheduling Algorithm* and writes the cluster configuration, which in turn is

semantically checked against functional and temporal requirements imposed by the application specification. The second interpreter takes a feasible cluster configuration as input and produces C source code files including RODL, ROSE, IFS files, and service task templates. Its name is the *TTP/A C-Code Generator*.

As a result, the whole modelling process beginning with the modelling of the application specification, to cluster configurations until the production of target system components, i. e., the source code files, takes place within one tool suite. Eventually, this makes GME to an integrated development environment (IDE) for TTP/A application development.

5.2 Outlook

One idea for my future work is the analysis of existing source code in order to re-construct an application specification and cluster configuration. In the end, we would have a bi-directional work-flow. On field of application for this feature would be re-modelling of TTP/A source code projects into the modelling formalism, so that we can conduct better maintenance and re-usage of legacy code.

The conceptual model already suits to these needs. The major development of that idea would focus on the scheduling and code generating interpreters, which have to be designed to support the modelling in both directions. Maybe, the meta-model would have been modified for practical reasons.

Another challenge is to extend the conceptual model to include service-level functional requirements. As a consequence the meta-model would be enriched, too.

The local algorithms, which are ignored and hidden behind the service's interfaces at the moment, shall become part of the application specification. So far, we have just been enabled to generate the service task templates, but no embedded code. With this enrichment, it will be possible to include the automatic source code generation of local algorithms by GME interpreters. As a result, we would be able to completely model a TTP/A application in GME notation and let the whole source code of a target application be generated by the GME interpreters.

Bibliography

[BH04] Luciano Baresi and Reiko Heckel. *Tutorial Introduction to Graph Transformation: A Software Engineering Perspective*, 2004. `http://www.elet.polimi.it/upload/baresi/papers/ICGT04.pdf`.

[Bos91] Robert Bosch GmbH. CAN Specification Version 2.0. Technical report, Robert Bosch GmbH, Stuttgart, Germany, 1991.

[Dej05] D. Dejmek. Probleme und Fallstricke in Embedded Systems Programming. Research report, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2005.

[Dia05] University in Erlangen. *DiaGen Overview*, 2005. `http://dokumente.unibw.de/pub/bscw.cgi/d443087/Overview.pdf`.

[dLV02] Juan de Lara and Hans Vangheluwe. Using AToM3 as a Meta-CASE Tool. In *4th International Conference on Enterprise Information Systems*, Ciudad Real, Spain, April 2002. `http://www.cs.mcgill.ca/~hv/publications/02.ICEIS.MCASE.pdf`.

[EHK+02] W. Elmenreich, W. Haidinger, R. Kirner, T. Losert, R. Obermaisser, and C. Trödhandl. TTP/A Smart Transducer Programming – A Beginner's Guide. Technical Report 33/2002, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002. Version 0.5.

[Elm02] W. Elmenreich. *Sensor Fusion in Time-Triggered Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 2002.

[EPS04] W. Elmenreich, S. Pitzek, and M. Schlager. Modeling distributed embedded applications on an interface file system. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 175–182, May 2004.

[Gen05] Institute for Software Integrated System, Vanderbilt University. *GME 5 User's manual*, September 2005. `http://www.isis.vanderbilt.edu/projects/gme/GMEUMan.pdf`.

[Jen97] Kurt Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume I - III of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer Berlin / Heidelberg, 2nd edition, December 1997.

[JKK+02] C. Jones, M.-O. Killijian, H. Kopetz, E. Marsden, N. Moffat, D. Powell, B. Randell, A. Romanovsky, R. Stroud, and V. Issarny. Final version of the DSoS conceptual model. *DSoS Project (IST-1999-11585) Deliverable CSDA1*, October 2002. Available as Research Report 54/2002 at http://www.vmars.tuwien.ac.at.

[KB03] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 – 126, January 2003.

[Kop97] H. Kopetz. *Real-Time Systems, Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.

[Kop01] H. Kopetz et al. Specification of the TTP/A Protocol. Research Report 61/2001, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2001. Version 2.00.

[KPM05] P. T. Kovács, G. Pintér, and I. Majzik. UML Based Design of Time Triggered Systems. Technical Report, Budapest University of Technology and Economics, Department of Measurement and Information Systems, H-1117 Budapest, Magyar Tudósok krt. 2, 2005.

[LaBK+01] A. Ledeczi, M. Maroti andA. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi. The Generic Modeling Environment. In *Proceedings of WISP 2001*, Budapest, Hungary, May 2001.

[LBM+01] A. Ledeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing domain-specific design environments. *Computer*, pages 44–51, November 2001.

[LW99] Bo Lindstrom and Lisa Wells. *Design/CPN Performance Tool Manual*. University of Aarhus, Aarhus, Denmark, 1.0 edition,

September 1999. `http://www.daimi.au.dk/designCPN/man/Misc/Performance.pdf`.

[MAR05] MoDELS 2005 International Workshop OCLWS, MoDeVA, MARTES, AOM, MTiP, WiSME, MODAUI, Nfc, MDD, WUsCaM. In Jean-Michel Bruel, editor, *Satellite Events at the MoDELS 2005 Conference*, pages 58–66, Montego Bay, Jamaica, October 2005. ACM/IEEE, Springer Berlin / Heidelberg. `http://www.cs.colostate.edu/models05/`.

[MDA05] OMG. *A Proposal for an MDA Foundation Model*, April 2005. `http://www.omg.org/docs/formal/05-04-01.pdf`.

[MoD05] 8th International Conference (MoDELS 2005). In Lionel Briand and Clay Williams, editor, *Model Driven Engineering Languages and Systems*, Montego Bay, Jamaica, October 2005. ACM/IEEE, Springer Berlin / Heidelberg. `http://www.cs.colostate.edu/models05/`.

[MOF02] OMG. *MetaObjectFacility(MOF) Specification*, April 2002. `http://www.omg.org/docs/formal/02-04-03.pdf`.

[OCL04] OMG. *UML 2.0 OCL Specification*, April 2004. http://http://www.omg.org/docs/ptc/03-10-14.pdf.

[OMG02] Smart Transducers Interface. Final Adopted Specification Document Number ptc/2002-10-02, Object Management Group, Needham, MA, U.S.A., August 2002. Available at `http://doc.omg.org/ptc/2002-10-02`.

[Pau04] C. Paukovits. Modellierung und Scheduling von flexiblen, zeitgesteuerten Kommunikationsprotokollen. Bachelor's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.

[QoS04] OMG. *UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms*, September 2004. `http://www.omg.org/docs/ptc/04-09-01.pdf`.

[Rob05] Robert Pettit and Hassan Gomaa. Modeling and Analysis of Concurrent and Real-Time Object-Oriented Designs. In *Satellite Events at the MoDELS 2005 Conference*, Montego Bay, Jamaica, October 2005. ACM/IEEE.

[SPT05] OMG. *UML Profile for Schedulability, Performance, and Time*, January 2005. http://www.omg.org/docs/formal/05-01-02.pdf.

[Sys05] IEEE. *Standard SystemC Language Reference Manual (IEEE 1666)*, November 2005. http://standards.ieee.org/reading/ieee/std/dasc/P1666_DD2.1.1.pdf.

[TTA03] TTAGroup. *Specification of the TTP/C Protocol*. TTAGroup, 2003. Available at http://www.ttagroup.org.

[TTT05] TTTech Computertechnik AG, Schoenbrunner Strasse 7, A-1040 Vienna, Austria. *The OSEKtime-Based Operating System for Safety-Critical Real-Time Applications (TTP-OS)*, 2005. http://www.tttech.com/technology/docs/general/TTTech-TTP-OS.pdf.

[TTT06a] TTTech Computertechnik AG. TTPbuild - The Node Design Tool for the Time-Triggered Protocol TTP/C, 2006. http://www.tttech.com/products/software/ttpbuild/.

[TTT06b] TTTech Computertechnik AG. TTPplan - The Cluster Design Tool for the Time-Triggered Protocol TTP/C, 2006. http://www.tttech.com/products/software/ttpplan/.

[UML04] OMG. *UML 2.0 Specification*, April 2004. http://www.omg.org/cgi-bin/doc?ptc/2004-10-14.

[vdW00] Hans-Christian v. d. Wense, editor. *LIN Specification Package: Revision 1.2*. Motorola, Munich, Germany, 2000.

[VHD00] IEEE. *IEEE Standard VHDL Language Reference Manual (IEEE 1076)*, 2000. http://standards.ieee.org/reading/ieee/std/dasc/1076-2000.pdf.

[XMI02] OMG. *XML Metadata Interchange (XMI)*, 2002. http://www.omg.org/technology/documents/formal/xmi.htm.