Modeling and Analysis of Real-Time Software Systems using UML

Abdelouahed Gherbi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfilment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

December 2007

Your file Votre référence
ISBN: 978-0-494-37752-9
Our file Notre référence
ISBN: 978-0-494-37752-9

ABSTRACT

Modeling and Analysis of Real-Time Software Systems using UML

Abdelouahed Gherbi, Ph.D.
Concordia University, 2007

Real-Time Systems (RTS) should not only function correctly but should also satisfy time constraints. RTS include embedded systems, which are used nowadays in a variety of applications. These are, for instance, house appliances, automotive, aeronautic/aerospace, and health monitoring systems, to mention just a few. The design of such systems is complex and challenging. In order to cope with the complexity of RTS, there is shift in their development to follow a model-driven approach, such as the Model Driven Architecture (MDA), which relies on using models of high level of abstraction. The Unified Modeling Language (UML) is the Object Management Group (OMG) standard modeling language to support MDA. UML is appropriate for software systems because it allows for a multi-view modeling approach through its multitude of diagrams covering the structure, the behavior and the deployment architecture. Moreover, UML is also used in the domain of real-time software systems. This is achieved through its profiles, including, the OMG standard profile for Schedulability, Performance and Time (UML/SPT) or the upcoming standard UML Profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). However, UML modeling faces some challenging issues such as model consistency. This issue becomes worse in the context of real-time software systems because additional aspects should be taken into consideration, including time, concurrency and schedulability. In this thesis, we address several issues related to modeling and validation of RTS with UML. We focus in particular on the consistency of UML/SPT models. We adopt an incremental approach to check the consistency of these models by distinguishing the syntactic and semantic levels. The latter is further decomposed into behavioral, concurrency-related and time consistency. Our contributions in this thesis are fourfold. First, we leverage the extensibility mechanisms of UML to propose an extension to UML/SPT. This extension enables the modeling of multicast communications, which is required for the description of the behavior of certain real-time protocols. Second, we propose a formalization of the concurrency modeling

capability in UML/SPT using timed automata. This formal semantics allows for applying well-established model checking techniques to check concurrency related consistency in UML/SPT models. Third, we propose an MDA-compliant approach to enable schedulability analysis of UML/SPT models. We present a proof of concept for this approach through a prototype implementation using the Atlas Transformation Language (ATL) and XML-based technologies. Finally, we use the schedulability analysis applied to UML/SPT models in order to check the time consistency of a system design modeled by means of a set of state machines with respect to time constraints modeled using a set of sequence diagrams annotated with UML/SPT time stereotypes.

Keywords: Real-time systems, Model-driven Architecture, UML, UML/SPT, Model transformation, ATL, XML, XSLT, Consistency, Concurrency, Model Checking, Schedulability Analysis.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my thesis director, Dr. Ferhat Khendek. Personally, I thank him for giving me the opportunity to make my Ph.D dream come true and for being there for me in moments of need. Professionally, his always frank, strong and sound comments and feedback on the different issues relevant for this thesis guided me throughout my research work.

I would like to thank the examining committee: Dr. H. Sahraoui (external examiner from Département d'informatique et de recherche oprationnelle, Université de Montréal), Dr. R. Dssouli from CIISE, Dr. J. Rilling (Computer Science & Software Engineering), and Dr. P. Gohari (Electrical and Computer Engineering). Their excellent comments and feedback contributed to enhance the quality of this thesis.

I would like to thank Dr. M. Debbabi for his support when I was doing my research work for my Master at Université de Constantine and for giving me the chance to join the LSFM research group in Université Laval and the Computer Security Laboratory at CIISE, where I spent the first period of my PhD program.

I would like to thank Dr. R. Dssouli from CIISE, Concordia University; Dr. R. Boutaba from Waterloo University; and Dr. M. Mejri and Dr. N. Tawbi from Université Laval for their help and support.

During my PhD journey I met and interacted with different people who made this PhD experience humanely enriching and interesting. These are my colleagues: R. Karunamurthy and A. Ntozi from the Telecommunication Software Engineering Lab; H. Yahyaoui, C. Talhi, S. Zhioua, and L. Ketari from the LSFM group; Ahmed Gario, Marc-André Laverdière and Nadia Belblidia from the CSL research group and Andreas Rasche from Operating Systems and Middleware Group at Hasso Platner Institut.

My thoughts and deep gratitude go straight to my mother and my father. I thank them endlessly for bringing that disabled little boy to school! I would like them to know that I am eternally indebted to them. Finally, I reserve a special word for Ihsan, a woman of an extraordinarily courage, intelligence and a great heart. Thank you very much for your tremendous support.

Table of Contents

List of Acronyms

ATL Atlas Transformation Language

CTL Computation Tree Logic

DSL Domain Specific Languages

INCOSE International Council on Systems Engineering

MDA Model Driven Architecture

MOF Meta Object Facility

MARTE UML Profile for Modeling and Analysis of Real-Time and Embedded Systems

MSC Message Sequence Charts

KM3 Kernel MetaMetaModel

OCL Object Constraint Language

OMG Object Management Group

OMT Object Modeling Technique

OOSE Object-Oriented Software Engineering

PIM Platform Independent Model

PSM Platform Specific Model

QVT Query Views Transformations

ROOM Real-time Object Oriented Methodology

RTMP2 Reliable Message Transport Protocol

SDL Systems Description Language

SOC System On Chip

SysML System Engineering Modeling Language

TURTLE Timed UML and RT-Lotos Environment

UML Unified Modeling Language

UML/SPT UML Profile for Schedulability, Performance and Time

UML-RT UML Profile for Real-Time

UML/QoS UML Profile for Quality of Service and Fault Tolerance Characteristics and Mechanisms

UPPAAL UPPsala and AALborg universities

XML eXtensible Markup Language

XMI XML Metadata Interchange

XSL eXtensible Style Sheet

XSLT XSL Transformations

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Real-Time Systems (RTS) are commonly defined as systems which are required not only to carry out their functionality correctly but to also satisfy a set of time constraints [121]. Most of RTS are embedded systems nowadays and are used in a wide variety of applications, such as consumer electronics (e.g. DVD and MP3 players), automotive (e.g. ABS), aircrafts (e.g. flight control systems), telecommunication systems (e.g. mobile phones), medical systems, military applications, and smart buildings to mention just these examples [76].

Real-time software systems have several characteristics. They are *reactive* and *concurrent* systems because they often interact with the physical world where different events can happen concurrently. They should react to many events concurrently. Moreover, such reaction should often satisfy some time constraints. Furthermore, as suggested by the aforementioned list of applications, real-time systems are often also *safety-critical* systems. They have severe dependability requirements in order to preserve life and/or property wherever these systems are used. These characteristics contribute to make the design of real-time systems complex and challenging.

In order to address the real-time software design complexity, there is a need to shift from ad hoc optimization techniques to high level abstractions, models and model-based developments methodologies [25][42] [70] [113] [114]. Model-Driven Development (MDD) is a software development approach, where models are first-class artifacts. The basic idea underlying MDD is that a software system can be developed starting with a high-level abstract model, which is then successively refined/enriched until, eventually, reaching a concrete implementation [77]. Model-Driven Architecture (MDA) [78] is an example of MDD promoted by the OMG. MDA separates the business/application logic from the underlying technology used for its implementation. The key concepts in MDA are *Platform Independent Model* (PIM), *Platform Specific Model* (PSM), and model *transformation*. The OMG supports MDA by defining a set of standards, which include the Unified Modeling Language (UML)[92].

UML is the *de facto* standard modeling language for software-intensive systems. UML is successful because it presents many features. It is a visual language [48], which makes it very intuitive for the user. It is a multi-view modeling language allowing to cover separately different aspects of a system (structure, behavior and deployment) using a variety of diagrams. This feature is very important to deal with the system's complexity. Finally, UML is customizable to the particularities of different domains through its extensibility mechanisms and profiles. The success of UML led to a surge of interest in using UML by system engineering community [52], [95] and System-On-Chip community [76] [96].

UML has been used in the context of embedded and real-time software systems first as a backbone for some development methodologies including ROOM [114], COMET/UML [42] and ROPES [25]. UML has then been the focus of many research initiatives as a modeling language for embedded and real-time software systems [70]. UML is adapted to the specifics of this domain through a variety of profiles developed in academia and the industry [38]. The most important UML profile for embedded and real-time software systems is the OMG standard called UML profile for *Schedulability, Performance and Time* [91] denoted UML/SPT throughout this thesis. This profile is in the process of a major revamp to define

the UML profile for *Modeling and Analysis of Real-Time and Embedded systems* (MARTE) [97]. UML/SPT is a general framework for the modeling and analysis of real-time designs. It enables the modeling of resources and quality of service; time concept and time-related mechanisms; and concurrency. As for model analysis, UML/SPT supports schedulability and performance analysis. It provides the end-user with a set of stereotypes and tagged values that can be used to annotate UML design models with quantitative information. This enables the prediction of key properties in the early stages of a software development process using quantitative analysis techniques (schedulability and performance analysis).

1.2 Issues

We have identified several issues related to the usage of UML and UML/SPT for the modeling and analysis of real-time software systems. These issues can be summarized as follows:

- Limitations in the UML/SPT expressiveness with respect to some real-time requirements modeling needs. Specifically, UML/SPT does not allow to model multicast communication required for some distributed real-time systems.

- UML/SPT is defined like UML itself using the metamodeling approach. UML/SPT metamodel is composed of a set of domain models where the semantics of the different concepts is defined informally in English. This lack of formal semantics hinders automatic manipulation of UML/SPT models either for verification or implementation synthesis purposes.

- UML/SPT is designed to support schedulability analysis of UML design models in the early stages of the development process. There are several schedulability analysis techniques in the literature. There is, however, a semantic gap between UML/SPT models and the task models expected/used by well-established real-time analysis techniques such as schedulability analysis. It is important to bridge this semantic gap in order to enable the application of these techniques for the validation of the schedulability property of UML/SPT models.

- As mentioned earlier, UML supports a multi-view modeling approach. It provides a multitude of diagrams to cover the structure, the behavior, and the deployment architecture of the system under consideration. This is very advantageous to cope with software complexity. However, these different views may be inconsistent. In addition, when UML is also used to model real-time systems using its profiles for real-time, UML/SPT for example, new aspects are taken into consideration. These include mainly concurrency, time constraints, and schedulability. These aspects may contribute to worsen the consistency issue.

1.3 Contributions

We have investigated the different issues listed above and developed different techniques and methods for addressing them. The main contributions of this thesis can be summarized as follows:

- We have surveyed the different UML profiles used to model real-time systems proposed in the academia as well as in the industry. We have established an assessment of their capabilities and limitations with respect to a variety of criteria such as formal foundation, tool support, etc. [38].

- UML is defined using the metamodeling approach and it has been designed with built-in extensibility mechanisms. These are used to define different domain-specific versions of UML, the UML profiles. We have used this approach and leveraged UML extensibility mechanisms to define an extension of UML/SPT. This extension enables to model multicast communications necessary to capture the behavioral requirements of protocols used in distributed real-time systems, such as the Reliable Message Transport Protocol (RMTP2). We have established a comparison with a similar extension for MSC [36].

- We have proposed a formal definition of the semantics of the concurrency domain model of UML/SPT using the formalism of timed automata. This formal semantics

enables applying model checking techniques using tools such as UPPAAL in order to verify the concurrency-related consistency of a UML/SPT model [41].

- We have investigated appropriate schedulability analysis techniques for UML/SPT models [35]. In particular, we have focused on a schedulability analysis technique used for object oriented design of real-time systems [111], [112]. We have defined an MDA-compliant approach to bridge the semantic gap between the task model used by this schedulability technique and UML/SPT models [37]. We have implemented a proof of concept for this approach as a prototype implementation using both ATL transformation language and XML technologies [40].

- We have defined a consistency framework for UML/SPT models. This framework addresses incrementally the various aspects of consistency including syntactic, semantic, concurrency-related and time consistency. In this framework, we introduced an approach for checking time consistency between statecharts and sequence diagrams using schedulability analysis [39].

1.4 Thesis Organization

This thesis is composed of an introduction, five chapters and a conclusion. In Chapter 2, we set the background for this thesis by reviewing the MDA framework, UML and UML profiles while focusing more on UML/SPT. We present in Chapter 3 an extension to UML/SPT to enable the modeling of multicast communications. We define a metamodel encapsulating the main concepts involved in multicast communications, we specify the semantics of the concepts using OCL, and we map these concepts to UML using new stereotypes. In Chapter 4, we investigate the semantics of concurrency in UML/SPT and present a formal specification using timed automata. This allows for using automata-based model checking techniques in order to validate UML/SPT models with respect to the concurrency-related consistency. Chapter 5 is devoted to present our MDA-based approach to bridge the gap between UML/SPT models and well-established schedulability analysis techniques. This

allows to validate the schedulability property of UML/SPT design models. In Chapter 6, we present a framework for the definition of the consistency of a UML/SPT model. In this framework , (1) we build on the established syntactical and behavioral consistency verification approaches; (2) we use our approach to verify the concurrency-related issues using our timed automata semantics of UML/SPT concurrency; (3) we propose an approach based on using schedulability analysis to verify the time consistency of a set of statecharts with respect to time constraints expressed using sequence diagrams annotated with UML/SPT time stereotypes. In Chapter 7, we review the main conclusions of this thesis and outline future work.

Chapter 2

Background: MDA and UML for Real-time Systems

In this thesis, our main research interest is about the modeling and analysis of real-time software systems. The design models of these software systems are expressed using UML profile for real-time systems. The formal verification of these design models using formal analysis techniques require to use models suitable for these analysis. Model-driven approach is used as a general framework for closing the semantic gap between UML real-time design models and those used by formal analysis techniques. The objective of this chapter is to present an overview of the main concepts used throughout this thesis, namely the model-driven development approach, UML, and UML profiles for real-time systems.

2.1 Model Driven Engineering Approach

Model-Driven Development is a software development approach, where models are first-class artifacts. The main idea underlying this apporach is that a software system can be developed starting with a high-level abstract model, which is then successively refined/enriched until eventually, a concrete implementation in a deployment environment is obtained [77].
MDA [78] is a software development framework promoted by the OMG. It is the OMG's incarnation of the MDD approach. The objective of MDA is to enhance the *productivity* of *portable*, *inter-operable*, *maintenable* and *well-documented* software [63]. In order to achieve

these goals, MDA separates the business/application logic from the underlying technology used for its implementation. The key concepts in MDA are, as illustrated in Figure 2.1, a *PIM*, a *PSM*, and *model transformations*:

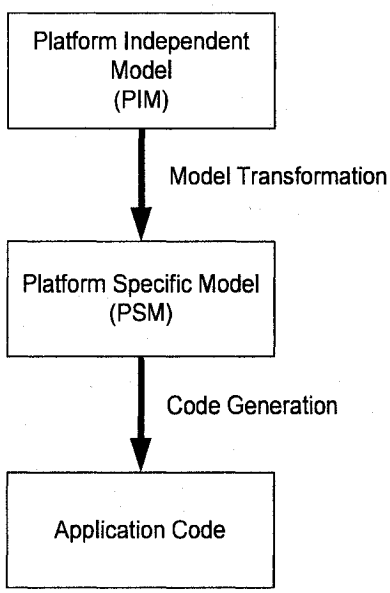

Figure 2.1: Model Driven Architecture

- A PIM is an abstract specification of a software business logic expressed using a modeling language. This is generally UML or one of its profiles.
- A PSM is a more refined model involving details of the implementation platform.
- A model transformation is a mapping of the concepts involved in a PIM into the corresponding ones in a PSM. In addition to the OMG's standardized model transformation language, QVT [88], several other languages have been used, including, OCL [18], XSLT [125], MTrans [73], Scripting languages [102] and ATL [59].

The OMG defines a set of standards to support MDA, including the standard modeling language UML [92], the standard meta-modeling language, Meta Object Facility (MOF) [93], the standard for the serialization and the exchange of models, XML Metadata Interchange (XMI) [89], and the standard for model transformation languages QVT [88]. In the following section, we focus on the UML modeling language and its profile for real-time systems.

The automation in MDA is not limited to the derivation of implementations from high level design models. MDA can also be used as a framework for an automatic generation of models suitable for specific analysis from the design models. In this case MDA is used as a framework for bridging the semantic gap between UML design models and models used in formal analysis. In this thesis, we use MDA for the derivation of task models suitable for the schedulability analysis of UML/SPT models.

2.2 The Unified Modeling Language

UML [92] is nowadays the *de facto* standard software modeling language. UML is used to specify, visualize, construct, and document the artifacts of a software system [110]. Originally, UML is the result of the unification of the main object-oriented development methods, namely the Object Modeling Technique (OMT) [109], Booch method [14], and Object-Oriented Software Engineering (OOSE) method [58]. This brought the version 0.9 of the UML language. UML became an industrial standard once it was adopted by the Object Management Group (OMG) in 1997. This corresponds to the version 1.0 of the language. Within the OMG, UML has then been updated with the main milestones being UML 1.3 [84] and UML 1.4 [85] to eventually reach UML 2.0 [92]. In the following, we focus on the main features of the UML language and highlight the main issues related to UML modeling.

2.2.1 UML Metamodel

UML is an object-oriented graphical modeling language. Considering the advantages of visual formalisms [48], UML modeling is very intuitive. This is probably behind the wide popularity of this modeling language. As a visual language, UML's concrete syntax consists of a set of visual elements including lines, arrows, boxes used to form different kind of diagrams. The abstract syntax of UML is defined by its metamodel.

Generally, a metamodel is a model of a model. The metamodeling approach used to define certain languages such as UML consists in using a part of the language to specify/define the very same language. UML abstract syntax is defined using this approach. UML metamodel

is then a model representing the main concepts of UML and their relationships. This model (i.e. the metamodel) is expressed using the concepts provided by UML class diagrams such as class, associations, multiplicities etc. Figure 2.2 shows a snapshot of the interaction metamodel in UML 2.0. OMG defines a standard language used to express metamodels, including UML metamodel. This standard is the Meta Object Facility [93]. In addition, UML metamodel is complemented with a set of constraints expressed in the Object Constraint Language (OCL) [94]. These constraints form a set of well-formedness rules, which can be used to validate the syntax of UML models.

The semantics of the modeling elements provided by UML is defined informally using English. This leads to many ambiguities and inconsistencies. In addition, several parts in UML are left intentionally open to interpretation. These are formally called *semantic variation points* [30] [117]. This supports the notion of UML as a family of languages [19], which makes it very flexible and widens UML application domains. The lack of formal semantics issue in UML is, however, contributing to the consistency issue in UML models, which is one of the issues considered in this thesis.

2.2.2 Multi-view Modeling Approach

UML supports a multi-view modeling approach. To this end, it provides a multitude of diagrams. UML 2.0 offers the user 13 different diagrams as shown in Figure 2.3. These diagrams cover the different activities in the software development process. Use cases and sequence diagrams can be used, for example, in the analysis stage to model the system requirements while the class diagrams, state machines, activity diagrams can be used in the design stage and deployment diagrams are used in the implementation stage. In addition, these diagrams allow the user to consider different aspects such as its structure, its behavior, and its deployment. This approach is very important in dealing with the increasing complexity of software systems. On the other hand, this approach presents the risk of obtaining inconsistent UML models.

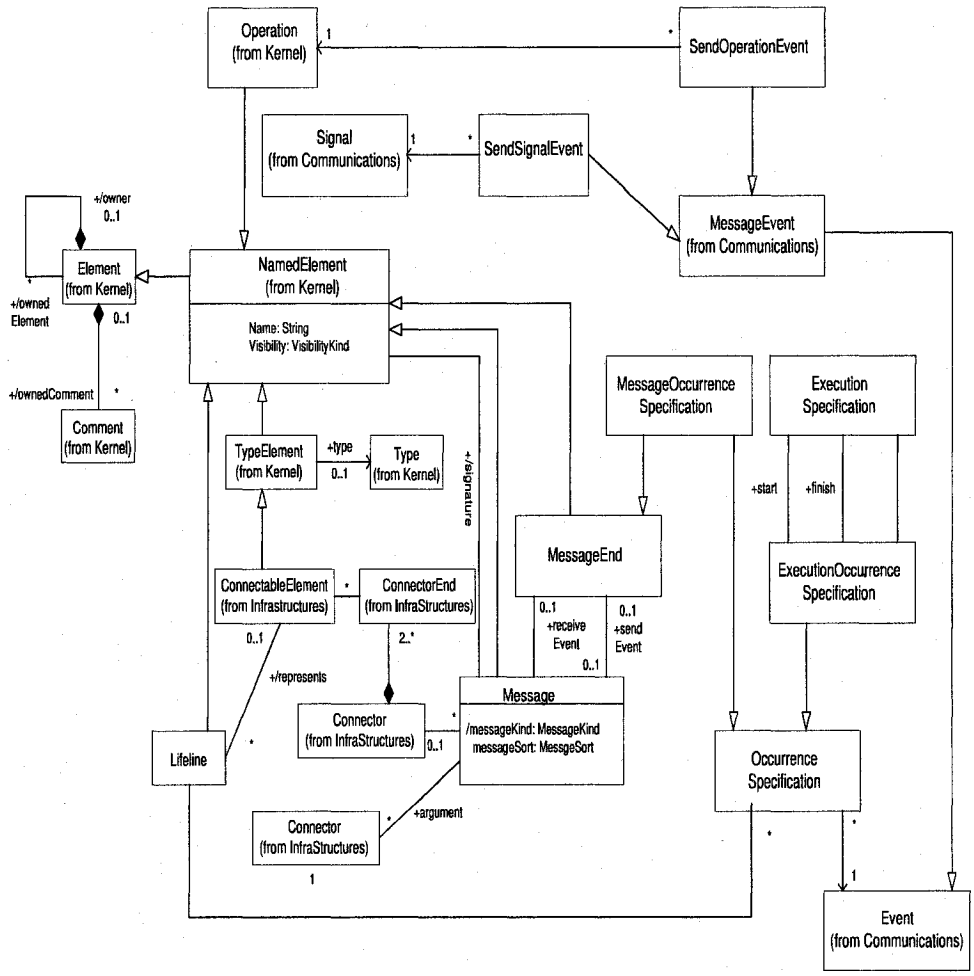

Figure 2.2: UML Interaction Metamodel

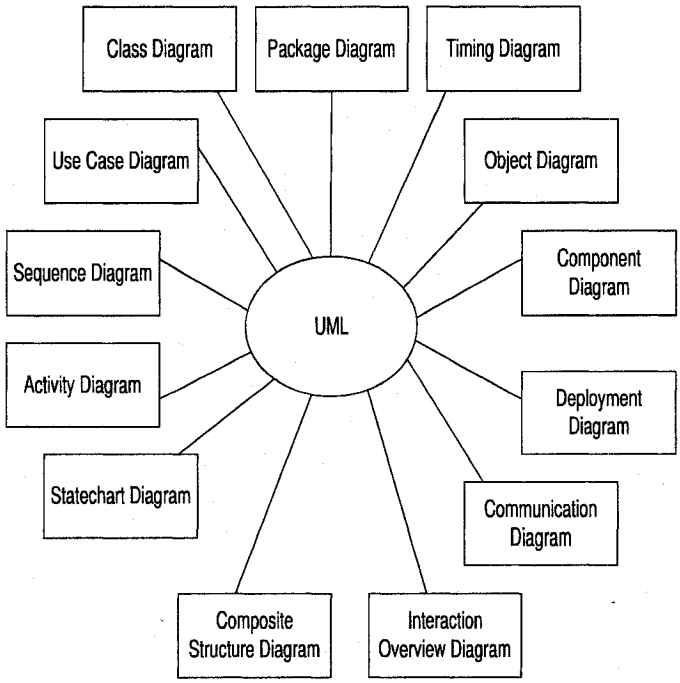

Figure 2.3: UML Diagrams

2.2.3 Extensibility Mechanisms

UML is designed with built-in extensibility mechanisms. These mechanisms allow UML users to attach some domain-specific semantics to existing UML model elements. The extensibility mechanisms in UML are:

- **Stereotype**: A stereotype is basically a new modeling element obtained by specializing an existing model element defined in the metamodel, a metaclass. The stereotype is rendered with its name between a pair of *guillemets*. It can also be represented by a new graphical notation or icon. Figure 2.4 shows the definition of a stereotype ≪task≫, which attaches to the UML metaclass Class the semantics of an operating system task.

- **Tagged values**: These are pairs (tag, value), which represent some specific properties of the newly introduced stereotype. This allows to extend/refine the meta-attributes of the metaclass refined by the stereotype. Figure 2.5 shows the stereotype ≪task≫ with some specific tagged values.

- **Constraints**: These enable to restrict further the semantics of the newly introduced

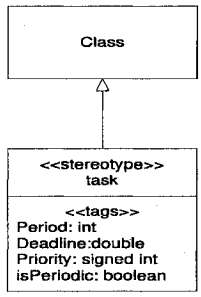

Figure 2.4: Stereotype Definition

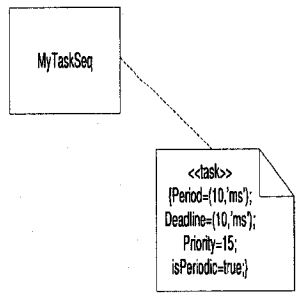

Figure 2.5: Stereotype in Use

stereotype, its tagged values or its relationship with other concepts. The constraints are often expressed using OCL but can also be simple textual constraints.

The extensibility mechanisms in UML can be combined to define an important specializations of (a subset of) UML. Such UML specializations are formally called *UML profiles.*

2.2.4 UML Profiles

A UML profile is a special version of UML tailored to the specifics of a particular domain, like the real-time domain, or a particular activity, like system requirement modeling. There are several UML profiles in the literature that are either results of different research activities or adopted as OMG standards [38]. However, very few are designed with a sound UML profile design method, which makes many of them less valid or of a poor quality.
An approach for defining a UML profile comprises two steps: the definition of a domain model and the mapping of the domain model to UML [118]. The OMG standard UML profile for real-time systems, which is used throughout this thesis and which we will present in the next section, is defined following this approach.

- **The Profile domain model:** This is the metamodel of the profile. It specifies the concepts relevant to the domain, the relationships between these concepts and the constraints that determine the valid models. It includes also a definition of the semantics of the different concepts introduced. This is generally specified in English. The domain model is generally expressed using MOF, where the basic concepts are captured using MOF classes and attributes and the relationships are represented by MOF associations. The constraints are often expressed using OCL.

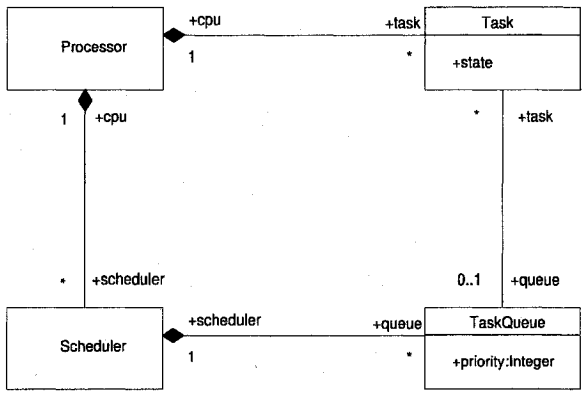

Figure 2.6: Example of a Partial Domain Model of a Profile

- **Mapping of the domain model to UML:** The main concepts introduced in the domain model are mapped to the UML using stereotypes and tagged values. In order to do this, the appropriate UML metaclasses (i.e. having the closest semantics to the considered concept) are determined to be used as base class of the corresponding stereotype.

Example: In order to illustrate this approach, we consider an example of a profile for modeling multi-tasks operating systems [118]. The main concepts involved include for example tasks, processors, priorities, etc. A partial domain model for this profile is shown in Figure 2.6. The Processor concept in this domain model can be mapped to UML using the Node metaclass in UML as base class and can be rendered using the stereotype ≪processor≫.

2.3 The UML Profile for Schedulability, Performance and Time

UML/SPT [91] is a framework for modeling and analysis of real-time software systems. It enables the modeling of resources and quality of service; time concept and time-related mechanisms; and concurrency. In addition, UML/SPT supports schedulability and performance analysis. From the end-user standpoint, UML/SPT is a set of stereotypes and tagged values that can be used to annotate UML design models with quantitative information. This enables the prediction of key properties in the early stages of a software development process using quantitative analysis (schedulability and performance analysis). UML/SPT definition

is based on UML 1.4 [85] and it is currently undergoing a major revamp that will lead to a new UML profile for MARTE [97], which is inline with UML 2.0 [92].

The structure of the UML/SPT profile is shown in Figure 2.7. It consists of a number of

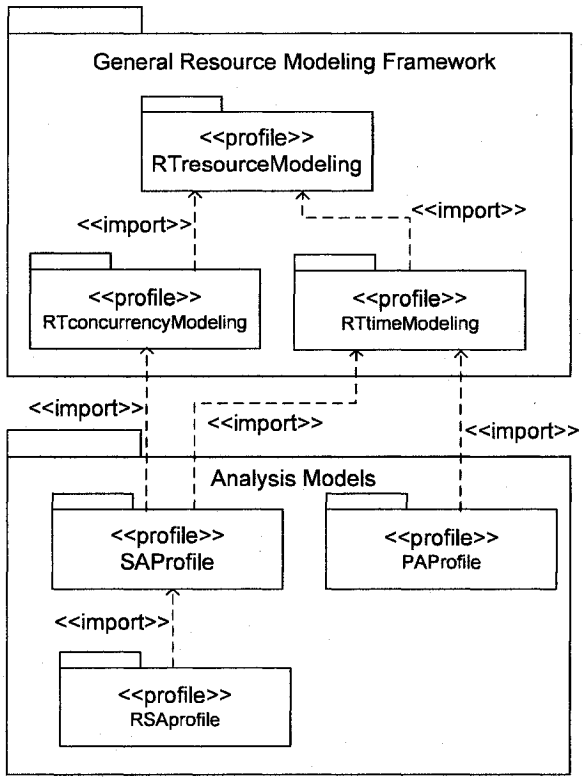

Figure 2.7: The Structure of UML/SPT Profile

sub-profiles. The core of the profile represents *the General Resource Model* framework. This is further partitioned into three sub-profiles: *RTresourceModeling* for the basic concepts of resource and quality of service; *RTconcurrencyModeling* for concurrency modeling; and *RTtimeModeling* for the time concept and time-related mechanisms. Furthermore, UML/SPT is composed of extensible analysis sub-profiles, including: *PAprofile* for performance analysis modeling and *SAprofile* for real-time schedulability analysis modeling. In the following, we are going to focus the main parts of UML/SPT that are used throughout this thesis. We are going to follow the profile definition method in our presentation. For each package, we give and describe the domain model and then its mapping to the UML in terms of stereotypes and tagged values.

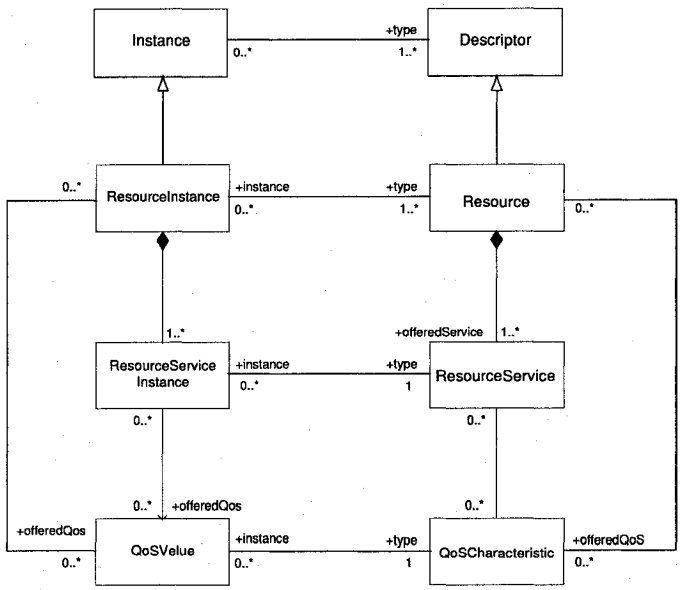

Figure 2.8: General Resource Domain Model

2.3.1 Resource Modeling in UML/SPT

An important property of a resource is that it is finite. For example, software cannot be infinitely fast because it is always limited by the number and the speed of the available CPUs and the amount of information transferred is also limited by the bandwidth. Consequently, in UML/SPT, a resource is modeled as a server providing one or more services to its clients and the Quality Of Service (QoS) attributes are used to capture the physical limitation of a resource [115]. These attributes express either how well a service can be done, the offered QoS, or how well the service should be done, the requested QoS. The basic idea behind quantitative analysis, such as schedulability analysis, is to determine whether the offered QoS meets the requested QoS.

These concepts are captured in the domain model shown in Figure 2.8. In this model a distinction is made between descriptors which are specifications of run-time entities such as Class, Association and Action and instances of these descriptors, which are run-time entities such as Object, Link and ActionExecution. The analysis methods are often instance-based, which means that they operate on models that describe particular situations involving instances.

UML/SPT defines the causality model used as a basis for dynamic modeling. This model

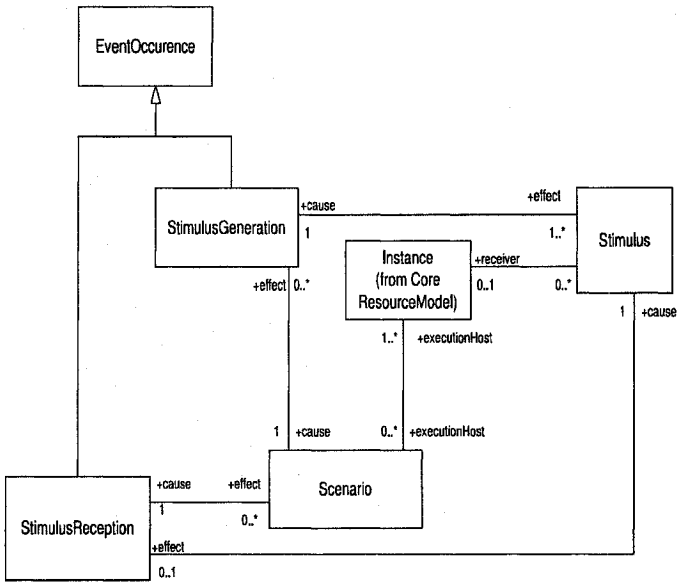

Figure 2.9: Causality Domain Model

captures the cause-effect chain of behaviors of run-time instances. The causality domain model is shown in Figure 2.9.

In order to support model analysis, UML/SPT defines the concept of *analysis context.* This concept captures a particular configuration of the system resources used according to some competing and concurrent scenarios to respond to the workload on the system. It is a starting point for the analysis tools within a design model. The conceptual model of an analysis context is shown in Figure 2.10. This concept is further refined to adapt it either to the schedulability or performance analysis. Figure 5.3 illustrates how the analysis context conceptual model is refined for schedulability analysis and is called a *real-time situation.*

UML/SPT defines the concept of ResourceUsage to capture the details of how the resources are used by the clients in an analysis context. A ResourceUsage can be static or dynamic. In the latter case, it describes a scenario detailing the order in which the clients use the resources, the type of access, the holding time, etc. Figure 2.11 shows the dynamic usage domain model.

Finally, UML/SPT allows the modeling of different kinds of resources. The general taxonomy of resources defined is shown in Figure 2.12.

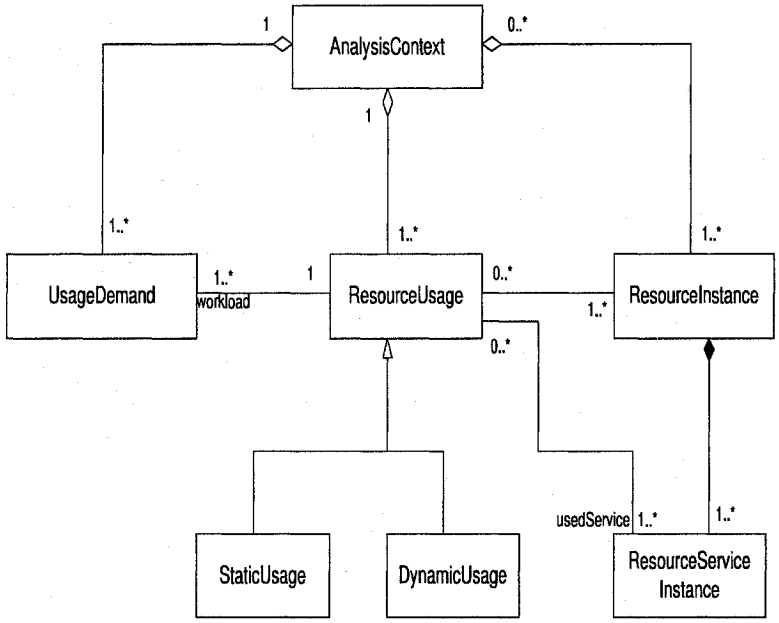

Figure 2.10: Model of Analysis Domain Model

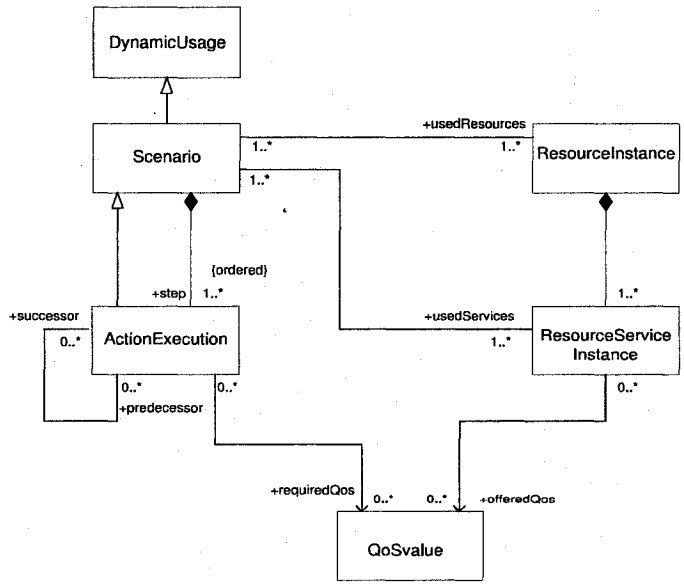

Figure 2.11: Dynamic Usage Domain Model

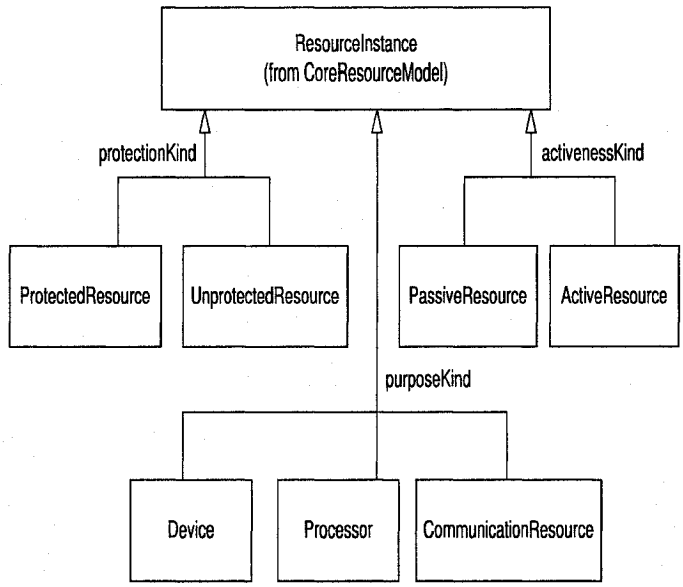

Figure 2.12: Resource Taxonomy Domain Model

2.3.2 Time Modeling in UML/SPT

The time domain model supported in UML/SPT is shown in Figure 2.13. In this model time is abstractly captured as a partial order on events. Physical time is a continuous and unbound progression of physical time instants. Using a periodic clock, time progress is measured by counting the number of expired cycles. The count associated with a particular instant is its measurement. TimeValue class represents a time measurement, which can be represented using the Integer data type for a discrete time model or the Real data type for a dense time model. TimeInterval class models a duration, which is the expired time between two instants. Since a duration is also a time value, TimeInterval is a subclass of TimeValue. Moreover, UML/SPT defines also a domain model for the time mechanisms, including clocks and timers. This is shown in Figure 2.14.

UML/SPT maps these domain models to UML using stereotypes. It provides a set of stereotypes and associated tagged values that the modeler could apply to UML models to specify time values ≪`RTtime`≫, time-related mechanisms such as ≪`RTclock`≫, ≪`RTtimer`≫, ≪`RTtimeout`≫ or timing constraints ≪`RTdelay`≫, ≪`RTintervale`≫. For example, Figure 2.15 illustrates a UML sequence diagram modeling the behavior of an elevator control system in reaction to the arrival sensor event. This sequence diagram is annotated with

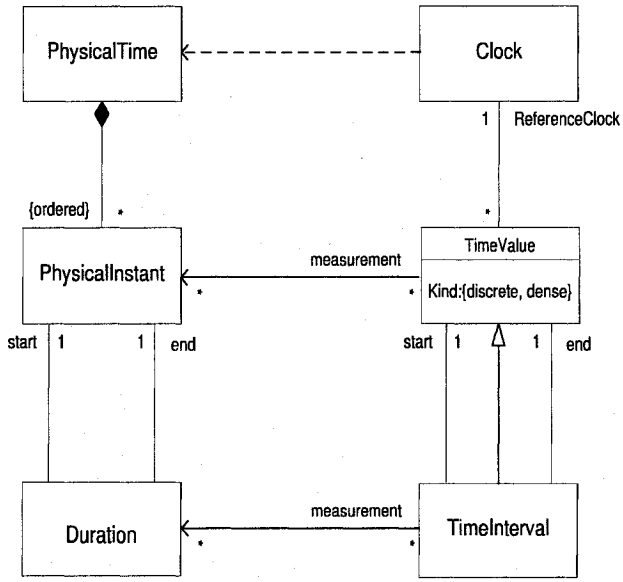

Figure 2.13: Time Modeling in UML/SPT

UML/SPT stereotypes to model the periodicity of the event, the time constraints on the actions of the different components of the system, etc.

2.3.3 Concurrency Modeling in UML/SPT

The package *RTconcurrencyModeling* encapsulates a domain model for concurrency modeling. This model is expressed using the class diagram shown in Figure 2.16. The main entity in the concurrency domain model is a *Concurrent Unit.* It is defined as an active resource instance that executes concurrently with other concurrent units. An *active* resource is defined in *RTresourceModeling* package as an entity capable of generating stimuli without being prompted by an explicit stimulus as opposed to a *passive* resource, which does not generate its own behavior but reacts to stimulus [91]. As shown in Figure 2.16, a concurrent unit owns one or more service resource instances through a composition relationship. It also owns one or more stimulus queue to hold stimuli that arrive while the concurrent unit is busy. The response to such stimulus is deferred until the concurrent unit is ready. In addition, each concurrent unit is associated with a *main scenario.* During this scenario, the concurrent unit may execute either an explicit *receive action*, a *synchronous invoke* or an *asynchronous invoke.* The execution of a receive action triggers the appropriate method in the corresponding service instance. Moreover, a service request has a property called

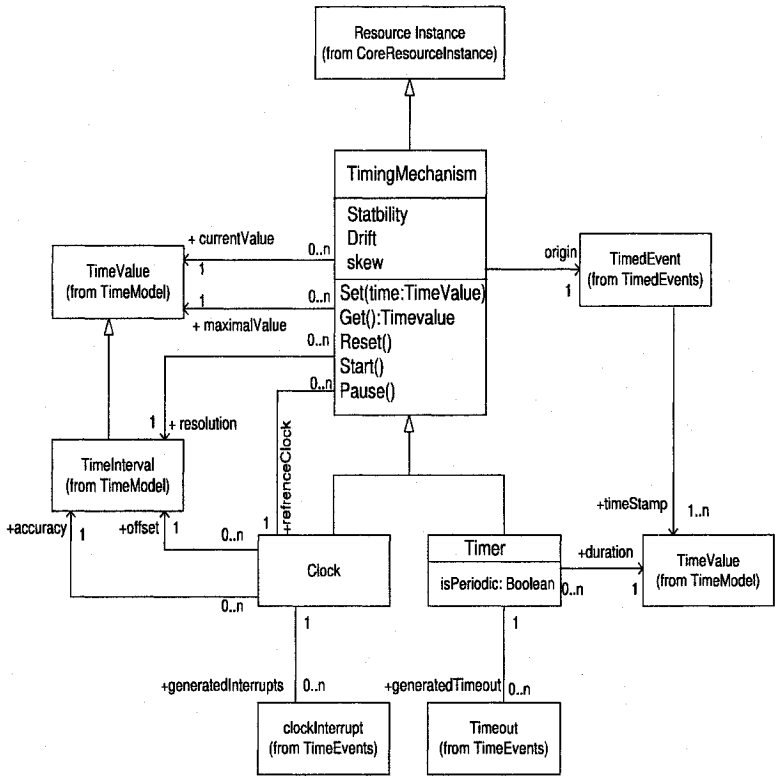

Figure 2.14: UML/SPT Timing Mechanisms Domain Model

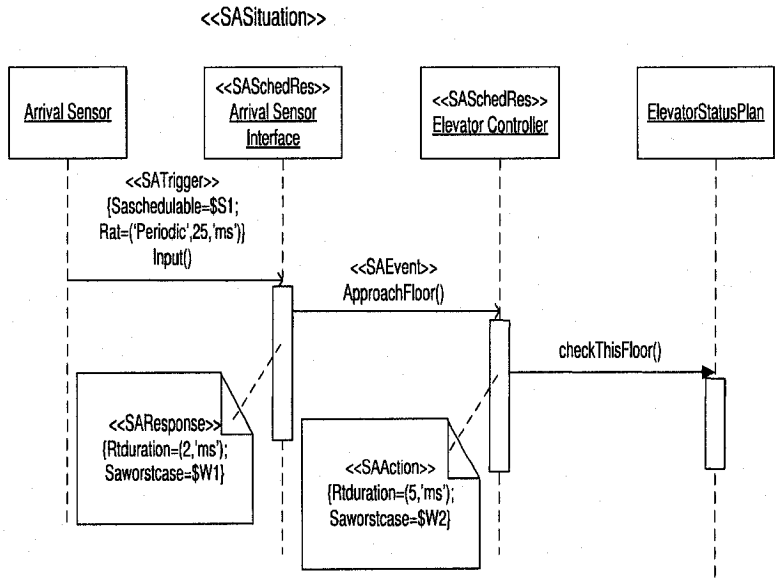

Figure 2.15: Sequence Diagram Annotated with UML/SPT Stereotypes

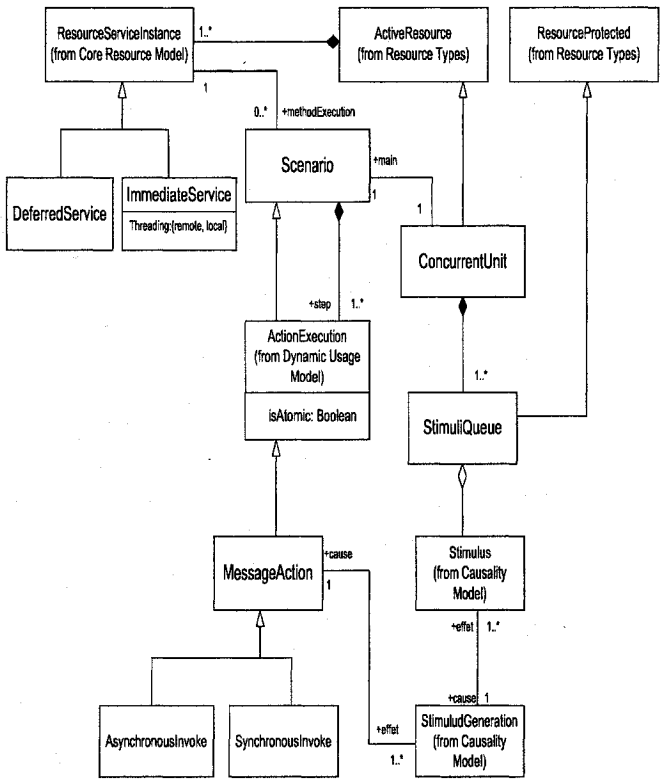

Figure 2.16: UML/SPT Concurrency Domain Model

threading, which may be local or remote. In the former case, the receiving instance spawns a local thread of execution to handle the request while in the later case, it assumes the availability of an existing thread.

2.3.4 Schedulability Analysis Modeling in UML/SPT

The main concepts involved in schedulability analysis modeling are encapsulated in the SAProfile domain model as depicted in Figure 2.17. The *RealTimeSituation* concept represents a specific analysis context. It is a specific configuration of resources including *ExecutionEngine* to model processors, *SResource* to model passive resources and *SchedulableResource* to model threads or tasks; and different entities, *scheduling jobs*, contending for these resources. A *scheduling job* is composed of a *Trigger* modeling an external event having an arrival pattern that could be periodic for instance and a *Response*. The latter is the root action of a sequence of actions, *SAction*, separately schedulable. It is worth mentioning that *SAction* is also a nested construct as, according to the *Dynamic Usage Model Package* of

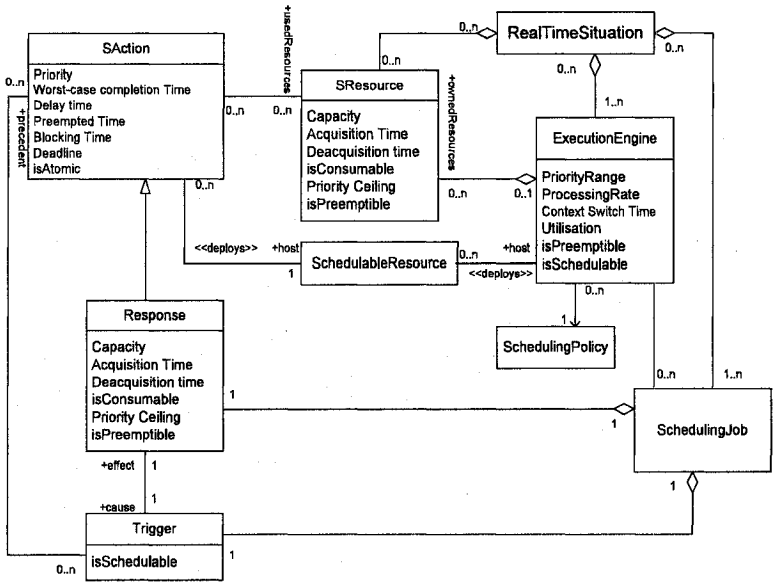

Figure 2.17: UML/SPT Schedulability Analysis Domain Model

Steriotype	Real-time Concept	UML Model Element
≪SASituation≫	Real-time situation	Collaboration, Sequence diagrams
≪SATrigger≫	Event	Message, Stimilus
≪SAResponse≫	Response	Method, Action
≪SAAction≫	Action	Method, Stimilus, Action
≪SASchedulable≫	Task, Thread	Instance, Object, Node
≪SAResource≫	Resource	Instance, Class, Node
≪SAEngine≫	CPU, Processsor	Object, Class, Node

Table 2.1: UML/SPT Common Stereotypes for Schedulability Analysis

the *General Resource Model Package* of UML/SPT, *SAction* is a subclass of the metaclass *Scenario.*

Correspondingly to these concepts, a set of stereotypes and their associated tagged values is defined in UML/SPT. A sample of these is presented in the Table 2.1. The application of these stereotypes on a collaboration diagram is illustrated in Figure 2.18.

2.4 UML Profile for MARTE

The OMG has issued an RFP in order to replace UML/SPT with a new profile in line with UML 2.0 [97]. The new UML profile is called UML profile for Modeling and Analysis of Real-Time and Embedded Systems. The beta adopted specification [90] is now made available

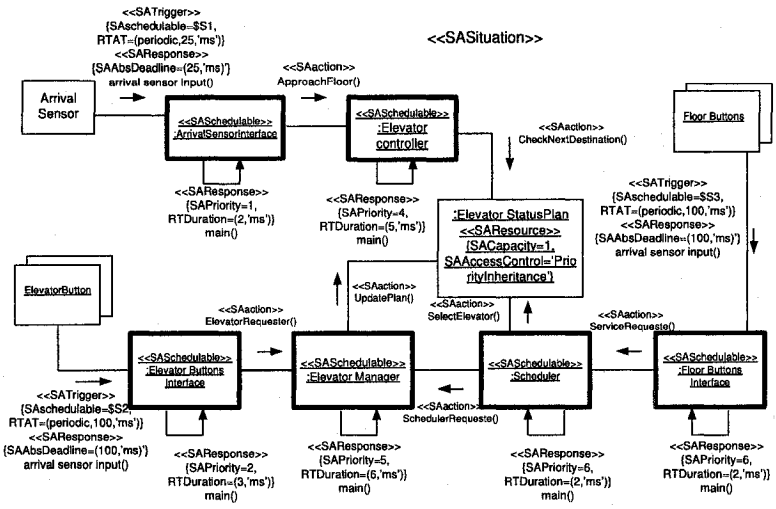

Figure 2.18: UML/SPT Schedulability Analysis Model

for the public to submit issues related to the specification for the MARTE Finalization Task Force. The structure of MARTE is shown in Figure 2.19. The profile is designed (1) to model the features of real-time and embedded systems (the MARTE design model package) and (2) to annotate the models in order to support the analysis of systems properties (MARTE analysis package). These two packages refine the MARTE foundations package, which includes:

- The Non-Functional Properties (NFPs) package provides a general framework for annotating UML profiles with NFPs.
- The Time package defines a general framework for representing time and time-related concepts and mechanisms that are relevant for modeling real-time and embedded systems.
- The Generic Resource Modeling (GRM) package defines the concepts necessary to model a general platform for executing real-time embedded applications.
- The Generic Component Model (GCM) defines concepts necessary to address the modeling of artifacts in the context of real-time and embedded systems component based approaches.

MARTE design model package is composed of the RTE Model of Computation and Communication (RTEMoCC) package, which provides high-level concepts to model the real-time

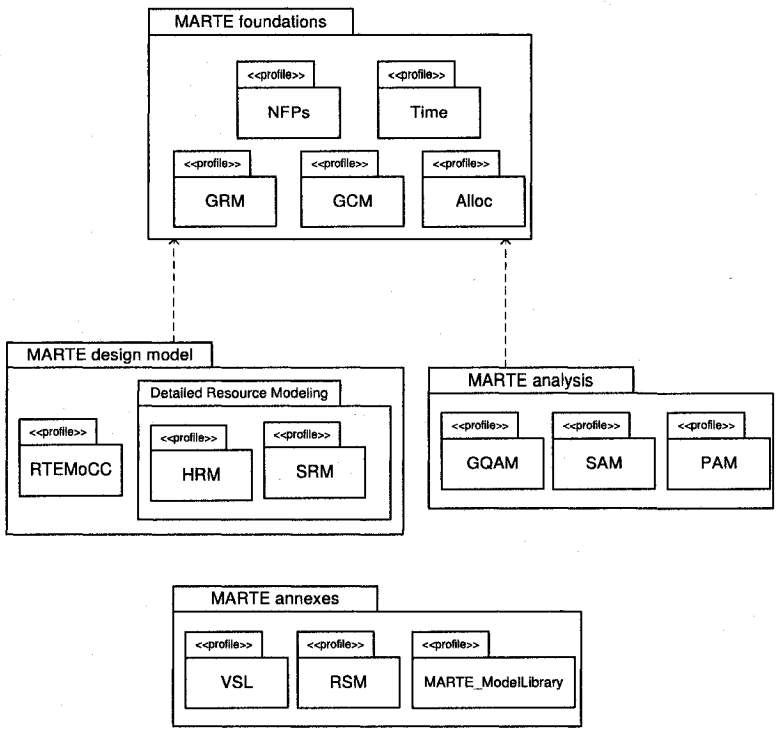

Figure 2.19: The Structure of UML Profile for MARTE

and embedded systems features, and the Detailed Resource Modeling (DRM), which provides specific modeling artifacts for the description of both software and hardware execution supports. MARTE analysis package defines a framework for generic quantitative analysis and specifically supports the modeling of schedulability (SAM) and performance (PAM) analysis.

2.5 Other UML Profiles for Real-time Systems

In this section, we give a brief overview of the other UML profiles for real-time systems in the literature.

2.5.1 UML Profile for Quality of Service

The UML profile for modeling *Quality of Service and Fault Tolerance Characteristics and Mechanisms* (UML/QoS) is an OMG standard UML profile [87]. This profile aims at capturing the concept of quality of service in general. It allows the definition of an open variety of quality of service requirements and properties [24].

UML/QoS is relevant for real-time software modeling because it is defined to complement UML/SPT. However, while UML/SPT is tailored to fit performance and schedulability analysis, UML/QoS allows the designer to define any set of quality of service requirements and carry out a corresponding analysis that could be relevant for the safety-critical aspect of real-time software. This was shown in [10], where a quality model has been defined to drive a dependability and performability analysis of an embedded automation system. Similarly to UML/SPT, UML/QoS can be used to annotate UML diagrams. In contrast to UML/SPT, UML/QoS proposes a procedure that consists of three steps:

- *Definition of QoS characteristics:* The new user-defined QoS characteristics could leverage, through specialization, the general QoS characteristics catalogue defined in this profile. This catalogue comprises the following categories: `Performance`, `Dependability`, `Security`, `Integrity`, `Coherence`, `Throughput`, `Latency`, `Efficiency`, `Demand`, `Reliability` and `Availability`. In particular, the Latency QoS category can be used in the real-time context. The QoS characteristics are templates classes having parameters. The later have to be instantiated in the next step.

- *Definition of the quality model:* The QoS characteristics parameters should be assigned actual values. This is done through the definition of quality characteristics bound classes and template bindings. The UML model containing the binding information and the bound classes is called the Quality Model.

- The last step is the UML models annotation using quality of service requirements.

2.5.2 UML-RT Profile

UML-RT is a UML real-time profile developed by Rational Software [116]. It uses the UML built-in extensibility mechanisms to capture the concepts defined in the Real-time Object Oriented Modeling (ROOM) methodology [114]. UML-RT allows the designer to produce models of complex, event-driven and possibly distributed real-time systems. However, it does not support time constraints modeling. UML-RT is supported by a CASE tool called RationalRT that allows for automatic code generation by compiling the models and linking

them with a run-time system. UML-RT includes constructs to model the structure and the behavior of real-time systems:

- **Structure Modeling:** UML-RT provides the designer with entities called capsules, which are communicating active objects. The capsules interact by sending and receiving messages through interfaces called ports. Furthermore, a capsule may have an internal structure composed of other communicating capsules and so on. This hierarchical decomposition allows for modeling complex systems.

- **Behavior Modeling:** The behavior is modeled by an extended finite state machine, and it is visualized using UML state diagrams. These state machines are hierarchical since a state could be decomposed into other finite state machines. A message reception triggers a transition in the state machine. Actions may be associated with transitions or the entry to and/or the exit from a state.

2.5.3 TURTLE Profile

TURTLE stands for Timed UML and RT-LOTOS Environment. It is a UML profile for the formal validation of complex real-time systems [4]. TURTLE uses UML's extensibility mechanisms to enhance UML structuring and behavioral expressive power. It has a strong formal foundations. TURTLE extensions semantics is expressed by a mapping to RT-LOTOS. This enables a formal validation as well as a simulation of the UML models. TURTLE essentially allows the description of the structure/archtecture as well as the behavior of the system using an extension of the UML class/object and activity diagrams. The main extensions brought by TURTLE are the following:

- **Structural Extensions:** TURTLE introduces the concept of TClass, which has special attributes called Gates. These are used by TClass instances, TInstances, to communicate and are specialized into InGate and OutGate. In addition, TURTLE introduces stereotypes called composition operators. These are used to explicitly express parallelism, synchronization, and sequence relationships between TClasses.

- **Behavioral Extensions:** The behavior of a TClass is expressed using activity diagrams extended with logical and temporal operators. These operators allow express-

ing synchronization on gates with data exchange. In addition, TURTLE enables the expression of temporal non-determinism and different sorts of delays (deterministic, nondeterministic).

TURTLE is supported by a toolkit composed of TTool [123] and RTL [108]. These are used by the designer to build a design, to run the simulation and to perform a reachability analysis for the validation of the system.

Finally, TURTLE was extended to fit the requirements of distributed and critical systems. The objective is to enable defining components and their deployment and to study their properties at early stages of the software development process. This is done using a formal definition of the deployment diagrams, which are the most suitable for distributed architecture description. Therefore, TURTLE has been extended to take into account deployment diagrams. The obtained profile is called TURTLE-P [5], which addresses the concrete description of communication architectures. TURTLE-P allows the formal validation of the components and deployment diagrams through its foundations in RT-LOTOS.

2.5.4 SDL Combined With UML

This is the title of the ITU-T recommendation Z.109 [56] [80]. It is a UML profile for SDL since it defines a specialization of a subset of UML and a one-to-one mapping to a subset of SDL. Thus, Z.109 has SDL as a formal semantics. This profile provides the designer with a combination of UML and SDL. Essentially, Z.109 defines a UML model for the main concepts of SDL, the domain model, and offers a corresponding set of stereotypes. In the following, we highlight the main concepts defined in Z.109:

- **Agent** An SDL system is composed of agents connected through channels. Each agent has a state machine and an internal structure composed hierarchically of other agents. In addition, an agent can be a process, a bloc or a system. In particular, an agent type is mapped into a class of active objects and its kind is stereotyped ≪system≫, ≪block≫ or ≪process≫.

- **Gates and Interface:** The agents communicate through gates by sending signals or requesting a procedure, which together, the signals and procedures, compose its

interface. The latter is mapped into a UML interface and the former are stereotyped ≪signal≫ and ≪procedure≫.

- **State Machine:** An SDL agent state machine is mapped to a UML state machine.
- **Package:** UML packages are used to represent SDL packages.

Finally, this profile has been implemented in the Telelogic CASE tool Telelogic TAU 3.5 [122].

2.5.5 The OMEGA UML Profile

This profile [44] is part of the OMEGA project [120]. It is a framework for UML-based real-time modeling. It enables the analysis and verification of time and scheduling aspects. It provides a set of timed-events primitives and the semantics of these primitives is formally expressed using timed automata with urgency.

2.5.6 OCL Profile

This profile is based on an extension of OCL 2.0 metamodel [29]. It allows for the specification of real-time constraints using OCL. The semantics of this profile is given by a mapping to time-annotated temporal logic formulae expressed in CTL. This enables formal verification of different system properties.

2.6 UML Profile for System Engineering

UML proved to be a very successful modeling language for software-intensive systems. It has very interesting features, which appeal to the system engineering community. However, it falls short to cover the needs of a wide range of engineering applications such as automotive, aerospace, communication in addition to information systems. This led the International Council on Systems Engineering (INCOSE) [54] to define the UML profile for system engineering. It is defined as a modeling language for system engineering applications and used to support the specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes,

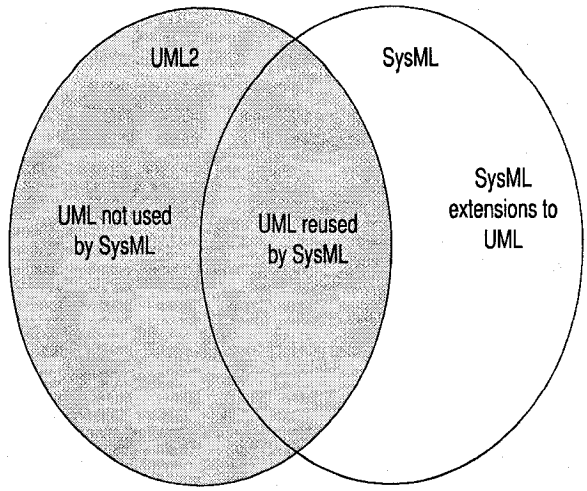

Figure 2.20: UML and SysML Relationship

personnel, and facilities [95].

SysML reuses a subset of UML 2.1 and extends it by additional diagrams and new concepts as it is shown in Figure 2.20. In the following, we summarize the main concepts introduced by SysML to enable system engineering modeling.

- SysML diagram taxonomy is shown in Figure 2.21. In system engineering, requirement modeling is of paramount importance. UML does not allow to represent the trace of the informal requirements specification to the system design elements. Generally UML Use Cases are used to understand the expected system functionalities but the requirements are traced to the use cases and not to the design. With this regard, SysML brings a major enhancement through the requirement diagrams. These allow to represent the requirements and many relationships among them as well as their relationship with the system architecture and design elements.

- SysML improves also UML structural modeling capabilities. It defines the concept of blocks and defines two new kind of diagrams to model a system structure: The Block Definition Diagram and the Internal Block Diagram. The former defines the features of a block in terms of properties/operations and the relationships between blocks such as associations, generalizations and dependencies. The latter captures the internal structure of a block in terms of properties and connectors between properties.

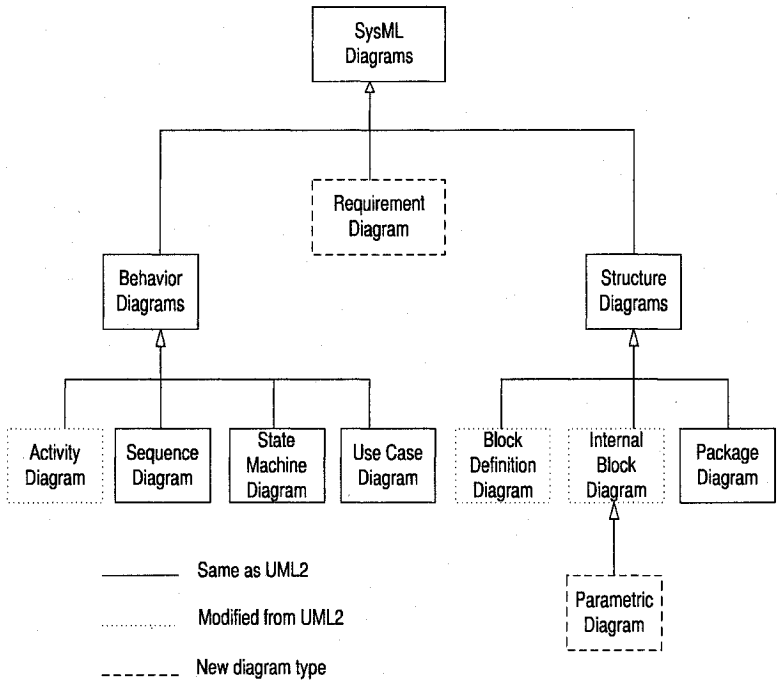

Figure 2.21: Taxonomy of SysML diagrams

- SysML introduces the concept of allocation as a more abstract form of a deployment in UML. It is a design-time relationship between model elements such as linking the requirements and design elements.

- Another important feature of SysML is the introduction of additional models of computation. It extends the behavior of UML activity diagrams so that the control of execution of a running action can be disabled. It is also extended to enable the modeling of continuous and probabilistic systems. Finally, SysML supports also the modeling of energy and material flows.

2.7 UML Profile for Systems-On-Chip

System-On-Chip refers to the integration of computing and communications components into a single chip. SoCs are incorporating more processors and software. The success of UML in the software community has led to a surge of interest in using UML in the SoC design flow [75]. In this context, the focus is put on how to customize UML so that it can be used as a System Level modeling language in the SoC design flow. Therefore,

many of these customizations are UML profiles for SystemC [55], which is a C++ based system-level language used in SoC design flows [81]. The OMG standardized profile for SoC [96] is defined to support the modeling and specification of SoC designs. In particular, it introduces the SoC structure diagrams. In additon, the profile defines a set of stereotypes to represent modules, connectors, channels, ports, clocks, processors, protocols, data types, and controllers.

2.8 Conclusions

We have studied the main UML profiles for real-time and drawn a comparison between them using some criteria, including formal foundation, expressiveness and tool support [38]. This comparison is summarized in Table 2.2. In the following, we discuss the salient points of this comparison and focus on the research issues related to UML/SPT.

- As for the formal foundation, we can distinguish two groups of profiles. The first group of profiles presents some formal semantics such as RT-LOTOS for TURTLE and SDL for Z.109. The second group includes UML/SPT, UML/QoS and UML-RT. These profiles lack formal foundations. In particular, UML/SPT is defined using the metamodeling approach, where the semantics of the main concepts is given informally in English.

- With regards to the profiles expressiveness, we notice that OMG's profiles, namely UML/SPT and UML/QoS, introduce models for the concepts of time, resource and quality of service characteristics respectively using stereotypes to apply annotations to UML design models while the others; UML-RT, TURTLE and Z.109; do not express explicitly time constraints but introduce new modeling elements and constructs such as capsules in UML-RT, TClass in TURTLE to build the model itself.

- The objective of UML profiles for real-time is to use UML in an integrated framework for the design, analysis and synthesis of real-time software. While several tools are available to support some of the aforementioned profiles, a full integrated framework is however still lacking. This is particularly true for the OMG standard UML/SPT,

whose implementation is very limited.

- UML/SPT is a modeling language and not a methodology. The standard does not specify how to use the profile.
- UML/SPT uses also a multi-view modeling approach, where different UML diagrams are used to capture different perspectives of the system. In addition, aspects relevant to real-time modeling such as time, concurrency and schedulability through appropriate stereotypes cross-cut through the UML design model. This contributes further to the consistency issue of such models.
- Finally, there is a semantic gap between design models using UML/SPT and models suitable for quantitative analysis such as schedulability. This gap needs to be bridged in order to use such analysis for the validation of UML/SPT design models.

	Issuer	Tool Support	System	Analysis	Expressiveness	Formal Semantics
UML/SPT	Industry (OMG)	Rhapsody (Telelogic) RT Studio (Artisan)	Generic and real-time systems	Peformance Schedulability	Time, Resource Concurrency	No
UML/QOS	Industry (OMG)	None	generic and real-time systems	User-defined	QoS Characteristic	No
UML-RT	Industry IBM/Rational	RationalRT	event-driven real-time systems	Schedulability	Capsules, ports Async messages	No
TURTLE	Academia	TTool RTL	Distributed Critical Systems	No	Synch, Parallele delays operators	RT-LOTOS
Z.109	Industry ITU-T	Telelogic Tau 3.5	Telecommunication Critical Systems	No	SDL concepts	SDL
OMEGA-RT	Academia	OMEGA tool set	Real-time Systems	Timing Scheduling	Timed events	Timed automata with urgency
OCL Profile	Academia	None	generic RT Systems	Verification	Real-time constraints	CTL

Table 2.2: UML Profiles for Real-Time Systems

Chapter 3

An UML/SPT Extension for Multicast Communications

Expressive, flexible, and customizable specification/modeling languages are necessary to capture complex behavioral requirements of distributed real-time systems. Message Sequence Charts (MSC) is a well-established specification language for high-level behavioral requirements modeling. It is widely used for telecommunications software systems. MSC has evolved through successive versions to enable the expression of time constraints, object orientation, data, and scenario composition [57]. However, the extensibility of MSC is hindered by a lengthy standardization process. This has led to ad-hoc extensions (e.g., [66], [128], [130]) that need to go through the standardization process before being accepted and effective. On the other hand, UML/SPT is designed using UML built-in extensibility mechanisms to capture the concepts necessary for the modeling of resource, concurrency and time. Moreover, UML/SPT inherits this extensibility, which allows for simple and natural extensions.

In this chapter, we present an UML/SPT extension, which enables the modeling of multicast communications [36]. Such extension is required to model multicast protocols such as RMTP2 [100]. Our main goal while researching this extension was to demonstrate the easiness of extending UML/SPT in comparison to languages such as MSC. The main contribution is an extension of UML/SPT for the modeling of multicast communications.

In order to achieve this, we extended UML/SPT with a domain model to capture the main concepts involved in multicast communications. As for the semantics of the introduced domain model, we give a declarative definition of the concepts required for the extension using OCL constraints. We mapped the domain model into the UML using new stereotypes. As a case study, we model the main requirements of the RMTP2 protocol using the extended UML/SPT. Finally, we compare our extension and the RMTP2 modeling exercise with the ones conducted in [51] using MSC.

3.1 Multicast Communication Extension for UML/SPT

We describe the multicast communication extension for UML/SPT using the profile definition approach [118], outlined in Chapter 2.

3.1.1 The extension Domain Model

The extension presented here allows to model *multicast communications*, which are important for communication protocols such as the protocol RMTP2. In order to do so, we present a metamodel capturing the main concepts in multicast communications. The package encapsulating this metamodel and its relationship with the structure of UML/SPT profile are illustrated in Figure 3.1.

A *multicasting resource* is a specialization of the metaclass *ActiveResource*. The latter is defined in UML/SPT *resource type* metamodel, illustrated in Figure 2.12, as an autonomous and concurrent entity able to generate stimuli independently. The multicasting resource is composed of a *dynamic group* of instances. An instance is defined in the *core resource model* of UML/SPT. Each message targeting a multicasting resource is implicitly forwarded to all the group members by the multicasting resource. The configuration of this group is dynamic where the instances could join and leave at will using *Joingroup* and *LeaveGroup* actions, derived from *ActionExecution*. The latter is defined in the *dynamic usage model* of UML/SPT. Figure 3.2 illustrates this metamodel extension and how it is linked to UML/SPT metamodel.

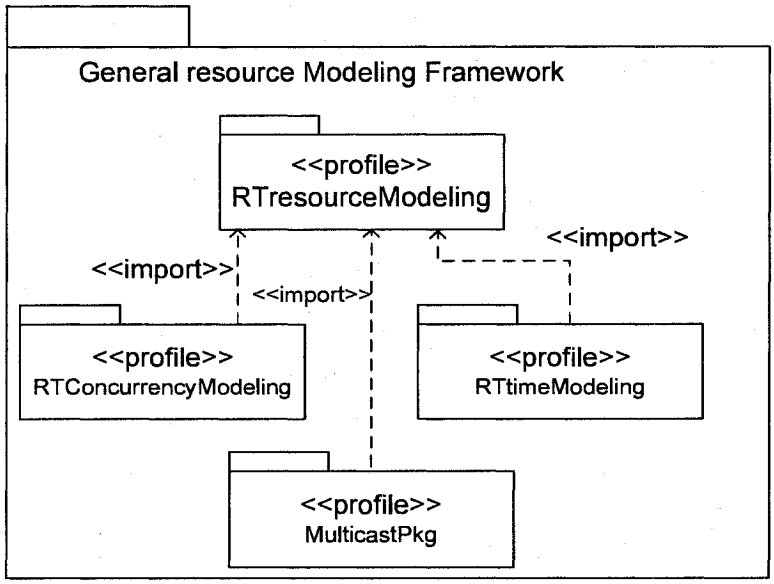

Figure 3.1: Multicast Extension Package

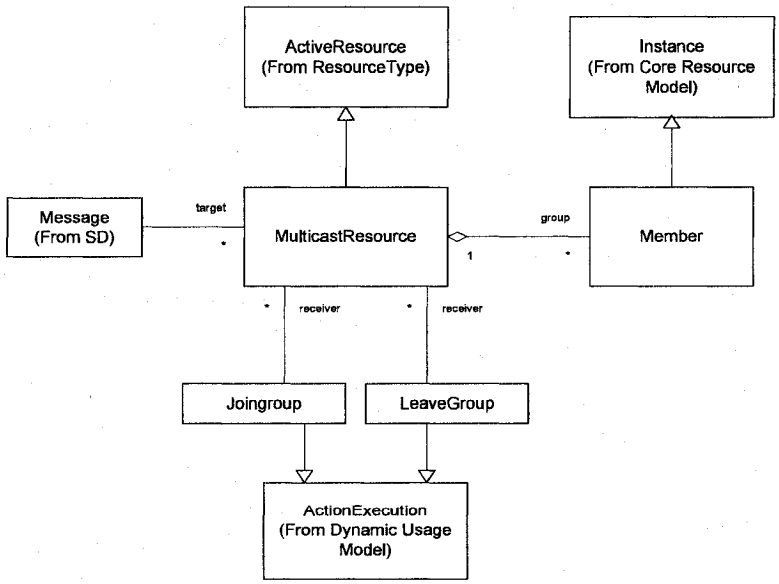

Figure 3.2: Multicast Extension Metamodel

package MulticastPkg
context Joingroup
pre: $self.receiver.group- > \textbf{select}(i : Instance|$
$i = self.executionHost)- > isEmpty$
post: $self.receiver.group- > \textbf{select}(i : Instance|$
$i = self.executionHost)- > isNotEmpty$

Table 3.1: OCL Specification of Joingroup

package MulticastPkg
context Leavegroup
pre: $self.receiver.group- > \textbf{select}(i : Instance|$
$i = self.executionHost)- > isNotEmpty$
post: $self.receiver.group- > \textbf{select}(i : Instance|$
$i = self.executionHost)- > isEmpty$

Table 3.2: OCL Specification of Leavegroup

3.1.2 Domain Model Semantics

We give a declarative specification for the main metaclasses introduced in our UML/SPT extension using OCL [94]. The OCL constraints in Table 3.1 and Table 3.2 define respectively the metaclasses *Joingroup* and *Leavegroup* representing respectively the actions of joining and leaving a multicasting resource.

The expression in Table 3.1 ensures that the instance joining a multicasting resource does not belong to it before the action is executed and that it does after the action is executed. Reciprocally, the expression in Table 3.2, specifies that an instance does no more belong to a multicasting resource after the execution of the action of leaving it. In these two OCL expressions, we use *executionHost*, which is the role of the instance in its association with the *executionAction* as defined in the *Causality Model Package* of UML/SPT shown in Figure 2.9, to identify the instance executing the action.

A message sent to a multicasting resource is expressed by a *metamodel transformation* whose platform independent contract [18] is specified using OCL as illustrated in Table 3.3. This expression specifies that for each message targeting a multicasting instance, a message

package MulticastPkg
context SD::Message
let $mcast : MulticastResource$
pre: $self$.**allInstances**$->$ **forAll**($m : SD :: Message|$
$mcast = m.receiveEvent.covered.represents.$
$OclAsKind(MulticastResource)$ **in:**
$mcast.group->notEmpty$)
post:
$mcast@pre.group->$ **forAll**($gm : Member|$
$SD :: Message$.**allInstance**$->$ **exists**($m1 : SD :: Message$
$(m@pre.signature = m1.signature)$ **and**
$(m1.sendEvent.Covered.represents = mcast@pre)$ **and**
$(m1.receiveEvent.Covered.represents = gm)))$

Table 3.3: OCL Specification of Multicast Message Sending

Stereotype	UML Model Element
≪Multicast≫	Object
≪Joingroup≫	Message, Stimilus
≪Leavegroup≫	Message, Stimilus

Table 3.4: Multicast Communication Extension Stereotypes

having the same signature is sent to all the instances member of this multicasting resource. In this OCL expression, we use UML modeling elements related to message exchange such as *message*, *sendEvent*, *receiveEvent*, and *lifeline*. These are specified in the UML sequence diagram metamodel [92].

3.1.3 Multicast Extension Stereotypes

We introduce three new stereotypes as illustrated in Table 3.4. They correspond to the main concepts introduced in our extension, and that are used to annotate UML models to express multicast communication requirements such as joining a multicast group, leaving a multicast group, and/or sending a message to a multicast group.

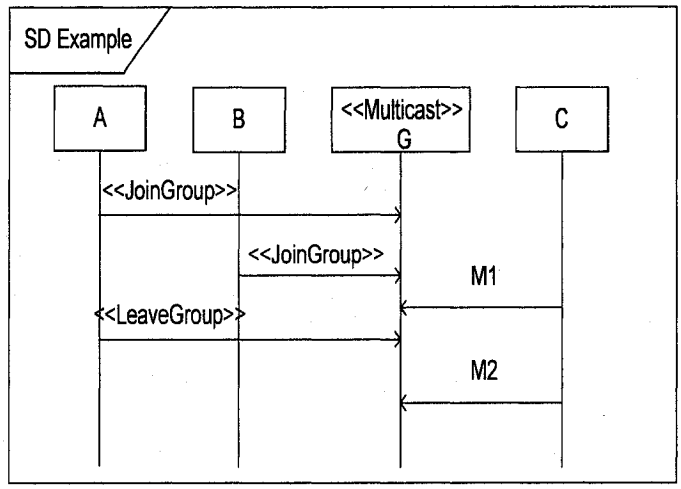

Figure 3.3: UML/SPT Multicast Extension Example

Example

The example illustrated in Figure 3.3 shows a UML sequence diagram annotated using these stereotypes. According to the semantics of our extension, the message *M1* will be received by both *A* and *B* since they both joined the multicast group, but *M2* will be received by B only because A has left the multicast group.

3.2 Application: RMTP2 Behavioral Requirement Modeling

The main features of RMTP2 [79], [100] are guaranteed reliability, high throughput, and low end-to-end delay regardless of the underlying network. RMTP2's reliability is achieved through acknowledgments, but the network congestion that would be caused by a growing number of direct ACKs is avoided using a tree-based organization of the network.

The *sender node*, the top of the *global multicast tree* that spans all the receivers, multicasts the data on the data channel. The receivers are grouped into local regions with a special *control node*. The control node could be: (1) an *aggregate node* which maintains the receivers membership, and aggregates the acknowledgements from the receivers to the sender and forwards missing packets; (2) a *designated receiver* node which keeps a copy of the data and retransmits it to the subtree below. Eventually, the acknowledgments are aggregated at the top level control node, which retransmits them to the sender node.

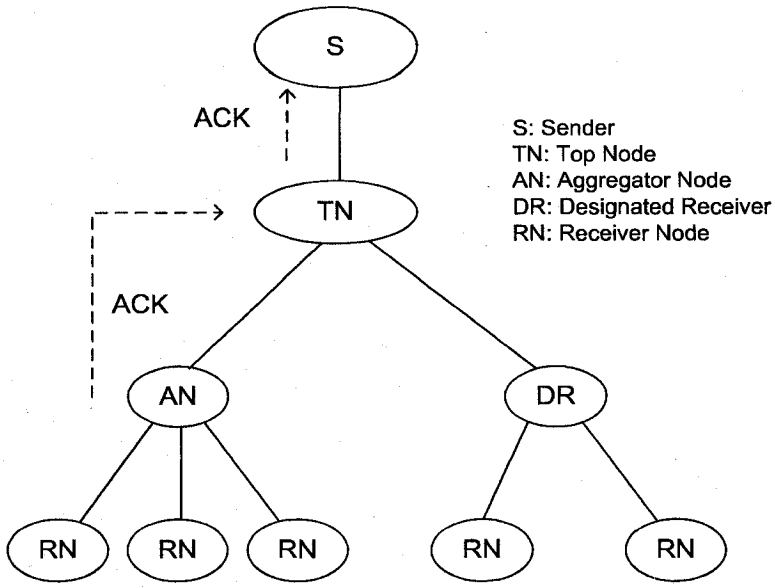

Figure 3.4: RMTP2 Tree Structure

We have used the extended version of UML/SPT to model the main requirements of the protocol RMTP2. We have used the UML sequence diagrams for the basic interactions. The latter are composed using UML interaction overview diagrams, and the real-time requirements are captured using the extended UML/SPT profile. In the following, we present the UML/SPT models for heartbeat packets, parent failure detection and join algorithm behavioral requirements of RMTP2.

3.2.1 Heartbeat Packets

The nodes cooperate to maintain the multicast tree integrity. Parent nodes send periodic heartbeat messages to notify their liveliness to the child nodes. This enables the child nodes to detect the parent failure and join another parent. This requirement calls for periodicity and multicast communication modeling. We use a periodic timer modeled using the stereotype ≪RTTimer≫ from the time sub-profile of UML/SPT to model periodicity and our ≪Multicast≫ stereotype to model multicast communication. Figure 3.5 illustrates how this requirement could be described using UML Sequences Diagrams and UML/SPT stereotypes.

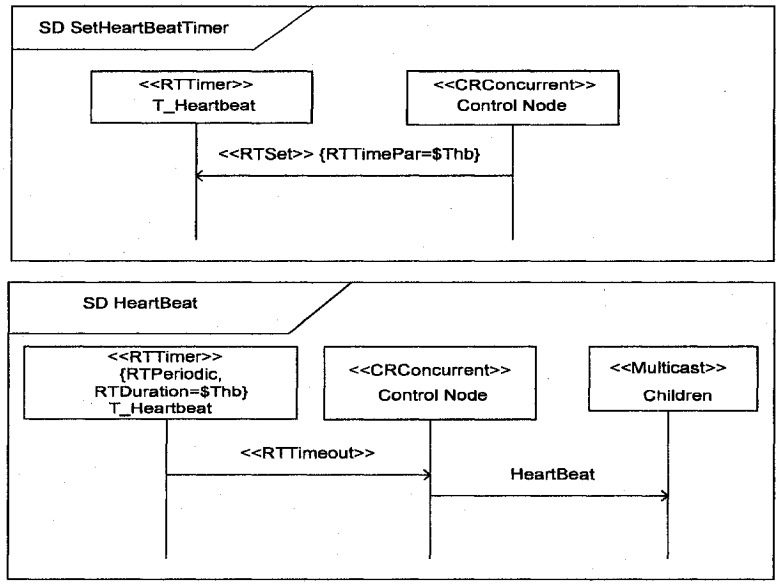

Figure 3.5: Heartbeat Packets Requirement Model

3.2.2 Parent Failure Detection

If a child node does not receive his parent heartbeat for a time interval specified by $F*Thb$, where F is a failure threshold constant, a parent failure is detected. Figure 3.6 illustrates the different scenarios modeling this requirement and Figure 3.7 is a UML overview interaction diagram composing these scenarios

3.2.3 Join Algorithm

A receiver node must join a multicast tree in order to be able to send acknowledgments or ask for retransmissions. The receiver node sends a *Joinstream* packet to its parent node and waits a period of time of *T_joinstream* for the response. This is specified with the UML/SPT annotated sequence diagram *SD Join* in Figure 3.8. The parent node sends as response either a *JoinConfirm* packet or a *JoinAck* in the case where it cannot handle the request immediately. The behavior of the receiver node in both cases is specified respectively in the *SD Confirm* and *SD JoinAck* in Figure 3.8. If no response is received upon receiving a *T_join* timeout, the receiver node retransmits the *JoinStream* request (*SD JoinFail*) in Figure 3.8. This is repeated for a maximum of *RJoin* times before reporting parent

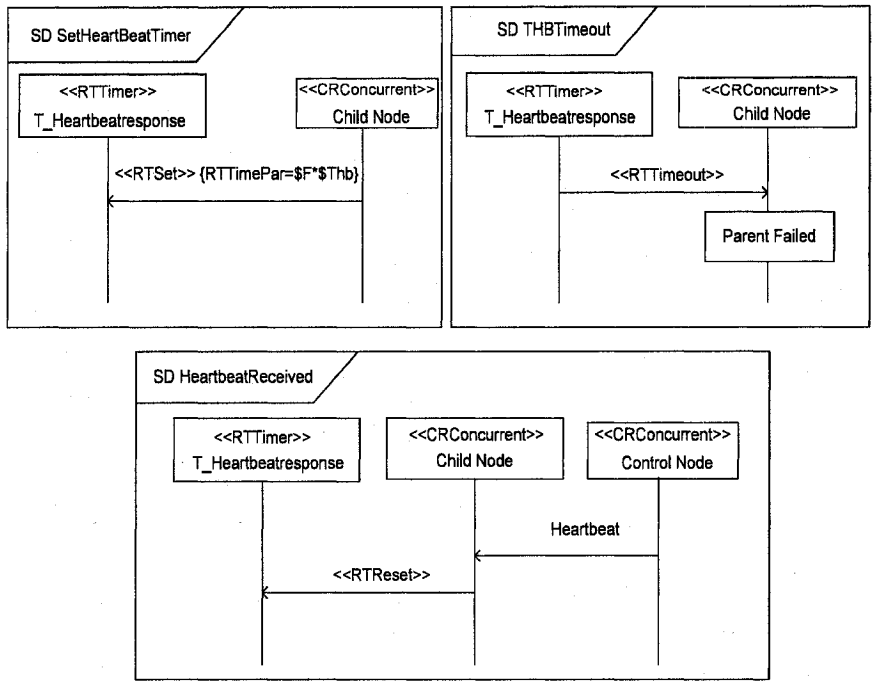

Figure 3.6: Parent failure Requirement Scenarios

unreachable error (*SD Can_Not_Reach*) in Figure 3.8. In the case where a receiver node gets a *JoinAck* from its parent node, it keeps transmitting *JoinStream* requests with waiting times growing exponentially. This is modeled with the sequence diagram (*SD JoinAck*) as illustrated in Figure 3.8. The whole join connection algorithm is specified by composing the different scenarios using a UML interaction overview diagram as illustrated in Figure 3.9.

3.3 UML/SPT-based vs. MSC-based RMTP2 requirement modeling

We compare the extension and modeling exercises using UML/SPT with the ones presented in [51] using MSC. The main criteria used for this comparison are: the language built-in constructs, the time-related mechanisms, the multicast communication extension, the extension approach, and the notation used for the extension. This comparison is shown in Table 3.5 and can be summarized as follows:

- **Language built-in constructs**: MSC and UML are comparable in terms of expressiveness to model behavioral requirements [49]. Both languages provide constructs

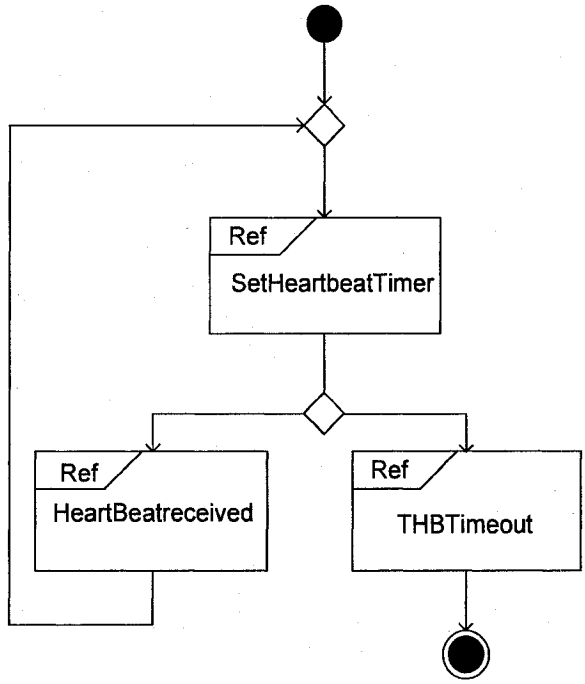

Figure 3.7: Parent failure Requirement Model

allowing for the expression of control flow, basic scenarios and their composition. In [51], bMSCs have been used to model the basic scenarios and HMSC have been used to compose them. We have used simple UML sequence diagrams for basic scenarios and interaction overview diagrams to compose them expressing more complex behavior as in the join algorithm requirement.

- **Time-related mechanisms**: MSC-2000 allows for expressing time constraints as well as time-related mechanisms such as timers. UML/SPT is probably more expressive in this regard. It is easier, for instance, to express periodicity, which is useful for periodic behavioral requirement such as the heartbeat packet requirement of RMTP2. With UML/SPT, periodicity can be modeled using a periodic timer (using the ≪RTtimer≫ with the tag value RTperiodic). With MSC, this can be modeled, as this was emphasized in [51], using either a time interval inside a loop or a loop composition of two basics MSC using an HMSC. On the other hand, the instance delay concept introduced in [130] can also be used to express process periodicity in general.

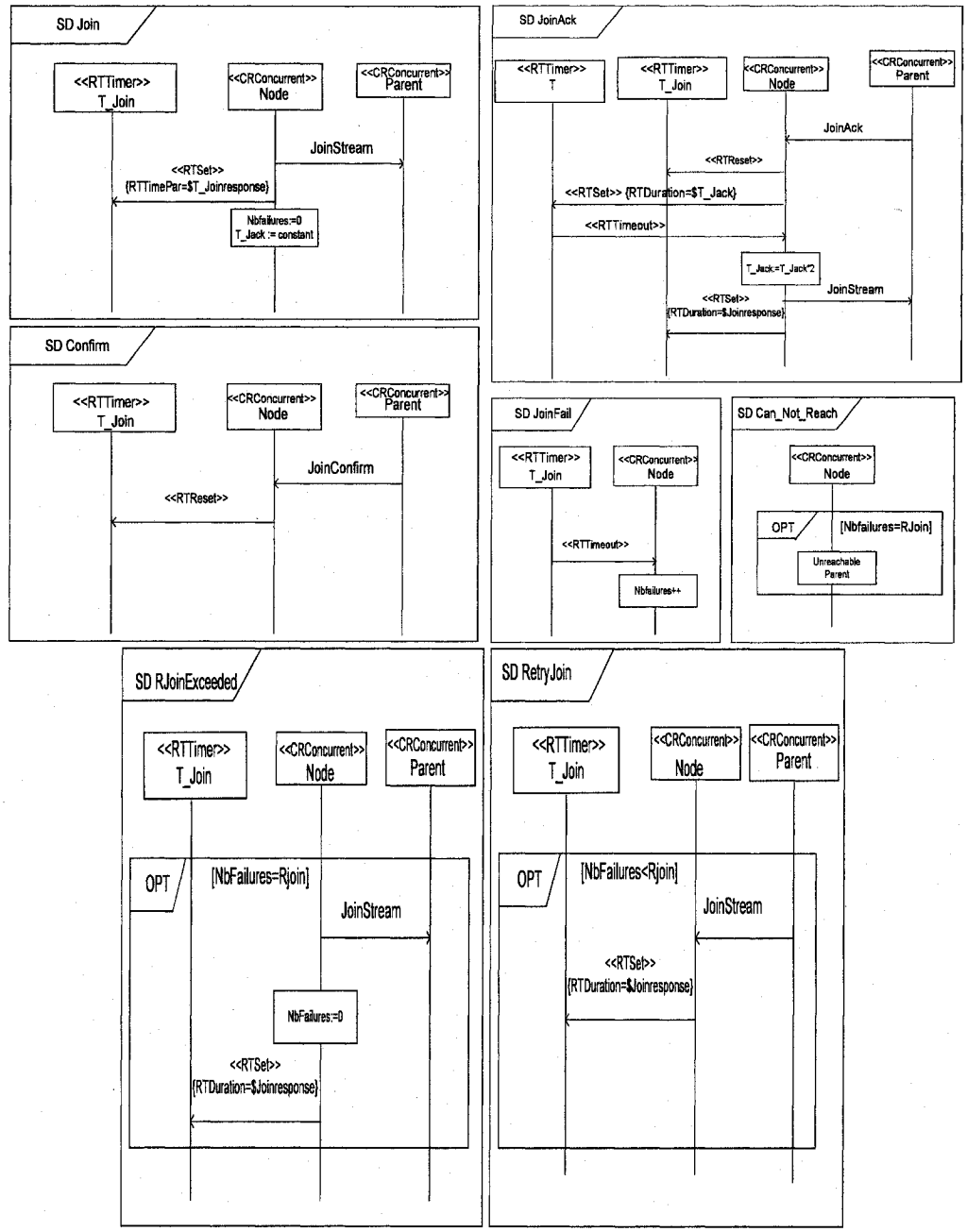

Figure 3.8: Scenarios for Tree Connection

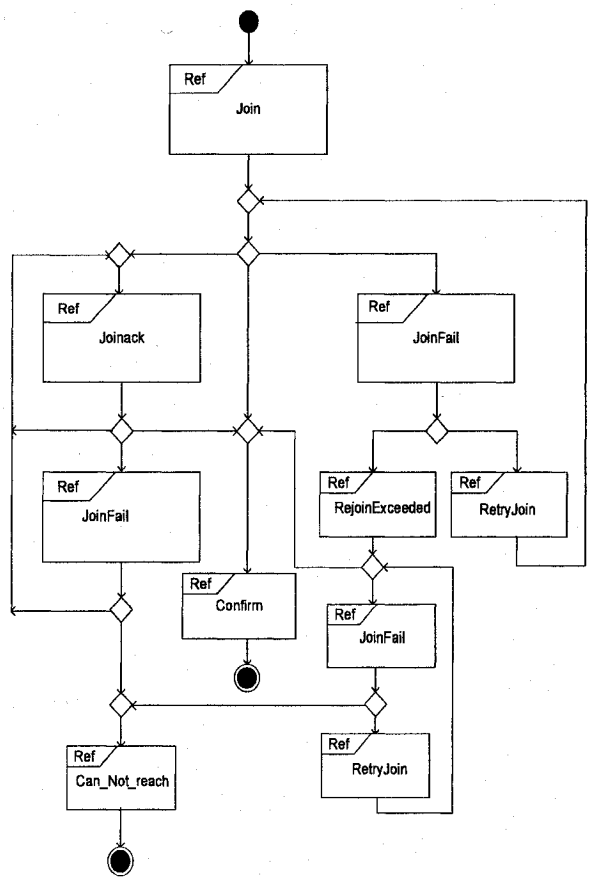

Figure 3.9: Tree Connection Requirement Model

- **Concurrency**: RMTP2 behavior involves the interaction of concurrent communicating entities. These are modeled in [51] by MSC instances which are implicitly concurrent. UML/SPT, on the other hand, allows a more explicit modeling of concurrency. We have used the ≪CRConcurrent≫ stereotype to identify the concurrent entities.

- **Multicast communication extension**: Both MSC and UML/SPT needed to be extended to model multicast communications.

- **Approach:** In [51], a formal semantics has been proposed for the multicast communication extension. We have defined our extension through a metamodel with constraints expressed in OCL.

- **The extension notation**: In [51], an MSC instance representing a multicast group is indicated by a simple note attached to the instance. We have proposed new stereotypes to model multicast communications. Stereotyping is the UML's standard way to introduce new and specific modeling elements.

3.4 Related Work

MSCs have been extensively used to capture the high-level behavioral requirements in telecommunication software engineering. In addition to the official extensions of MSC through the successive versions MSC'92, MSC'96 and MSC'2000, several other extensions have been proposed in the literature including [66], [128], [130]. In particular, Hélouët presented in [51] an MSC-based model for the requirements of the RMTP2 protocol. In order to model multicast communication, Hélouët proposed an extension of his semantics for MSC. For this extension as well as the aforementioned ones to be effective, they should

Criterion	MSC-based Model	UML/SPT-based Model	Comments
Language built-in construct	bMSC hMSC MSC operators	UML Sequence Diagrams UML Interaction Overview Diag. UML Interaction operators	Comparable Expressiveness between MSC-2000 UML interactions [49]
Time-related mechanisms	Timer + HMSC loop composition Time constraints + MSC loop operator	Periodic Timer ≪RTtimer≫ + tag value *RTperiodic*	With UML/SPT it is easier to express time constraints, periodicity
Concurrency	MSC instances are implicitly concurrent	Explicit concurrency modeling ≪CRConcurrent≫	UML/SPT allows explicit concurrency modeling
Multicast communications extension	Needed Not standardized	Needed Easy and natural	UML/SPT extensibility using UML built-in mechanisms facilitates the extension
Approach	Tentative formal semantics [51]	Metamodel OCL constraints	MSC extension in [51] is formal but not standardized. Our extension is not formal but takes advantage of the standard extensibility
Introduced notation	Simple note attached to an MSC instance modeling a multicast group	New Stereotypes	UML Standard way to introduce new modeling elements

Table 3.5: MSC vs UML/SPT Behavioral Modeling Summary

be integrated to the standard.

Our proposal for an extension of UML/SPT is inspired by Hélouët's extension of MSC. It is, however, easier and more natural because of the built-in extensibility mechanisms provided by UML. Moreover, the UML/SPT model for RMTP2 presented in this chapter takes advantage of the time concepts and time-related mechanisms offered by UML/SPT profile. For instance, expressing periodicity, which is necessary to model the heartbeat requirement, is simply modeled using a periodic timer *≪RTTimer≫* {*RTperiodic, RTDuration=value*} such as in the basic scenario *SD Heartbeat* in Figure 3.5, while this required either the introduction of a non standard notation and a loop or an HMSC loop composition in [51].

Several other proposals of extensions to UML/SPT using UML extensibility mechanisms have been presented in the literature. Cortellessa et al. presented in [20] a similar approach to extend UML/SPT to represent the concepts used in the reliability analysis domain. A metamodel for these concepts was presented and their relationship to UML/SPT metamodel was illustrated. A set of new stereotypes was introduced as well. Rodrigues et al. defined in [107] a profile for reliability analysis. It is also an approach to bridge the gap between UML/SPT models and MSC enabling early reliability prediction. Addouche et al. presented in [1] a UML profile called DAMRTS aiming at adding stochastic and probabilistic information to real-time systems models to enable their dependability analysis. This profile is an extension for UML/SPT, but neither its metamodel was extended nor new stereotypes were presented.

3.5 Conclusion

In this chapter, we motivated and presented an extension of the UML/SPT profile to enable multicast communications modeling. This extension was defined using the profile definition method. Specifically, we presented a metamodel encapsulating the main concepts involved in multicast communications. We specified the semantics of the concepts introduced in the domain model using OCL constraints to give declarative definitions for these concepts.

We mapped the domain model concepts to the UML with new stereotypes. We illustrated the application of this extension with the modeling of the main behavioral requirements of RMTP2 protocol. We have used the UML sequence diagrams to express the basic scenarios and the UML interaction overview diagrams to compose those scenarios. The extended version of UML/SPT has been used to model the time-related mechanisms, concurrency and multicast communication requirements of the protocol. Finally, we have contrasted this exercise with the extension of MSC for the modeling of RMTP2 [51].

There is an interesting issue related to the metamodeling approach used for defining UML profiles in general as well as for their extension. The concepts required for a given domain could be modeled in a variety of manners, which lead to different metamodels. For instance, two different metamodels have been proposed in [20] and [107] for the reliability prediction domain and used to extend UML/SPT metamodel. Therefore, it is necessary to assess the consistency between the different profiles and extensions.

Chapter 4

Timed-automata Semantics and Analysis of UML/SPT Models with Concurrency

As mentioned before, UML supports a multi-view modeling approach. This is very important to cater with software complexity. The system behavior, for instance, can be modeled using state machines and sequence diagrams. The former describe the individual behavior of the system components while the latter capture the interactions/collaborations between these components. However, such modeling approach faces the issue of consistency [53]. Each view or diagram of the system has to be *internally consistent* as well as *consistent with the other views and diagrams.* This issue of consistency becomes more challenging for UML/SPT models. Indeed, UML/SPT models are in addition enriched with other aspects such as time constraints and concurrency using the appropriate stereotypes. These enrichments complicate the *behavioral consistency* of UML/SPT models.

In this chapter, we focus on the concurrency dimension of UML real-time design models. We capture the semantics of the concurrency domain model of UML/SPT using timed automata [3]. We provide an abstract definition of a UML/SPT concurrent model and we define the semantics of a such model in terms of timed automata. This mapping allows for the validation of the *behavioral consistency* of UML/SPT models with concurrency using

existing model checking techniques.

4.1 Concurrency Modeling using UML/SPT

The package *RTconcurrencyModeling* encapsulates a domain model for concurrency modeling. This model is expressed using the class diagram shown in Figure 4.1. The semantics of the main concepts introduced in this domain model is informally described in English [91]. This description contains some ambiguities, which need to be clarified in order to enable a rigorous analysis of models. Our aim is to use a formal language like timed automata as a semantic domain onto which models using this concurrency domain model are mapped. Therefore, we need to make some semantic choices for the aspects that are open to interpretation.

The main entity in the concurrency domain model is a *Concurrent Unit.* It is defined as an active resource instance that executes concurrently with other concurrent units. An *active* resource is defined in *RTresourceModeling* package as an entity capable of generating stimuli without being prompted by an explicit stimulus as opposed to a *passive* resource, which does not generate its own behavior but reacts to stimulus [91]. It is interesting to mention the similarity between the concept of concurrent unit and the active class/object defined in UML 1.4 on which UML/SPT is based. The standard specification [91] does not, however, make any link between these two concepts.

The standard specification states that all the behavior in a system is carried out by the concurrent units executing the actions of the different scenarios. Consequently, we infer the following assumption:

Assumption 4.1 *Concurrent units are the only support of scenario executions in the system.*

This assumption means that any scenario should have the thread associated with a concurrent unit to proceed. Therefore, the only threads of control available at run-time are those associated with the different concurrent units. The rational behind this interpretation is that it gives the designer a full control over the different sources of concurrency in a design model. We argue that this helps in making the design choices in terms of concurrency and

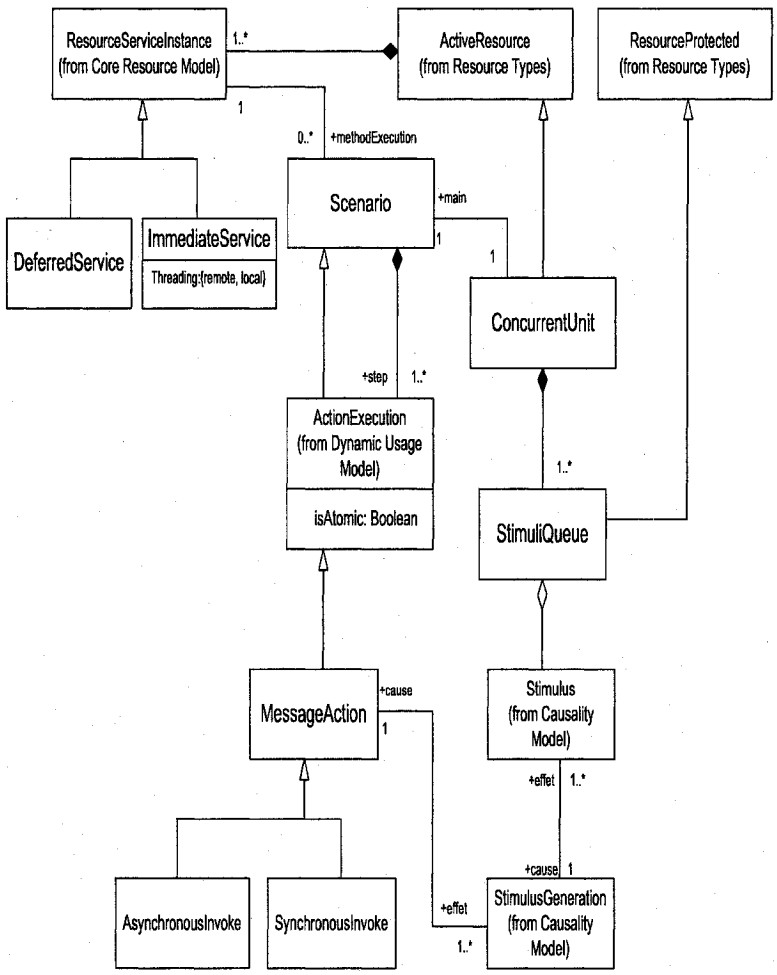

Figure 4.1: Concurrency Domain Model of UML/SPT

allows for the analysis of the design model to validate concurrency-related properties.

As shown in Figure 4.1, a concurrent unit owns one or more service resource instances through a composition relationship. It also owns one or more stimulus queues to hold stimuli that might arrive while the concurrent unit is busy. The response to such stimuli is deferred until the concurrent unit is ready. This kind of services are referred to as deferred service in the standard specification [91].

In addition, each concurrent unit is associated with a *main scenario.* During its main scenario, the concurrent unit may execute either an explicit *receive action*, a *synchronous invoke* or an *asynchronous invoke.* The execution of a receive action triggers the appropriate method in the corresponding service instance. The standard states that during the service method execution, the main scenario may be blocked or may proceed concurrently. This is another ambiguity in the standard which needs to be resolved. Indeed, if a service instance may execute concurrently with the main scenario of the concurrent unit, this means it has a thread of control different from the one associated to the main scenario. This is clearly in contradiction with the previous assumption 4.1.

Moreover, a service request has a property called *threading* which may be local or remote. In the former case, the receiving instance spawns a local thread of execution to handle the request while in the latter case it assumes the availability of an existing thread. This statement is also ambiguous and needs to be clarified. It is not clear from the standard specification what the receiving instance is. Is it the concurrent unit receiving the service request from the stimulus queue? Or the service instance which will be triggered to execute the service? What does it mean exactly for the receiving instance to create a thread of execution? And, finally, which thread of execution should be assumed to be existing? Is it the thread of control associated with the concurrent unit that has executed the message action to request the service?

Answering the aforementioned questions requires to make some semantic choices. Our choices are based on the assumption that the only threads of control in the underling runtime environment are the ones associated with the concurrent units. Our motivation is to give the designer a full control over the concurrency sources and to enable the analysis of the design model in terms of concurrency.

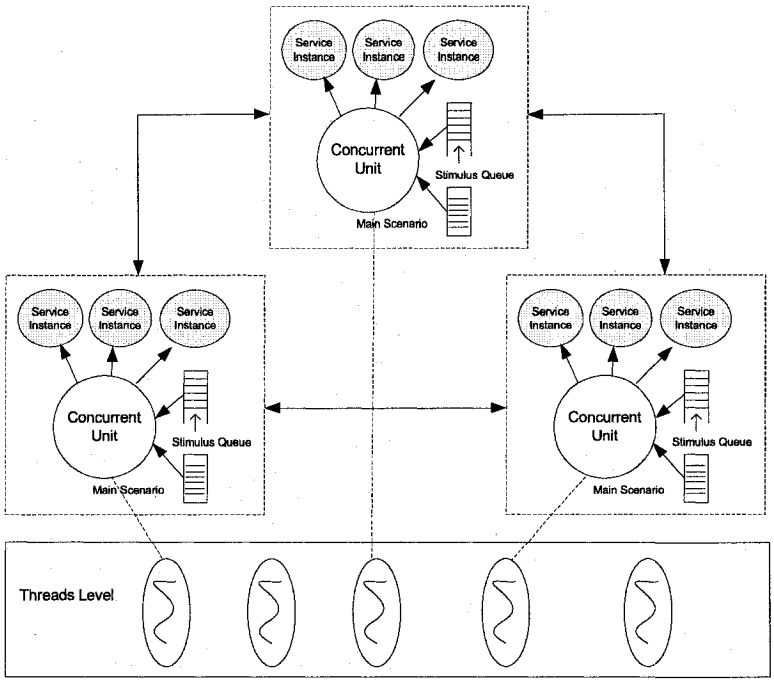

Figure 4.2: A Computational Model corresponding to the Concurrency Model in UML/SPT

Therefore, we consider the local threading to mean that the service instance will use the thread of control associated with its concurrency unit and the concurrent unit main scenario is then blocked during the execution of the service. Remote threading means that the service instance will use the thread of control associated with the concurrent unit that requested the service. Consequently, the concurrent unit and the service instance may proceed concurrently while the requesting concurrent unit is blocked waiting for service response.

The computational model corresponding to this semantics is depicted in Figure 4.2. It is worth mentioning that this model is very similar to the concurrency model based on the concept of *activity group* [82], which is composed of one active object and a set of passive objects. The only concurrency is between the different activity groups while within an activity group the behavior is sequential. This concurrency model has been defined formally in [22], [23] and implemented in [82] using the IF environment [15]. We will discuss this further in Section 4.6. The difference with the computational model we are considering is that behavior within a concurrent unit can be concurrent but using the thread of control of the caller concurrent unit.

4.2 Semantic Domain: Timed Automata

The theory of timed automata was introduced by Alur and Dill [2] [3]. It is now a well-established formal model for real-time systems. This formalism provides concepts such as time through clocks, parallel composition of timed automata, and synchronization through channels. This makes it suitable to capture the dynamic and concurrent behavior of systems. Consequently, we propose to use timed automata as a semantic domain to express formally the semantics of the concurrency domain model defined in UML/SPT.

A timed automata is a non-deterministic finite state machine extended with real-valued clocks. The states, called locations, may have invariants. These are conditions on clocks of the form $x \sim c$ where x is a clock, c is an integer constant and $\sim \in \{<, \leq\}$. The transitions may be labeled using triples composed of a guard, an action and clock reset operations. The guard of a transition is a conjunction of timing constraints of the form $x \sim c$ or $x - y \sim c$ where x and y are clocks, c is an integer constant and $\sim \in \{<, \leq, >, \geq, =\}$.

Formally, let *Act* be a set of actions, C a set of real-valued clocks, and $B(C)$ is the set of the conjunctions over conditions of the form $x \sim c$ or $x - y \sim c$ where x and y are clocks, c is an integer constant and $\sim \in \{<, \leq, >, \geq, =\}$.

Definition 4.1 *A timed automata over Act and C is a tuple* $\langle L, l_0, E \rangle$ *where*

- L *is a finite set of locations.*
- l_0 *is the initial location.*
- $E \subseteq L \times B(C) \times Act \times 2^C \times L$ *is a set of edges between locations.* □

Initially, all the timed automata clocks are set to zero. These clocks progress when the automata is in a certain location. A transition may be instantaneously fired if its clocks values satisfy the guard.

At the semantic level of timed automata, the state of a system of timed automata is a pair composed of a control location and a clock valuation. Execution traces of timed automata are infinite sequence of system states which satisfy the invariants. The transitions between the system states are labeled either by instantaneous actions or positive real numbers representing time delays. This is expressed in the following formal definition:

Definition 4.2 *Let $\mathcal{V} = \{\nu : C \rightarrow R_{\geq 0}\}$ be the set of clock valuations. The semantic state of a timed automata $\langle L, l_0, E\rangle$ is the pair (l, u) composed of a location $l \in L$ of the automaton and a clock evaluation $u \in \mathcal{V}$. The semantics is defined by a labeled transition system $\langle S, s_0, \rightarrow\rangle$ with two types of transitions:*

- ***Delay transitions:*** $(l, u) \xrightarrow{d} (l, u + d)$
- ***Action transitions:*** $(l, u) \xrightarrow{a} (l', u')$ *if* $l \xrightarrow{g,a,r} l' \in E$ *and* $g \vDash u$ *and* $u' = [r \mapsto 0]u$

Where:

- $u + d \in \mathcal{V}$ *and* $\forall x \in C : (u + d)(x) = u(x) + d$
- $\forall r \subseteq C$ *and* $\forall x \in r : [r \mapsto 0]u(x) = 0, u(x)$ *otherwise* □

We use in particular the UPPAAL model checking tool [69] to verify properties of interest in the system's model. In UPPAAL, a system is modeled as a network of timed automata extended with synchronization primitives. These may be of the form $a!$ to initiate a synchronization using the channel a or $a?$ to accept it. UPPAAL allows, in addition, for the declaration of integers variables and one-dimension integer arrays. Constraints on these variables and arrays could be used in the transitions guards and assignment to the variables could also be used in the actions. Integer variables and clocks are however incompatible and thus cannot be compared neither be assigned to each other.

4.3 Timed Automata-based Semantics of UML/SPT Concurrent Models

In this section, we formally define the semantics of the UML/SPT concurrent domain model using timed automata as a semantic domain. We then present a mapping of a UML/SPT concurrent model to timed automata.

Definition 4.3 *A UML/SPT concurrent design model $\mathcal{M}_{Conc}$ is a set of concurrent units. Each concurrent unit is a tuple $\mathcal{CU} =< \mathcal{MS}, \mathcal{Q}, \mathcal{SI} >$ composed of its main scenario $\mathcal{MS}$, its stimulus queue $\mathcal{Q}$ and its service instance set $\mathcal{SI}$.* □

Definition 4.4 *The semantics of $\mathcal{M}_{Conc}$ is defined as the parallel composition of timed automata corresponding to the main scenario, the stimulus queue and the set service instances of the concurrent units in the model:*

$$[\![\mathcal{M}_{Conc}]\!] \stackrel{def}{=} \|_{\mathcal{CU} \in \mathcal{M}_{Conc}} ((TAMainScenario_{\mathcal{CU}} \parallel TAQueue_{\mathcal{CU}}) \|_{j \in \mathcal{CU}.\mathcal{SI}} TAService_j)$$

□

In the following sections, we present and discuss a mapping of the main components of a concurrent unit into timed automata. In order to simplify the presentation, we use the following notations:

- Main scenario: $\mathcal{MS} =< a_i >$ where $i = 1..m_{\mathcal{CU}}$
- Basic actions: $a_i = synchInvoke \mid asynchInvoke \mid receiveAct$
- Given an automaton transition t, we denote its synchronization part $t.synch$, its guard $t.guard$, and its action $t.a$.

4.3.1 Concurrent Unit Timed Automata

Each concurrent unit is mapped to a set of corresponding timed automata according to Algorithm 1 shown below. The first timed automata will capture the behavior of the main scenario of the concurrent unit. The second one implements the behavior of the stimulus queue associated with the concurrent unit. This allows to capture the asynchrony of the stimulus arrivals while the concurrent unit is busy and implicitly captures the notion of deferred service. In addition, a set of automata are built for the different service instances as will be discussed later in Section 4.3.2.

The timed automata corresponding to the stimulus queue, denoted $TAQueue_{\mathcal{CU}}$, captures the asynchrony of the stimulus arrivals. It synchronizes with the other concurrent units timed automata that request services from the concurrent unit owning the queue through synchronization channels. Each one of these corresponds to a specific service instance of the concurrent unit. In addition, $TAQueue_{\mathcal{CU}}$ synchronizes with the timed automata corresponding to the main scenario, denoted $TAMainScenario_{\mathcal{CU}}$, for the receive actions.

Algorithm 1 Concurrent Unit Corresponding Timed Automata

for all $\mathcal{CU} \in \mathcal{M}_{Conc}$ **do**
  Build a TA $TAMainScenario_{\mathcal{CU}}$ for $\mathcal{CU}.\mathcal{MS}$
  Build a TA $TAQueue_{\mathcal{CU}}$ for $\mathcal{CU}.\mathcal{Q}$
  for all $s \in \mathcal{CU}.\mathcal{SI}$ **do**
    Build a TA TA_s
  end for
end for

Algorithm 2 is used to build the timed automata $TAQueue_{\mathcal{CU}}$ where UPPAAL integer and array integer data types are used. Indeed, each service instance is identified using an integer identifier and $TAQueue_{\mathcal{CU}}$ uses a local array integer as an underlying structure to store the services requests. In addition, $TAQueue_{\mathcal{CU}}$ and $TAMainScenario_{\mathcal{CU}}$ share an integer variable $\mathcal{CU}\_service$ used by $TAQueue_{\mathcal{CU}}$ to indicate the service corresponding to the next stimulus in the queue when the $TAMainScenario_{\mathcal{CU}}$ executes a receive action.

The timed automata $TAMainScenario_{\mathcal{CU}}$ is constructed to capture the behavior dictated

Algorithm 2 Concurrent Unit Queue Timed Automata

$\mathcal{CU}\_service$: **int** {integer variable shared between $TAQueue_{\mathcal{CU}}$ and $TAMainScenario_{\mathcal{CU}}$}
{Q: Queue that holds the integer identifier of each service waiting and manipulated through the helper functions $put()$ and $next()$}
let $Initial$:location
let t_0:transition
$t_0 \leftarrow < Inital, Inital >$
$t_0.synch \leftarrow \mathcal{CU}\_receive?$
$t_0.a \leftarrow \mathcal{CU}\_service := Q.next()$
for all $s \in \mathcal{CU}.\mathcal{SI}$ **do**
  let t:transition
  $t \leftarrow < Inital, Inital >$
  $t.synch \leftarrow s?$
  {$identifier()$ is a helper function that returns the service instance integer identifier}
  $t.a \leftarrow Q.put(identifier(s))$
end for

by the main scenario of the concurrent unit. If the action under consideration is a message action of the type $SynchronousInvoke$, $TAMainScenario_{\mathcal{CU}}$ synchronizes with the target concurrent unit timed automata, actually with its stimulus queue automata, through a synchronization channel having the requested service name. Then it stays in an intermediate location *waiting* for the response to the service. This is modeled by a transition whose synchronization uses a channel corresponding the service response from the called concurrent

unit timed automata. If the action is an *asynchronous invoke*, the timed automata synchronizes with the called concurrent unit timed automata to send the service request but does not synchronize with the called concurrent unit for service achievement. If the action is an explicit *receive action*, the generated timed automata synchronizes with the timed automata representing the service instance. Two cases are distinguished here: if the service threading is local, then the $TAMainScenario_{CU}$ synchronizes with the service instance's automata waiting for the service to finish. If the threading is remote, which means that the service instance will execute using the thread of control of the caller concurrent unit as discussed in Section 4.1, then the timed automata will continue the execution of its scenario without waiting. Consequently the generated timed automata does not synchronize with the service instance timed automata to wait for the service achievement. We emphasize here that this accounts for our semantic choice where the only threads of control underlying a real-time application designed using UML/SPT are those associated with the concurrent units. The motivation of this choice is to have a full control on the concurrency mechanism. This time automata generation is depicted in Algorithm 3.

4.3.2 Service Instance Mapping

The service instances associated with a concurrent unit are passive objects. They do not have a thread of control of their own and should rely on the concurrent unit's thread of control to carry out their behavior. This behavior is generally specified using UML state machines, which can be transformed into timed automata as it is already done and reported in the literature [64]. We assume that these established results can be used to generate the timed automata associated with the service instances. The concurrent unit timed automaton $TAMainScenario_{CU}$ synchronizes with each of its service instances timed automata through specific channels s to trigger their execution upon receiving the corresponding service request. In the case of a local threading, the service instance will be executed using the thread of control of its concurrent unit. The latter is then blocked waiting for the service instance to finish using a synchronization through the channel s_return. Otherwise, i.e. the threading is remote, $TAMainScenario_{CU}$ does not wait for the service instance timed automaton as this will use the requesting concurrent unit's thread of control and can

Algorithm 3 Concurrent Unit Main Scenario Timed Automata

let $Initial, l$: location {$Initial$ location of $TAMainScenario_{\mathcal{CU}}$}
$l := Initial$
for all $a_i \in \mathcal{CU}.\mathcal{MS}$ **do**
  if $a_i == synchInvoke$ **then**
    let l':location and **let** t_1: transition
    $t_1 \leftarrow < l, l' >$
    $t_1.synch \leftarrow a_i!$
    let t_2: transition
    $t_2 \leftarrow < l', l >$
    $t_2.synch \leftarrow return\_a_i?$
  else
    if $a_i == asynchInvoke$ **then**
      let t:transition
      $t \leftarrow < l, l >$
      $t.synch \leftarrow a_i!$
    else {$a_i == receiveAct$}
      let $switch$:location and **let** t:transition
      $t \leftarrow < l, switch >$
      $t.synch \leftarrow \mathcal{CU}\_receive!$
      for all $s \in \mathcal{CU}.\mathcal{SI}$ **do**
        if $s.threading == local$ **then**
          let $s_{running}$:location and **let** t_1:transition
          $t_1 \leftarrow < switch, s_{running} >$
          $t_1.guard \leftarrow \mathcal{CU}\_service == identifier(s)$
          $t_1.synch \leftarrow s!$
          let t_2:transition
          $t_2 \leftarrow < s_{running}, l >$
          $t_2.synch \leftarrow s\_return?$
        else {$s.threading == remote$}
          let t_1:transition
          $t_1 \leftarrow < switch, l >$
          $t_1.guard \leftarrow \mathcal{CU}\_service == identifier(s)$
          $t_1.synch \leftarrow s!$
        end if
      end for
    end if
  end if
end for

proceed in parallel. This scheme is shown in the innermost *if-then-else* control structure of Algorithm 3.

4.3.3 Time Constraints Mapping

Real-time systems should carry out their functionality while satisfying the associated time constraints. These time constraints reflect requirements that are generally dictated by the systems' environment. Using UML/SPT, the time constraints are modeled using stereotypes defined in the time modeling package. The main stereotypes are:

- $\ll RTevent \gg$ provides the *RTat* tag, which can be used to represent specific instants of an event occurrence.
- $\ll RTstimulus \gg$ provides the couple of tags (*RTstart*, *RTend*) that can be used to model the occurrence times of a stimulus send and reception events, respectively, and the *RTduration* tag to model the time required for a message to be transmitted over communication media.
- $\ll RTaction \gg$ has the same tags as $\ll RTstimulus \gg$ to model time constraints on some action or computation at some level of abstraction.
- $\ll RTtimer \gg$ is used to model a timer.

The time constraints specified using the aforementioned stereotypes are captured in the timed automata formalism through constraints on clocks associated to the timed automata. The examples in the next two sections illustrate these stereotypes usage to model time constraints and how they are represented in the generated timed automata.

4.4 An Example of Transforming a UML/SPT Model with Concurrency into Timed Automata

In this section, we illustrate our proposal to capture the concurrency semantics in UML/SPT through a mapping into timed automata. The analysis of this example using model checking is presented in the next section.

We consider a system that expects two concurrent and periodic events from its environment. The behavior of the system is carried out by the collaboration of three main concurrent components. The design model is given by sequence diagrams annotated with stereotypes from UML/SPT as depicted in Figure 4.3 and Figure 4.4.

The timed automata corresponding to the concurrent units used in the concurrent design

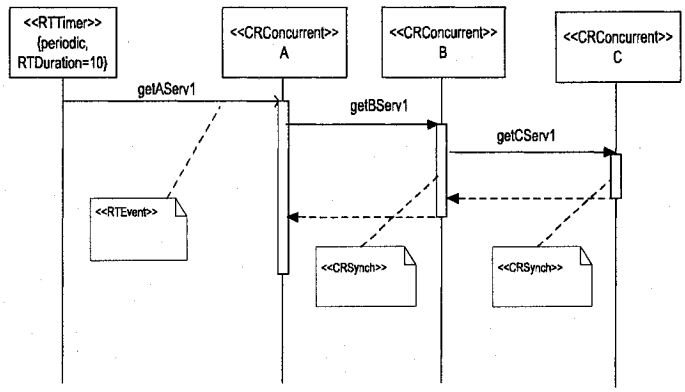

Figure 4.3: Concurrent Periodic Event One associated Behavior

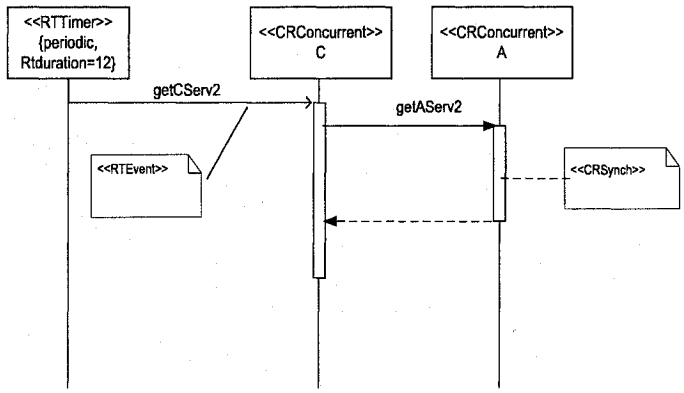

Figure 4.4: Concurrent Periodic Event Two associated Behavior

above are depicted in Figure 4.5, Figure 4.6 and Figure 4.7. These timed automata were built using the algorithms presented in the previous section but have been simplified. For example the timed automaton associated to the main scenario of the concurrent unit B, $TAMainScenarioB$, depicted in Figure 4.6, has only one service so the test on the value of the $B\_service$ integer variable has been removed and the timed automaton does not synchronize with any service instance as these are not explicit in the model and hence they were abstracted away. The timed automata corresponding to the stimulus queue have been

simplified also. The timed automaton $TAQueueB$ corresponding to the queue associated with the concurrent unit B, for example, does not use the local array integer Q. This is because there is only one kind of service associated with B so there is no need to queue their identities in Q. In addition, the location $Wait$ in the three automata is an instance of the location $s_{running}$ used in Algorithm 3 where a timed automaton waits for its service instance to finish. Since there is no explicit modeling of the service instances in this example, handling a service, such as $getAServ1$ for A for example, is simply calling synchronously another concurrent unit and waiting for its response. Finally, two special timed automata are used corresponding to the timers that will generate the service stimuli continuously as depicted in Figure 4.8.

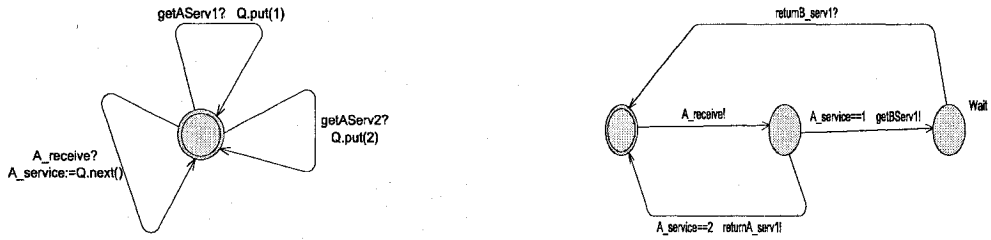

Figure 4.5: Concurrent Unit A Timed Automata

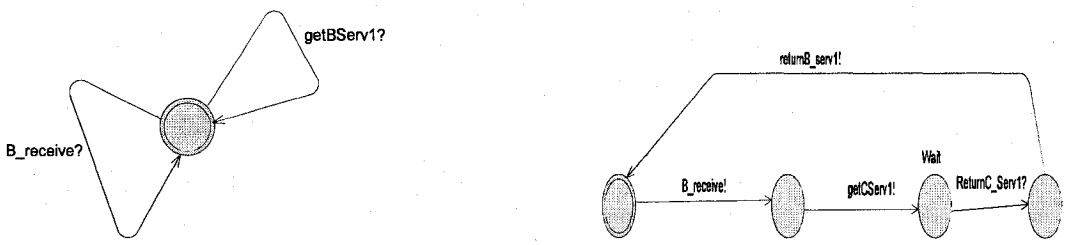

Figure 4.6: Concurrent Unit B Timed Automata

Figure 4.7: Concurrent Unit C Timed Automata

Figure 4.8: Timers Timed Automata

4.5 Model Checking UML/SPT Models with Concurrency

Using timed automata as a semantic domain to interpret the concurrency domain model used in UML/SPT confers a formal semantics to this important part of the UML profile for real-time modeling. In addition, this allows for the analysis of concurrent design models. This analysis can be achieved through model checking. Indeed, concurrency related issues such as deadlocks, livelocks and the impact of concurrency design choices on time constraints can be analyzed through the validation of appropriate properties, expressed in temporal logic. Model checking tools such as UPPAAL [69] and IF environment [15] have now reached a certain maturity and are very effective despite the difficult problem of state explosion.

Let us consider again the model presented in the previous section. A scenario where the concurrent unit C receives an external event requesting its second service $getCServ2$ right after A has received an external event requesting its first service $getAServ1$, may lead to a problematic state of the system. This scenario may lead into a state where the three concurrent units are waiting for each other since each one has executed a $synchronousInvoke$ action. This is a deadlock situation. The CTL expression 4.1 describes this situation and it is checked using the verifier of UPPALL. The scenario where this property holds is illustrated in Figure 4.9 output by the UPPAAL simulator. Therefore, the system may reach a deadlock state.

$$\exists\Diamond(TAMainScenarioA.Wait \ and \ TAMainScenarioB.Wait \ and \ TAMainScenarioC.Wait) \tag{4.1}$$

Some design decisions related to concurrency could be flawed when we take into account the time constraints. Conditions on the clocks in the timed automata corresponding to the

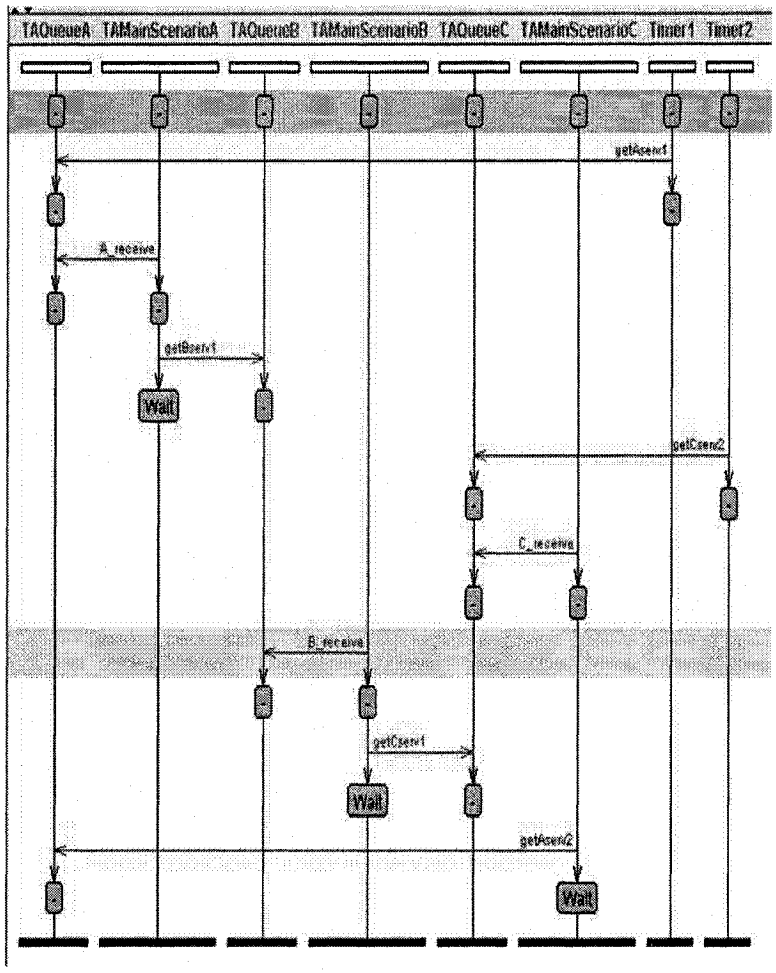

Figure 4.9: Deadlock Scenario in UPPAAL

UML/SPT concurrent models can be used to express such time constraints. This allows for checking whether these constraints may be violated. The technique we use is inspired by the technique of observer timed automata [28]. In the following, we present an example that illustrates how time constraints in a concurrent design model are captured and checked.

The sequence diagrams in Figure 4.10 and Figure 4.11 describe the behavior of a system composed of three concurrent components. The concurrent unit A is in charge of servicing a periodic external event. A uses the services of the concurrent unit B to achieve its service, which in turns uses the resource R. The system should also deal with a second periodic event for which the concurrent unit C is in charge of the corresponding service. C uses for its service the resource R shared with the concurrent unit B. A and B are required to accomplish their services with a deadline of 8ms and 7ms, respectively.

The timed automata capturing the semantics of the sequence diagrams with the stereotypes

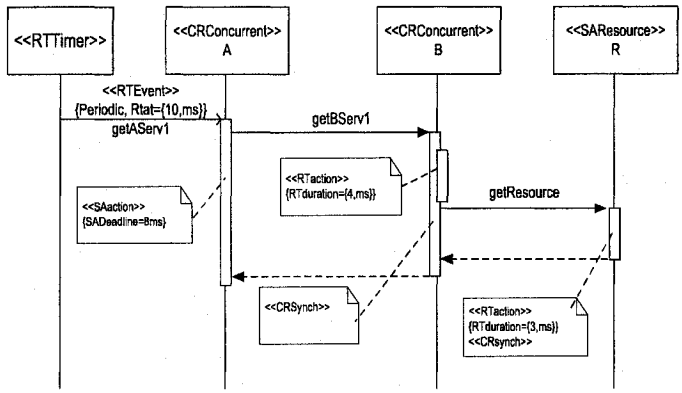

Figure 4.10: Periodic Event One with Time Constraints

from UML/SPT concurrency package are shown in Figure 4.12, Figure 4.13 and Figure 4.14. We only show and focus on the main scenario timed automata since the stimulus queue timed automata are straightforward. The timed automaton corresponding to the concurrent unit A uses a clock c and a condition on the clock expressed using the invariant $c \leq 8$ associated with the location $Wait$. The location $TAMainScenarioA.DeadlineMiss$ is reached through a transition with the guard $c == 8$. This happens in case the result from the concurrent B does not arrive in time and A misses its deadline. The same scheme is used in the timed automaton corresponding to the concurrent unit C. The timed automaton SR captures the behavior of the shared resource. The integer variable pr shared between the timed automata $TAMainScenarioB$, $TAMainScenarioC$ and SR is used to ensure the mutual exclusion, i.e the synchronization with SR using the channel $release$ respects the

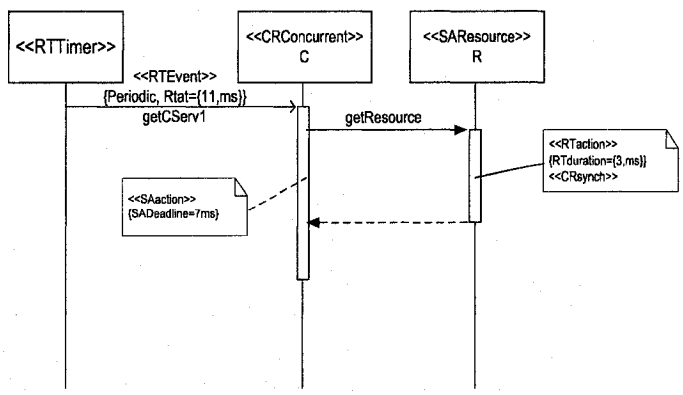

Figure 4.11: Periodic Event Two with Time Constraints

order of the resource acquisition.

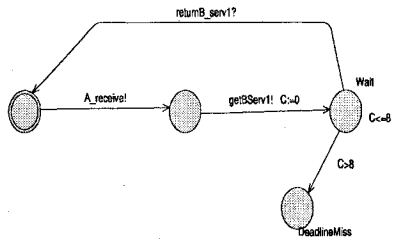

Figure 4.12: Concurrent Unit A Timed Automata

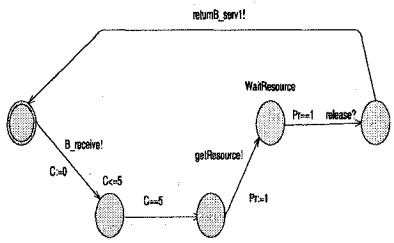

Figure 4.13: Concurrent Unit B Timed Automata

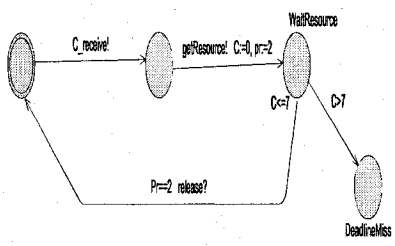

Figure 4.14: Concurrent Unit C Timed Automata

The generated set of timed automata has been checked using the UPPAAL to verify if a deadline could be missed. This is achieved using the property 4.2 expressed in CTL. This property holds in the model and Figure 4.17 shows a scenario that the UPPAAL simulator generated. Therefore, the deadlines may not be met.

$$\exists\Diamond(TAMainScenarioA.DeadlineMiss \ \ or \ \ TAMainScenarioC.DeadlineMiss) \qquad (4.2)$$

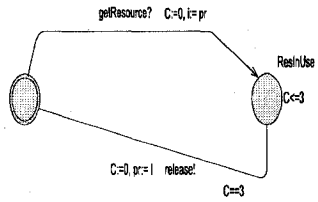

Figure 4.15: Shared Resource Timed Automata

Figure 4.16: Timer Timed Automata

4.6 Related Work

UML/SPT has been defined for UML 1.4 [85]. The latter includes in its metamodel provisions for concurrency modeling, e.g. active object, concurrent composite state, concurrency meta-attribute of the operation metaclass [33] [34]. Ober outlines in [83] the concurrency modeling features in UML 1.4 and their anomalies. It is worth mentioning here that despite some similarity between the concept of concurrent unit in UML/SPT and the active object in UML 1.4, the relation between them has not been addressed in the standard.

Damm et al. define in [22] a subset of UML called *krtUML* covering concepts used for real-time applications such as active object and synchronous/asynchronous communications. The semantics of *krtUML* is described formally using symbolic transition systems. Regarding concurrency, the UML concept of active object is generalized to the concept of *component*, which is a group composed of one active object and a set of passive servers. The active object acts as an event handler to the associated passive objects within a component. The semantics of *krtUML* enforces that there is just one thread of control active in one component. The concept of component is very similar to the concept of concurrent unit and its service instances defined in UML/SPT. The semantic choice of a single thread per component is also similar to our choice in terms of concurrency within a concurrent unit and its service instances, which is not clear in the UML/SPT standard.

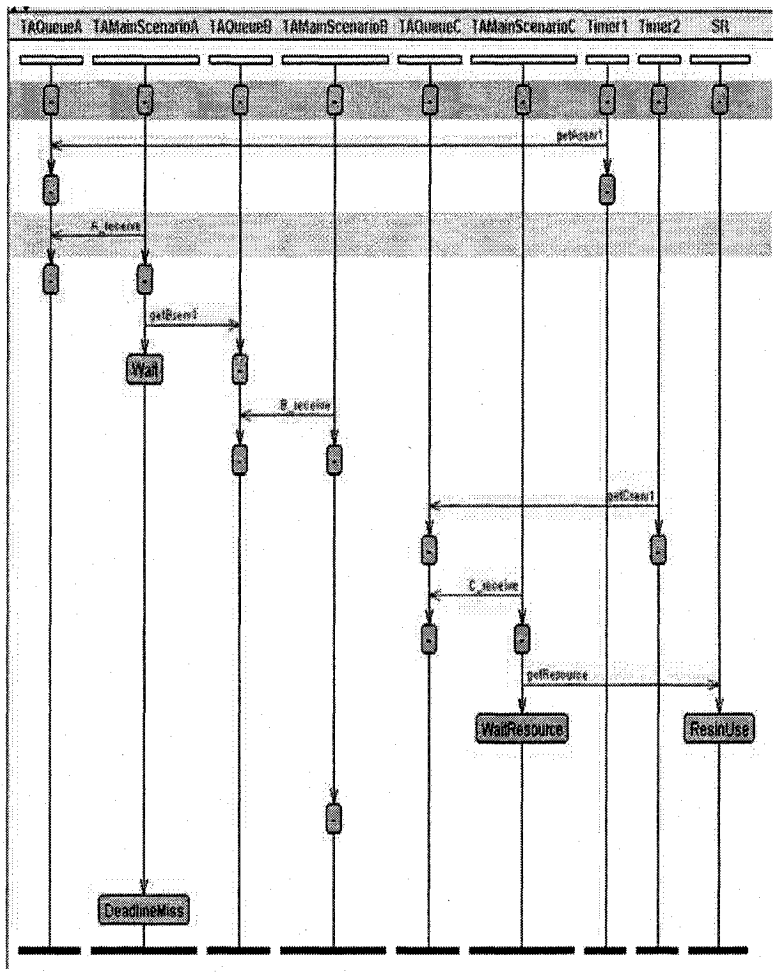

Figure 4.17: Concurrent Unit A Deadline Miss Scenario in UPPAAL

Ober et al. propose in [82] a model checking technique to validate UML models. The focus is on a subset of UML concepts used to define an operational view of a system. This work focuses on the implementation of the formal semantics defined in [22] and described previously. This implementation is based on a mapping to communicating extended timed automata in the IF format [15]. The time requirements are modeled using the time extensions defined in [44]. In addition, this work presents a property description language called observer object similar to observer automata and expressed using UML state machines. The concurrency model assumed in this work is based on units of concurrency called activity groups that are synonymous to the component concept defined in [22]. The semantic choice in terms of concurrency is such that the different activity groups run concurrently and the objects within an activity group run sequentially.

Madl et al. use in [74] timed automata as an underlying computation model called DRE Semantic Domain for tasks with time-triggered and event-driven interactions on a non-preemptive distributed platform. The objective is to check the schedulability of these tasks. The schedulability problem is translated to a reachability problem where a system of tasks is schedulable if a predefined error state is not reachable by any of the tasks' corresponding timed automata.

Finally, the transformation of the UML artifacts used to model the dynamic behavior into timed automata for purposes of verification has been the focus of several researches including [28], [64]. Firley et al. consider in [28] an approach to transform sequence diagrams with time constraints to observer timed automata. Knapp et al. address in [64] the issue of consistency between the main UML artifacts used to model the real-time system dynamic behavior: timed state machines and sequence diagrams with time constraints. The former express the detailed design of the system and the latter specify the main scenarios. This work proposed a technique for the verification of the consistency between the two views based on UPPAAL timed automata. The timed state machines are compiled into timed automata and the sequence diagrams annotated with time constraints are transformed to observer timed automata. The latter transformation is a slight extension to the technique proposed in [28]. The model checker UPPAAL is then used to verify the timed automata with respect to the observer timed automata. This technique is embodied in a prototype

tool called HUGO/RT. This work focuses however on the timing aspect of real-time and does not address the concurrency aspect in real-time UML models.

4.7 Conclusion

Real-time systems should not only compute their value correctly but should also satisfy prescribed time constraints. In addition, real-time systems are designed as concurrent components that collaborate to achieve the overall functionality. Consequently, a modeling language should support the modeling of the main features of real-time systems, including concurrency and time requirements. The UML profile for real-time, UML/SPT, defines domain models that encapsulate the concepts necessary for real-time modeling. In order to model the concurrency aspect of real-time systems, UML/SPT defines the concurrency domain model.

We have presented in this chapter a formalization of the concurrency domain model. We use the formalism of timed automata as semantic domain because it provides concepts such as concurrency, synchronization and time. We have reviewed the main concepts introduced in the concurrency domain model of UML/SPT and have made semantic choices to resolve some ambiguities in the standard. We have proposed a mapping of the concepts in the concurrency domain model into the timed automata formalism. A straightforward application of this formal semantics is the usage of the automata based model checking technique. In particular, concurrency-related properties such as deadlock or the impact of some design decisions in terms of concurrency on the time constraints requirements, can be validated at the model level. We have illustrated this validation with examples using UPPAAL.

Chapter 5

From UML/SPT Design Models to Schedulability Analysis: Approach and Implementation

There are several kinds of models which are often involved in the development of a software system. Some of these models are used to define the different aspects of a software system. These include UML class/object diagrams, which are used to specify the software system structure, and UML state machines or UML sequence diagrams, which are used to model the system behavior. In addition, other models are used to support the analysis and validation of software nonfunctional properties. These include, for example, performance models, such as, queuing networks (e.g., QN, EQN, and LQN) used for performance analysis [7] and tasks models used for schedulability analysis [68]. A software development framework should enable the seamless integration of the different types of models used throughout the software development process.

There is an increasing interest, as demonstrated in several research papers, such as [21], [46], [119], [124], in using MDA as a framework for integrating different models involved in the software development process. This is the motivation behind our MDA-compliant approach that aims at bridging the gap between UML/SPT models and the task model used in the schedulability analysis technique presented in [112].

In this chapter, we discuss this approach and we present a proof of concept for the proposed transformation. This consists of a prototype implementation of the transformation process using both a mature model transformation language, ATL, as well as a low-level implementation using XML-based technologies. The first implementation consists of the specification of the schedulability analysis domain model defined in UML/SPT [91] and our metamodel for schedulability analysis, respectively, using KM3 metamodel specification language [60]. We specify the model transformation using ATL [59] [62]. As for the XML-based implementation, we provide XML schemas corresponding to the source and target metamodels. These schemas are used to define valid XML documents representing the models manipulated by the transformation. We describe an implementation of the model transformation using XSLT template rules.

5.1 MDA-compliant Schedulability Analysis

There are now several well-established analysis techniques dedicated to the verification of non-functional properties of software systems. The challenge is how to bridge the gap between these well-established techniques and the software UML-based design models. Considering the relevant research initiatives, MDA emerges as an interesting approach to address this challenge.

We propose an MDA-compliant approach that aims at validating real-time software models using a well-established real-time schedulability analysis technique [111] [112]. Our approach consists of transforming a software PIM to another PIM more suitable for schedulability analysis, which then feeds back the results of this analysis to the design PIM. The design PIM can thus be validated before transforming it into a PSM and then generating code. Our approach is depicted in Figure 5.1. Our objective is to shift the schedulability analysis of object-oriented real-time models to the level of UML/SPT models. We define a rule-based transformation allowing for the derivation of schedulability analysis models from UML/SPT design models. The generated models are then analyzed for schedulability. The feedback from the analysis to the design model helps the modeler to make design-level decisions.

It is very useful to perform the analysis at the PIM level. This corresponds to the early

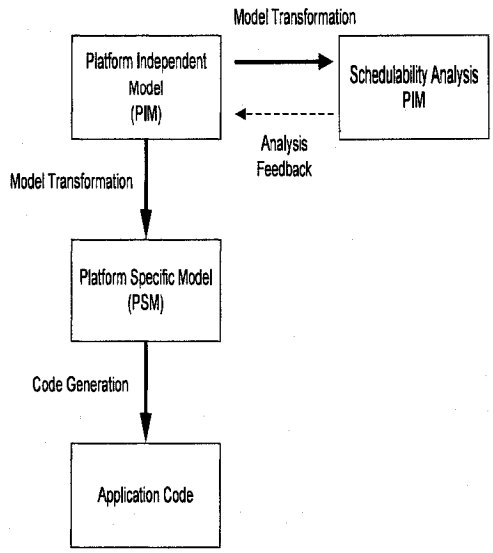

Figure 5.1: MDA-compliant Schedulability Analysis Approach

stages of the development process and hence helps in the early detection of flaws. However, this approach suffers from a lack of platform/implementation-dependent information. In terms of schedulability analysis, some key information such as event arrival patterns (e.g. periodicity) and priorities are platform independent. This information is often extracted from the system requirement analysis. It is true, however, that certain information cannot be available at this level - such as the actions' computation times. In realistic software engineering settings, this information can be estimated based on previous projects. On the other hand, schedulability analysis can also be performed at the PSM level, taking advantage of the availability of more information and providing more accurate analysis. However, PSMs correspond to late stages in the development process and consequently many design decisions should already have been made, which could seriously impact the implementation. Alternatively, another interesting approach would be to perform an incremental analysis. At the PIM level, the analysis uses the available/estimated information to guide the developer in making design decisions and then this analysis is refined along with the availability of more platform-dependent information at the PSM level.

5.2 From UML/SPT to Schedulability Analysis: Approach

The main concepts used in the approach we use to enable applying schedulability analysis to UML/SPT models using the MDA framework are shown in Figure 5.2. In the following, we give the details of the different steps of our approach.

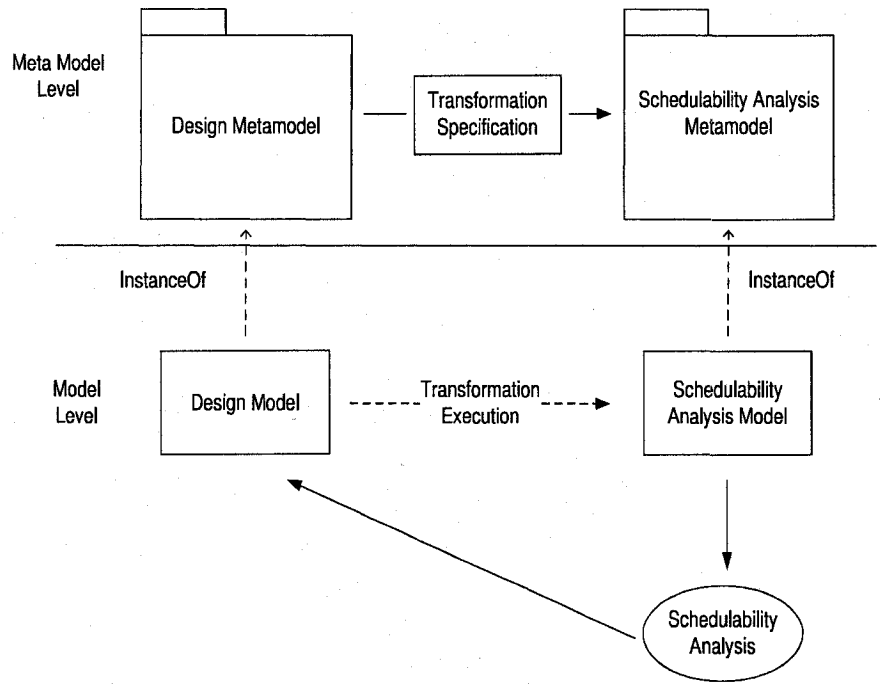

Figure 5.2: MDA-based Approach for Schedulability Analysis

5.2.1 Source Metamodel

The inputs to our model transformation are UML/SPT models, which are UML models enriched with information relevant for real-time context. The system structure is modeled using UML class diagrams and the behavior may be specified using UML state machines and UML sequence diagrams, for instance. UML sequence diagrams specify the different end-to-end behaviors of the system.

We are interested in validating the schedulability property of real-time design models. The schedulability-related information is captured using specific stereotypes. The main concepts underlying this information are captured in a domain model defined in the schedulability

analysis sub-profile, *SAProfile*, of UML/SPT [91].

The source metamodel to our transformation is depicted in Figure 5.3. This metamodel is a compilation of the main domain models defined in UML/SPT to support schedulability analysis. Its main components are the *dynamic usage domain model* defined in the *General Resource Modeling framework* of UML/SPT, some concepts from the *concurrency domain model* defined in *RTconcurrencyModeling* package, and the *schedulability analysis domain model* defined in the *SAProfile* package.

In this metamodel, the *RealTimeSituation* concept represents a specific analysis context. It is a specific configuration of resources including *ExecutionEngine* to model processors, *SResource* to model passive resources and *SchedulableResource* to model threads or tasks; and different entities, *SchedulingJobs*, contending for these resources. A *SchedulingJob* is composed of a *Trigger* modeling an external event having an arrival pattern that could be periodic, for instance, and a *Response*. The latter is the root action for a sequence of actions, *SAction*, separately schedulable. It is interesting to observe that an *SAction* is a nested construct. As depicted in Figure 5.3, an *SAction* is a subclass of *Scenario*. *Scenario* is in turns composed of a sequence of *ActionExecution*, which are also subclasses of *Scenario*. This allows all of the behavior compositions to be captured.

5.2.2 Target Metamodel

The schedulability analysis presented in [111], [112] is basically a busy-period response time analysis adapted to a real-time object-oriented model. The computations involved in this analysis are specified in [111], [112] through a set of quite elaborate equations that are not necessary to understand the metamodel. We do not reproduce these equations in this chapter but we recall the main elements required for a better understanding of the metamodel we are proposing to capture the concepts involved in this schedulability analysis.

A set of external events is given: $\xi = \{E_i\}$ where $i = 1..n$. Each event instance E_i triggers an end-to-end transaction, which is a sequence of actions in the system. An action is a *run-to-completion* processing unit composed of a sequence of sub-actions, i.e. $A_i = (a_{i_j}), j = 1..m_i$,

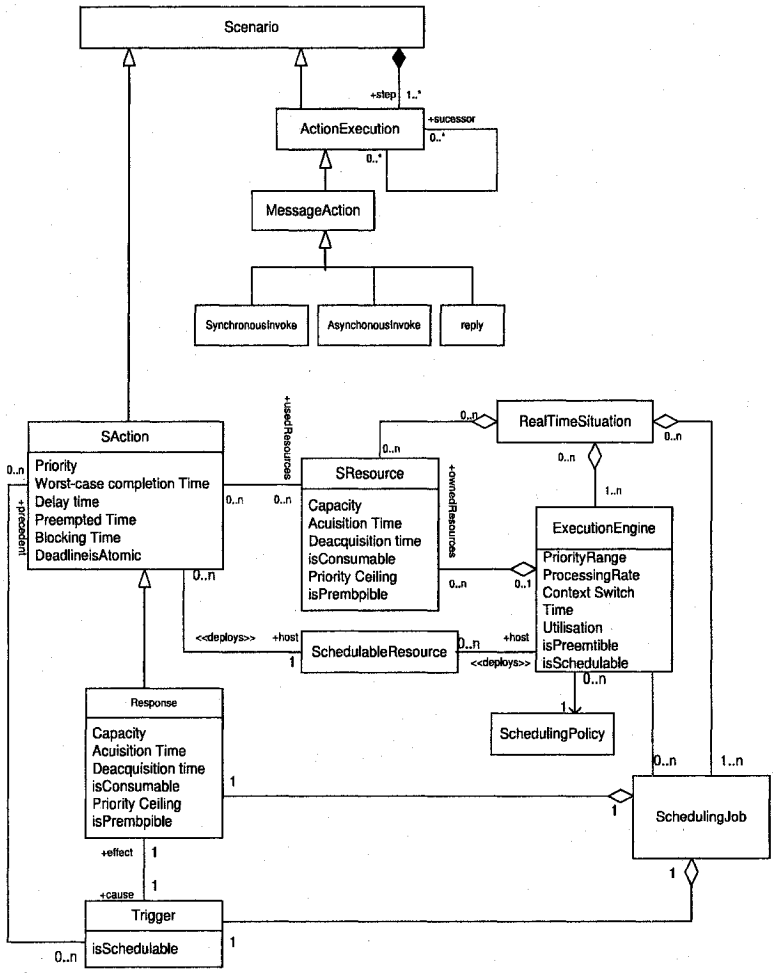

Figure 5.3: Schedulability Analysis Sub-profile Metamodel

representing primitive actions limited for the sake of simplicity to *call*, *reply* and *send* sub-actions.

In this model, actions could be triggered asynchronously or synchronously. In addition, each action is characterized by a nominal priority $\pi(A_i)$, which is inherited from its triggering event. An action executes in the context of an active object $O(A_i)$ and it is assigned to a thread $\Gamma(A_i)$. Furthermore, each sub-action a_{i_j} is characterized by a computation time C_{i_j}. The computation time of an action is the summation of its sub-actions' computation times, i.e. $C(A_i) = \sum_j C_{i_j}$.

The concept of *synchronous set* is defined in [111], [112] for the schedulability analysis purpose and used to compute the worst-case response time of the different actions. The synchronous set of A_i denoted $\Upsilon(A_i)$ is a set of actions that includes A_i and, recursively, each action called synchronously by any action in $\Upsilon(A_i)$. A_i is called the root of $\Upsilon(A_i)$. The cumulative computation time of all the actions in $\Upsilon(A_i)$ is denoted $C(\Upsilon(A_i))$.

The concept of *preemption threshold* [127], is used to bound the blocking factor suffered by an action: in addition to the nominal priority of an action and given the ceiling priorities of each object $\Omega(O(A_i))$ and each thread $\Omega(\Gamma(A_i))$, each action A_i is assigned a preemption threshold $\gamma(A_i)$ satisfying the following constraint:

$$\gamma(A_i) \geq max(\Omega(O(A_i)),\ \ \Omega(\Gamma(A_i))) \tag{5.1}$$

We define a metamodel that encapsulates the main information required for the schedulability analysis as described previously. This metamodel is represented by the UML class diagram shown in Figure 5.4. We define some constraints on this metamodel to reflect the main constraints and assumptions introduced in [111], [112]. These constraints are specified using OCL and shown in Table 5.1. The OCL expression between Line 21 and Line 27,

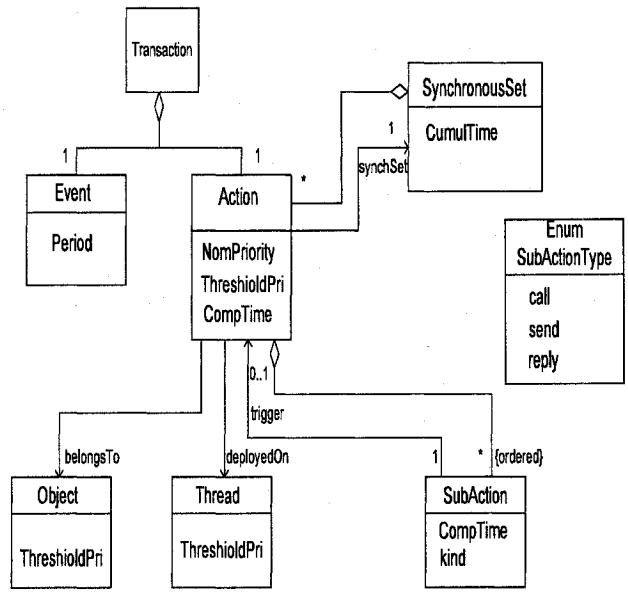

Figure 5.4: Schedulability Analysis Metamodel

for example, specifies the constraint on the preemption threshold of an action expressed in Equation 5.1. The implementation of the model transformation, which is defined in the next section, generates an instance of this metamodel. Such instances are task models expected in input by the schedulability analysis tool.

5.2.3 Model Transformation

In this section, we define a model transformation allowing for the derivation of models used for schedulability analysis from UML/SPT models. This transformation is essentially a mapping of the main concepts and information defined in the source metamodel to the concepts in the target metamodel. This mapping is outlined in Table 5.2. The main features of our model transformation are the following:

- The concept of *SchedulingJob* encapsulates a pair composed of a *Trigger* and the system *Response* to this trigger. This corresponds to a system transaction as defined in the target schedulability analysis metamodel. Consequently, each instance of a *SchedulingJob* is mapped into an instance of a *Transaction.*

- The *SchedulingJob* metaclass has a composition relationship with a pair consisting of a trigger and a system response. These are mapped to a corresponding instance of

```
1  package SAMM
2  context Action inv:
3    self.allInstances-> forAll(a : Action|
4        let np : int = a.NomPriority in:
5        a.SubAction-> forAll(sa : SubAction|
6            sa.kind = #call
7            implies
8            np = sa.trigger.NomPriority))
9  context Action inv:
10   self.allInstances-> forAll(a : Action|
11       let np : int = a.NomPriority in:
12       a.SubAction-> forAll(sa : SubAction|
13           np >= sa.trigger.NomPriority))
14 context Action inv:
15   self.allInstances-> forAll(a : Action|
16       let thr : Thread = a.deployedOn in:
17       a.SubAction-> forAll(sa : SubAction
18           sa.kind = #call
19           implies
20           thr = sa.trigger.deployedOn))
21 context Action inv:
22   self.allInstances-> forAll(a : Action|
23       (a.ThresholdPri ≥
24       a.blongsTo.ThresholdPri)
25        and
26       (a.ThresholdPri ≥
27       a.deployedOn.ThresholdPri))
```

Table 5.1: Schedulability Analysis Metamodel Constraints

SAProfile Concept	Schedulability Analysis Concept
RealTimeSituation	Transaction Set
Trigger	Event
Response	Action
SAction	Action
ActionExecution	SubAction
Scheduling Resource	Thread
SAPriority tag	Nominal Priority
Ceiling Priority tag	Threshold Priority

Table 5.2: SAProfile and the Schedulability Analysis Metamodel Concept Mapping

the concept Event and an instance of the concept *Action* in the target model. This *Action* instance is the root action of the system transaction as defined in the target metamodel.

- *SAction* is a behavior characterized by its own required QoS characteristics. Each instance of *SAction* in the source model is mapped to an *Action* in the target model.

- As shown in Figure 5.3, *Response* is a subclass of *SAction* which is, in turn, a subclass of *Scenario*. Consequently they are composed of a sequence of *ActionExecution*s. The nested *ActionExecution*s are mapped to *SubAction* of the *Action* corresponding to their enclosing *SAction* or *Response*.

- An instance of the *SynchronousSet* is created for each created *Action*. It contains a sequence of references to its corresponding *Action* as well as to each *Action* instance called synchronously.

5.3 Model Transformation Prototype

We present here a proof of concept for the model transformation discussed in the previous section. We provide, respectively, an ATL- and an XML-based prototype implementation of the model transformation.

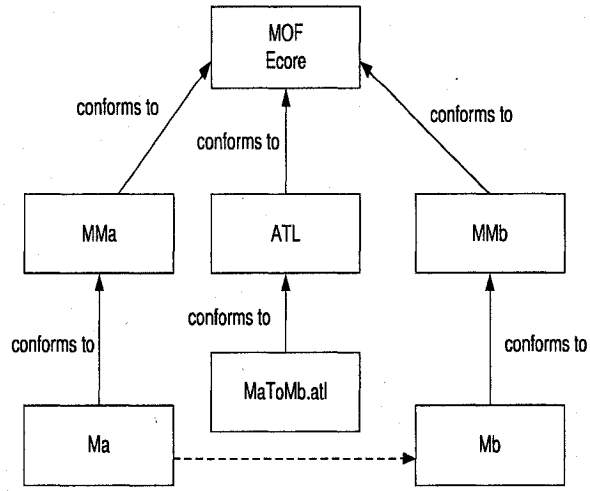

Figure 5.5: ATL Transformation Pattern

5.3.1 Implementation using ATL

The ATLAS Transformation Language (ATL) [59] [62] is a model transformation language and it is a part of the Atlas Model Management Architecture (AMMA) platform [13]. ATL was developed in response to the OMG MOF2.0 /QVT RFP [86]. ATL is used in the transformation pattern shown in Figure 5.5. According to this pattern, ATL allows to transform a source model Ma, that conforms to a source metamodel MMa, into a target model Mb, that conforms to a target metamodel MMb. The transformation is also a model conforming to ATL metamodel. The different metamodels conform, in turn, to a metametamodel such as MOF [93] or Ecore [17].

ATL is a hybrid language allowing a mixture of declarative and imperative programming styles. In addition to the specification of mappings between source and target model elements, ATL provides imperative constructs, which help in specifying some mappings that are difficult to express in a declarative fashion. In the following, we give a succinct overview of the main features of ATL:

- *ATL Units:* ATL provides the developer with three kinds of units. ATL modules are the main ATL units and allow to specify model-to-model transformations. In addition, ATL allows to compute a primitive data type value from a model using ATL Queries. Finally, ATL allows to develop ATL libraries, which enable modularity and code reuse in ATL development. ATL libraries can be imported from the different

ATL units.

- *Rules:* These are the main constructs of an ATL model transformation. ATL offers two kinds of rules correspondingly to its two programming styles. The first kind of rules are called *matched* rules and correspond to the declarative style. The matched rules specify a mapping between a source model element (source pattern) and the corresponding target model element (target pattern) as well as how the latter is initialized. The second kind of rules are *called* rules and correspond to the imperative style. Called rules have, optionally, parameters and do not have a match source model. Called rules should be explicitly called from the imperative code section of another rule.

- Helpers: These are ATL subroutines and are used to factorize and reuse the code: they are defined once and used throughout the transformation. Since target models are not navigable in ATL, helpers can only be defined on an OCL type or a source model element type. This is the helper context. ATL helpers can be either operation helpers or attribute helpers. They both have a context, a name, input parameters (except for the attribute helpers) and return type. OCL expressions are used to specify the value returned. Helpers can be recursively defined.

Within Eclipse open development platform [26], ATL is an Eclipse project and it is part of the Model-to-Model (M2M) eclipse project [106]. M2M is a subproject of the eclipse top-level Eclipse Modeling Project [105]. ATL Integrated Development Environment (IDE) is an Eclipse plug-in build on top of Eclipse Modeling Framework (EMF) [17]. ATL IDE allows diverse operations including the injection operation between different metamodels in addition to the traditional operations of edition, execution and debugging. We have used the Eclipse ATL IDE to develop our prototype implementation for the transformation discussed in the previous section. It consists mainly in a specification of the source and target metamodels using KM3 and an ATL module specifying the model transformation. These are presented in the following section.

5.3.2 Metamodel Definition in KM3

The Kernel MetaMetaModel (KM3) is a domain specific language used for the specification of metamodels [45] [60]. KM3 provides a text-based concrete syntax that is used to code metamodels. It is a simple and Java-like syntax, which simplifies writing metamodel specifications. Such KM3 specifications can then be injected into Ecore using the Eclipse ATL IDE tool. We have specified the source and target metamodels using KM3. These specifications are shown in Figure 5.6 and Figure 5.7, respectively. We have then used the KM3 to Ecore injection facility provided in the ATL IDE to transform these specifications to the EMF format (.ecore) based on the Ecore Metamodel [17]. Figure 5.8 (a) and Figure 5.8 (b) show, respectively, the source and target metamodels using Ecore and which are manipulated by the ATL rules.

5.3.3 Model Transformation in ATL

The model transformation is implemented by an ATL module, which is composed, mainly, of the following two matched transformation rules and helper operations:

- ***SchedJob2Transaction***: This ATL rule, shown in Figure 5.9, maps an instance of the *SchedulingJob* concept to a corresponding *Transaction* instance and creates an instance of *Event* and *Response*. The bindings allow to initialize the features (attributes, references, etc.) of the created instances using the values of the features in the corresponding instance in the source model. An instance of the *SynchronousSet* is created and initialized. The imperative block of the rule computes the computation time of the system root action and completes the synchronous set using the helper operation *getSynchSet()*.

- ***SAction2Action***: This rule, shown in Figure 5.10, creates an instance of *Action* for each *SAction* instance in the source model and maps its nested *ActionExecutions* to *SubAction*. This rule creates also the corresponding *SynchronousSet* for the *Action*.

- We define four ATL helper operations used in the previous matched rules, which are shown in Figure 5.11. The helper *getActionExecutions()* returns the sequence

```
1  package UMLSPT{
2   abstract  class Scenario{
3         attribute name : String;
4         reference step[1-*] container: ActionExecution;
5    }
6    class ActionExecution extends Scenario{
7        attribute rtDuration : Integer;
8        reference predecessor[0-*] : ActionExecution oppositeOf successor;
9        reference successor[1-*] : ActionExecution oppositeOf predecessor;
10       reference effect [0-1]: SAction;
11   }
12   class MessageAction extends ActionExecution {}
13   class SynchronousInvoke extends MessageAction {}
14   class AsynchronousInvoke extends MessageAction {}
15   class Reply extends MessageAction {}
16
17   class SAction extends Scenario {
18        attribute priority : Integer;
19        attribute wcct : Integer;
20        attribute blocking : Integer;
21        reference usedResources [*] : SResource;
22        reference host : SchedulableResource;
23   }
24   class Response extends SAction {
25        attribute priorityCeiling : Integer;
26   }
27   class Trigger {
28      attribute name : String;
29      attribute period : Integer;
30   }
31
32   class SchedulingJob {
33     attribute name : String;
34      reference response[1-1] container: Response;
35      reference trigger[1-1] container : Trigger;
36   }
37
38   class ExecutionEngine {
39       reference ownedResources[0-*] container : SResource;
40       reference schedRes[0-*] : SchedulableResource;
41   }
42   class SchedulableResource {
43      reference host : ExecutionEngine;
44   }
45   class RealTimeSituation {
46       reference sres[0-*] container: SResource;
47       reference exeengine[1-*] container : ExecutionEngine;
48       reference schedJob[1-*] container : SchedulingJob;
49   }
50
51   class SResource {
52   }
53
54 }
55
56 package PrimitiveTypes {
57    datatype Integer;
58    datatype Boolean;
59    datatype String;
60 }
```

Figure 5.6: Source Metamodel in KM3

```
1  package SA {
2    class Transaction {
3        attribute name : String;
4        reference act[1-1] container : Action;
5        reference evt[1-1] container : Event;
6    }
7    class Action {
8       attribute name : String;
9       attribute nomPriority : Integer;
10      attribute ThresholdPri : Integer;
11      attribute compTime : Integer;
12      reference synchSet : SynchronousSet;
13      reference belongsTo : Object;
14      reference deploysOn : Thread;
15      reference subActs[*] container : SubAction;
16   }
17
18   class Event {
19     attribute name : String;
20      attribute period : Integer;
21   }
22   class SynchronousSet {
23      attribute name : String;
24      attribute cumulTime : Integer;
25      reference acts[*]  : Action;
26   }
27
28   class Object {
29      attribute thresholdPri : Integer;
30   }
31
32   class Thread {
33      attribute thresholdPri : Integer;
34   }
35
36   class SubAction {
37     attribute name : String;
38     attribute compTime : Integer;
39     attribute kind : SubActionType;
40     reference trigger[0-1] : Action;
41   }
42
43   enumeration  SubActionType{
44      literal call;
45      literal send;
46      literal reply;
47      literal computation;
48   }
49 }
50
51 package PrimitiveTypes {
52   datatype Integer;
53   datatype String;
54 }
```

Figure 5.7: Target Metamodel in KM3

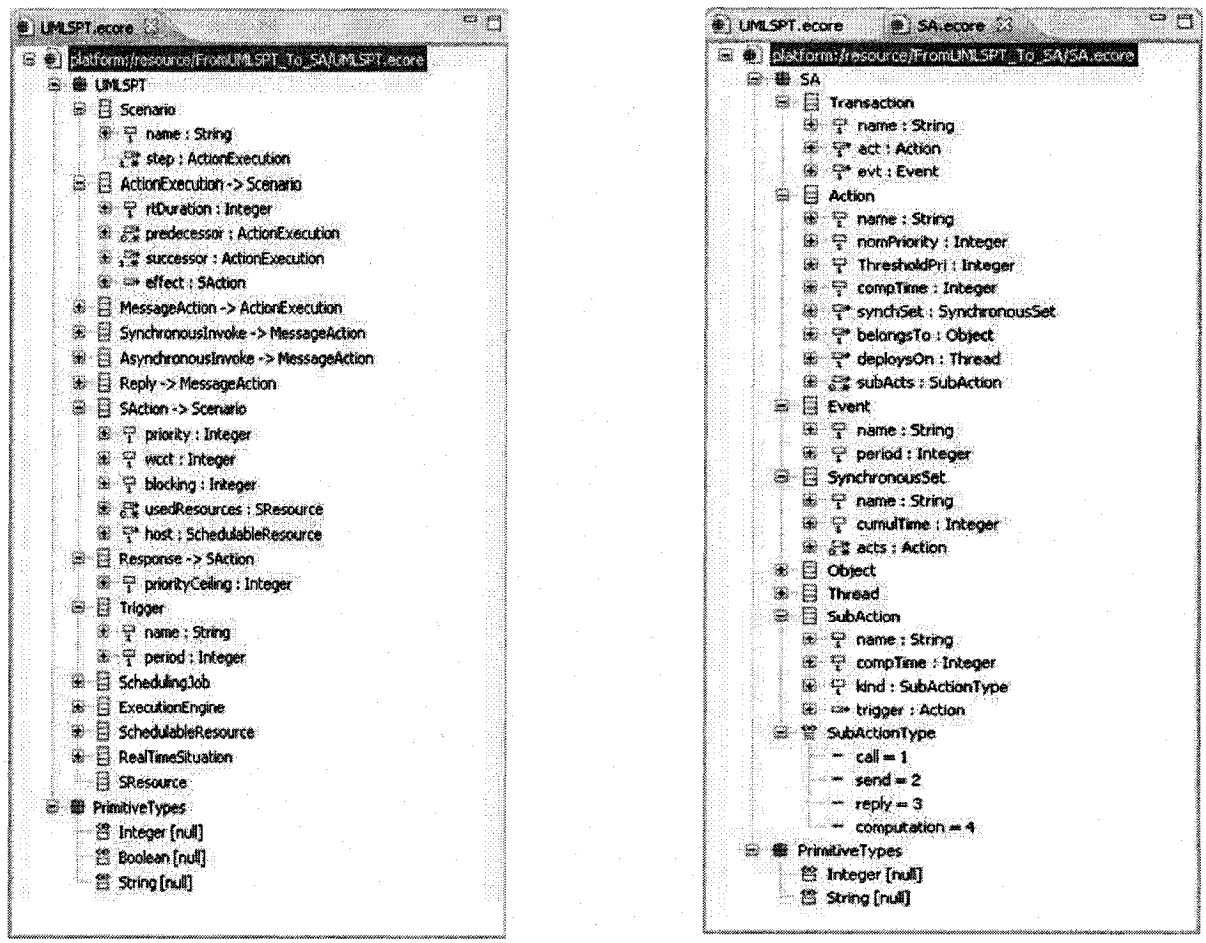

Figure 5.8: Source and Target Metamodels in Ecore

```
1  rule schedJob2Transaction {
2      from
3              s:UMLSPT!SchedulingJob
4
5     to
6             t:SA!Transaction
7             (  name <- s.name ,
8                evt <-ev,
9                act <-ac),
10
11           ev:SA!Event
12                (name <-s.trigger.name,
13                 period <-s.trigger.period),
14
15           ac:SA!Action
16               (name <- s.response.name,
17                nomPriority <-s.response.priority,
18                subActs <-subac,
19                synchSet <-syncset),
20
21          subac: distinct SA!SubAction foreach(e in s.response.
            getActionExecutions()) (
22                name <- e.name,
23                compTime <- e.rtDuration,
24                kind <-e.getSubActionKind(),
25                trigger <-e.effect),
26
27         syncset: SA!SynchronousSet(
28               name <-'SS '+ac.name ,
29               acts <-  syncset.acts->including(ac),
30               cumulTime <- 0
31            )
32
33     do {
34               ac.compTime <- s.response.getCompTime();
35               for(e in s.response.getSynchSet()){
36                  syncset.acts <- syncset.acts->including(thisModule.
                    resolveTemp(e,'t'));
37               }
38          }
39
40 }
```

Figure 5.9: Scheduling Job to Transaction Transformation Rule

of *ActionExecution* composing a *Scenario.* The helper *getSynchSet()* computes the synchronous set associated with each *Action.* The computation time for each *Action* instance based on its *SubActions* is computed using the helper *getCompTime().* Finally, *getSubActionKind()* is used to determine the kind of the *SubAction* depending on the type of the *ActionExecution* used in the source model.

```
1  rule SAction2Action {
2   from
3           s1:UMLSPT!SAction (s1.oclIsTypeOf(UMLSPT!SAction))
4
5
6    to
7        t:SA!Action
8                ( name <- s1.name,
9                  nomPriority <-s1.priority,
10                 subActs <-subac,
11                 synchSet <- syncset),
12
13     subac: distinct SA!SubAction foreach(e in s1.getActionExecutions())(
14                 name <- e.name,
15                 compTime <- e.rtDuration,
16                 kind <-e.getSubActionKind(),
17                 trigger <- e.effect),
18
19
20     syncset: SA!SynchronousSet(
21                name <- 'SS '+t.name,
22                acts <-  syncset.acts->including(t)     --t
23              )
24
25     do {
26               t.compTime <- s1.getCompTime();
27               for(e in s1.getSynchSet()){
28                  syncset.acts <- syncset.acts->including(thisModule.
                    resolveTemp(e,'t'));
29               }
30     }
31
32 }
```

Figure 5.10: SAction to Action Transformation Rule

```
1  helper context UMLSPT!SAction def : getCompTime() : Integer =
2      self.getActionExecutions()->iterate(e; sum : Integer = 0|
3           if true
4           then sum + e.rtDuration
5           else sum + 0
6           endif
7          );
8
9  helper context UMLSPT!SAction def : getSynchSet() : Sequence(UMLSPT!
   SAction) =
10     self.getActionExecutions()->iterate(e; list : Sequence(UMLSPT!SAction)
       = Sequence{}|
11           if(e.oclIsTypeOf(UMLSPT!SynchronousInvoke))
12           then (list->including(e.effect))->union(e.effect.getSynchSet())
13           else list->excluding(e.effect)
14           endif);
15
16 helper  context UMLSPT!Scenario def : getActionExecutions() : Sequence(
   UMLSPT!ActionExecution)  =
17     self.step.asSequence() ;
18
19 helper context UMLSPT!ActionExecution def : getSubActionKind() : SA!
   SubActionType =
20     if self.oclIsTypeOf(UMLSPT!SynchronousInvoke)
21     then  #call
22     else if self.oclIsTypeOf(UMLSPT!AsynchronousInvoke)
23            then #send
24            else if self.oclIsTypeOf(UMLSPT!Reply)
25                   then #reply
26                   else #computation
27                   endif
28            endif
29     endif;
```

Figure 5.11: ATL Helpers

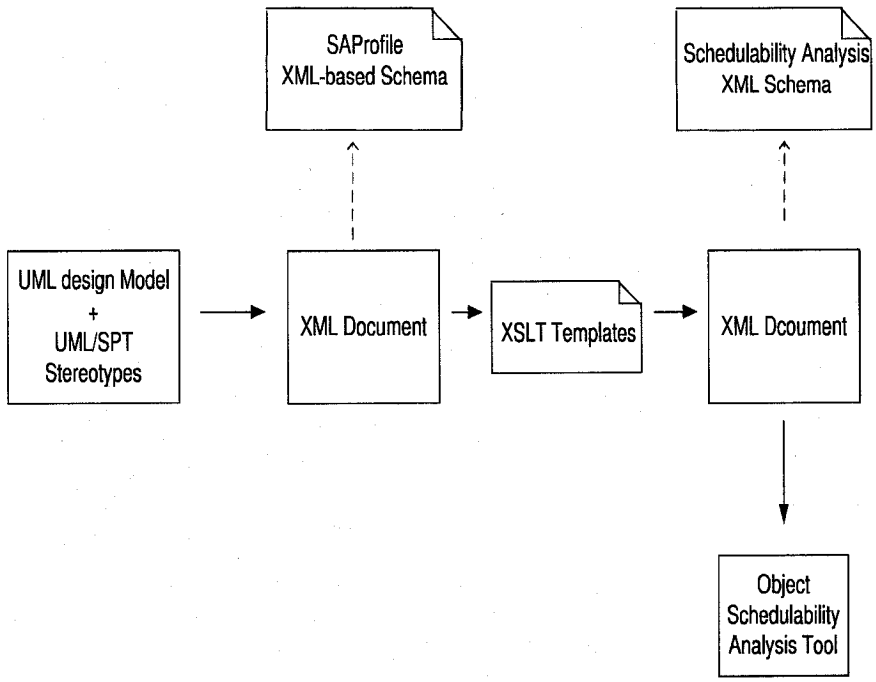

Figure 5.12: XML-based Transformation Process

5.3.4 XML-based Implementation

In this section, we overview our prototype implementation of the transformation process using XML-based technologies. We give an XML schema for the source metamodel and an XML schema for the target metamodel. We specify the model transformation using XSLT template rules, allowing for the transformation of XML documents valid with respect to the source XML schema to XML documents valid with respect to the target XML schema. The latter can then be accessed and manipulated by the schedulability analysis tool. This prototype implementation is outlined in Figure 5.12.

5.3.5 XML Schema for the Metamodels

We have defined an XML schema that corresponds to this compiled metamodel. This schema, shown in Figure 5.13, defines the different XML elements and XML attributes that a valid XML document representing a model as input to our transformation may have. We have included one type for each entity in the compiled metamodel that is relevant for the transformation detailed in the next section. The type `SchedulingJobType` corresponds to the concept of *SchedulingJob* in the metamodel. An XML element of this type should con-

tain two corresponding child XML elements, `TriggerType` and `ResponseType`, respectively. The type `ScenarioType` defines the XML elements corresponding to the *SAction* concept. An XML document using this schema will be presented in Section 5.4. This schema has been validated using W3C's XML schema validation tool, XSV 2.10-1 [126].

We use the XML schema shown in Figure 5.14, which corresponds to our target metamodel. This schema is used by XML documents representing models that are input to the schedulability analysis tool. The schema defines mainly the types for XML elements and XML attributes corresponding to the entities and information in the metamodel. For instance, the type `TransactionType` corresponds to the metaclass *Transaction* in our metamodel. An XML element of this type should contain two child XML elements of type `EventType` and `ActionType` corresponding to the metaclasses *Event* and *Action*, respectively, in the metamodel. An example of an XML document using this schema is given in Section 5.4. This schema has been validated using XSV.

5.3.6 Model Transformation using XSLT

Given the XML schemas corresponding to the source and target metamodels, the transformation rules can be expressed using XSL Transformation (XSLT) [125]. We use the templates rules defined in the XSL stylesheet given in Figure 5.15. The template rules are used to transform an XML document, representing the schedulability analysis information and valid with respect to the XML schema presented in Section 5.3.5, into XML documents representing task models as anticipated by the schedulability analysis, i.e valid with respect to the schedulability schema presented in Section 5.3.5.

The stylesheet specifies how to translate the XML elements defined in the name space `http://SAProfileSchema` into XML elements defined in the `http://SchedAnalysis` name space, which are defined in the XML schemas presented in Section 5.3.5. The first template rule, for example, specifies how the XML element `SchedJob` should be transformed into a `Transaction` XML element, and recursively, that its child XML elements `SATrigger` and `SAReponse` and its descendant XML elements `SAction` should be transformed using the corresponding templates rules. These are specified in the same stylesheet. The template

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3              targetNamespace="http://SAProfileSchema"
4              xmlns:sap="http://SAProfileSchema">
5
6
7  <!--SATriger-->
8  <xsd:complexType name="TriggerType">
9          <xsd:attribute name="name" type="xsd:string"/>
10         <xsd:attribute name="Period" type="xsd:integer"/>
11 </xsd:complexType>
12 <!-- -->
13
14 <xsd:simpleType name="MessageActionType">
15   <xsd:restriction base="xsd:string">
16     <xsd:enumeration value="syncInvoke"/>
17     <xsd:enumeration value="asyncInvoke"/>
18     <xsd:enumeration value="reply"/>
19   </xsd:restriction>
20 </xsd:simpleType>
21
22
23 <xsd:complexType name="ActExecType">
24         <xsd:attribute name="id" type="xsd:ID"/>
25         <xsd:attribute name="name" type="xsd:string"/>
26         <xsd:attribute name="RTduration" type="xsd:integer"/>
27         <xsd:attribute name="succ" type="xsd:IDREF"/>
28         <xsd:attribute name="matype" type="sap:MessageActionType"
           use="optional"/>
29         <xsd:attribute name="effect" type="xsd:IDREF"/>
30 </xsd:complexType>
31
32 <xsd:complexType name="ScenarioType">
33         <xsd:choice minOccurs="0" maxOccurs="unbounded">
34                 <xsd:element name="ActionExec" type="sap:ActExecType"/>
35         </xsd:choice>
36         <xsd:attribute name="id" type="xsd:ID"/>
37         <xsd:attribute name="execHost" type="xsd:string"/>
38         <xsd:attribute name="name" type="xsd:string"/>
39         <xsd:attribute name="priority" type="xsd:integer"/>
40 </xsd:complexType>
41
42 <xsd:complexType name="ResponseType">
43         <xsd:choice minOccurs="0" maxOccurs="unbounded">
44                 <xsd:element name="ActionExec" type="sap:ActExecType"/>
45                 <xsd:element name="SAction" type="sap:ScenarioType"/>
46         </xsd:choice>
47         <xsd:attribute name="id" type="xsd:ID"/>
48         <xsd:attribute name="execHost" type="xsd:string"/>
49         <xsd:attribute name="name" type="xsd:string"/>
50         <xsd:attribute name="priority" type="xsd:integer"/>
51 </xsd:complexType>
52
53 <xsd:complexType name="SchedulingJobType">
54         <xsd:sequence>
55                 <xsd:element name="SATrigger" type="sap:TriggerType"/>
56                 <xsd:element name="SAResponse" type="sap:ResponseType"/>
57         </xsd:sequence>
58         <xsd:attribute name="name" type="xsd:string"/>
59 </xsd:complexType>
60
61 <xsd:element name="SchedJob" type="sap:SchedulingJobType"/>
62
63
64 </xsd:schema>
```

Figure 5.13: XML Schema for the Source Metamodel

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://SchedAnalysis" xmlns:sa="http://SchedAnalysis">
3          <xsd:complexType name="SAObjModelType">
4                  <xsd:choice minOccurs="0" maxOccurs="unbounded">
5                          <xsd:element name="Transaction"
                           type="sa:TransactionType" />
6                          <xsd:element name="SynchronousSet"
                           type="sa:SynchronousSetType" />
7                          <xsd:element name="Object" type="sa:ObjectType" />
8                          <xsd:element name="Thread" type="sa:ObjectType" />
9                  </xsd:choice>
10         </xsd:complexType>
11         <xsd:element name="SAObjModel" type="sa:SAObjModelType" />
12         <xsd:complexType name="TransactionType">
13                 <xsd:sequence>
14                         <xsd:element name="Event" type="sa:EventType" />
15                         <xsd:choice minOccurs="0" maxOccurs="unbounded">
16                                 <xsd:element name="Action"
                                   type="sa:ActionType" />
17                         </xsd:choice>
18                 </xsd:sequence>
19                 <xsd:attribute name="name" type="xsd:string" />
20                 <xsd:attribute name="id" type="xsd:ID" />
21         </xsd:complexType>
22         <xsd:complexType name="EventType">
23                 <xsd:attribute name="name" type="xsd:string" />
24                 <xsd:attribute name="Period" type="xsd:integer"
                   use="optional" />
25         </xsd:complexType>
26         <xsd:complexType name="subActionsType">
27                 <xsd:choice minOccurs="0" maxOccurs="unbounded">
28                         <xsd:element name="subAct" type="sa:subActionType"
                           />
29                 </xsd:choice>
30         </xsd:complexType>
31         <xsd:complexType name="ActionType">
32                 <xsd:choice minOccurs="0" maxOccurs="unbounded">
33                         <xsd:element name="subAction"
                           type="sa:subActionType" />
34                 </xsd:choice>
35                 <xsd:attribute name="id" type="xsd:ID" />
36                 <xsd:attribute name="name" type="xsd:string" />
37                 <xsd:attribute name="NomPri" type="xsd:integer" />
38                 <xsd:attribute name="synchSet" type="xsd:IDREF"
                   use="optional" />
39                 <xsd:attribute name="belongsTo" type="xsd:IDREF"
                   use="optional" />
40                 <xsd:attribute name="deployedOn" type="xsd:IDREF"
                   use="optional" />
41         </xsd:complexType>
42         <xsd:complexType name="synchSetElementType">
43                 <xsd:attribute name="ref" type="xsd:IDREF" />
44         </xsd:complexType>
45         <xsd:complexType name="SynchronousSetType">
46                 <xsd:choice minOccurs="0" maxOccurs="unbounded">
47                         <xsd:element name="action"
                           type="sa:synchSetElementType" />
48                 </xsd:choice>
49                 <xsd:attribute name="name" type="xsd:string" />
50                 <xsd:attribute name="id" type="xsd:ID" />
51                 <xsd:attribute name="CumulTime" type="xsd:integer" />
52         </xsd:complexType>
53         <xsd:simpleType name="subActionKindType">
54                 <xsd:restriction base="xsd:string">
55                         <xsd:enumeration value="call" />
56                         <xsd:enumeration value="send" />
57                         <xsd:enumeration value="reply" />
58                         <xsd:enumeration value="computation" />
59                 </xsd:restriction>
60         </xsd:simpleType>
61         <xsd:complexType name="subActionType">
62                 <xsd:attribute name="name" type="xsd:string" />
63                 <xsd:attribute name="trigger" type="xsd:IDREF"
                   use="optional" />
64                 <xsd:attribute name="compTime" type="xsd:integer" />
65                 <xsd:attribute name="kind" type="sa:subActionKindType" />
66         </xsd:complexType>
67         <xsd:complexType name="ObjectType">
68                 <xsd:attribute name="name" type="xsd:string" />
69                 <xsd:attribute name="id" type="xsd:ID" />
70                 <xsd:attribute name="ThresholdPri" type="xsd:integer"
                   use="optional" />
71         </xsd:complexType>
72         <xsd:complexType name="ThreadType">
73                 <xsd:attribute name="name" type="xsd:string" />
74                 <xsd:attribute name="id" type="xsd:ID" />
75                 <xsd:attribute name="ThresholdPri" type="xsd:integer"
                   use="optional" />
76         </xsd:complexType>
77 </xsd:schema>
```

Figure 5.14: XML Schema for the Target Metamodel

rule matching the `SAResponse`, for example, generates an `Action` XML element. The information held in the attributes associated with the generated XML elements are extracted from the initial XML elements using XPath expressions. The XPath expression `@priority`, for example, is used in the `SAResponse` template rule to extract the value of the `priority` attribute and assign its value to the `NomPri` attribute associated with the `Action` element.

5.4 Implementation Applied on an Illustrative UML/SPT Model

In order to illustrate the model transformation presented in this chapter, we consider an example system, which was introduced in [111] [112]. We provide a UML model for this system. This model consists mainly of: (1) a UML collaboration diagram representing the overall behavior of the system and the synchronization between the actions in the different interacting objects, (2) a deployment diagram showing the object-to-thread allocation, and (3) sequence diagrams detailing the end-to-end transactions in the system in response to each external event. As for the real-time requirements and schedulability information, we use different UML/SPT stereotypes to represent them.

The overall dynamic behavior of the system is modeled by the UML collaboration diagram annotated with UML/SPT stereotypes shown in Figure 5.16. This diagram shows the different objects composing the system and their communications. It shows also that the system reacts to three external and periodic events, which are represented using the stereotype ≪`SATrigger`≫. For example, the external event that triggers the action *A1()* is stereotyped with ≪`SATrigger`≫, which has the tagged value {`RAT=(periodic,60,'ms')`} to indicate a period of 60 ms. The actions are modeled using the stereotype ≪`SAaction`≫ and their priorities are modeled using the tag `SApriority`. The priority of the action *A1()* is given using {`SApriority =10`} tag value. These different stereotypes are defined in the schedulability analysis sub-profile, SAProfile. We use the stereotypes ≪`CRsynch`≫ and ≪`CRasynch`≫ from the concurrency sub-profile, *RTconcurrencyModeling*, to distinguish between synchronous and asynchronous actions, respectively. For example, the action *A5()*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
3                 xmlns:pix="http://SAProfileSchema"
4                 xmlns:pox="http://SchedAnalysis">
5
6          <xsl:template match="pix:SchedJob">
7              <pox:Transaction name="{@name}">
8                  <xsl:apply-templates select="SATrigger"/>
9                  <xsl:apply-templates select="SAResponse"/>
10                 <xsl:apply-templates select="//SAction"/>
11             </pox:Transaction>
12         </xsl:template>
13
14
15         <xsl:template match="SATrigger">
16             <Event name="{@name}" Period="{@Period}" />
17         </xsl:template>
18
19
20         <xsl:template match="SAResponse">
21             <Action id="{@id}" name="{@name}" NomPri="{@priority}"
               belongsTo="{@execHost}">
22                <xsl:apply-templates select="ActionExec"/>
23             </Action>
24         </xsl:template>
25
26
27         <xsl:template match="ActionExec">
28             <xsl:choose>
29               <xsl:when test="@effect">
30                   <xsl:choose>
31                      <xsl:when test="@matype = 'asyncInvoke'">
32                          <subAction id="{@id}" name="{@name}"
                            compTime="{@RTduration}" trigger="{@effect}"
                            kind="send"/>
33                      </xsl:when>
34                      <xsl:when test="@matype = 'syncInvoke'">
35                          <subAction id="{@id}" name="{@name}"
                            compTime="{@RTduration}" trigger="{@effect}"
                            kind="call"/>
36                      </xsl:when>
37                   </xsl:choose>
38               </xsl:when>
39               <xsl:otherwise>
40                 <xsl:choose>
41                      <xsl:when test="@matype = 'reply'">
42                            <subAction id="{@id}" name="{@name}"
                              compTime="{@RTduration}" kind="reply"/>
43                      </xsl:when>
44                      <xsl:otherwise>
45                          <subAction id="{@id}" name="{@name}"
                            compTime="{@RTduration}" kind="computation"/>
46                      </xsl:otherwise>
47                 </xsl:choose>
48               </xsl:otherwise>
49             </xsl:choose>
50         </xsl:template>
51
52         <xsl:template match="SAction">
53                 <Action id="{@id}" name="{@name}" NomPri="{@priority}"
                   belongsTo="{@execHost}">
54                    <xsl:apply-templates select="ActionExec"/>
55                 </Action>
56         </xsl:template>
57 </xsl:stylesheet>
```

Figure 5.15: Model Transformation XSLT Templates

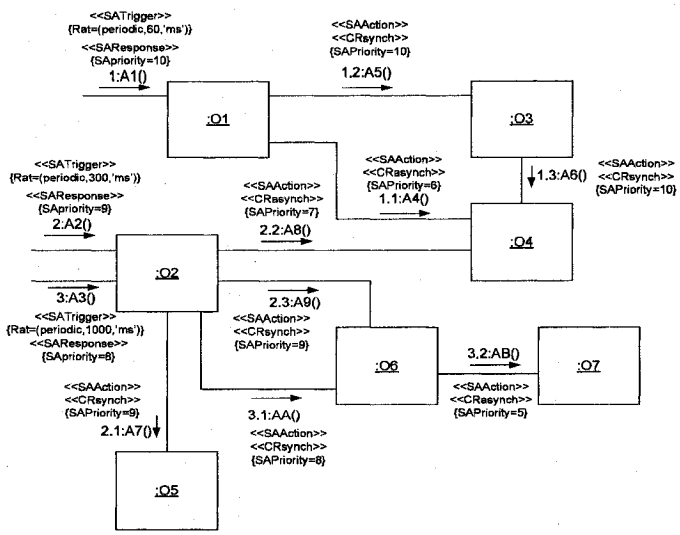

Figure 5.16: UML/SPT-annotated UML Collaboration Diagram

executed by the object *O3* is annotated with the ≪CRsynch≫ stereotype. This means that it is executed following a synchronous message sent by the object *O1* as a result of the execution of the action *A1()* as shown in Figure 5.16. The resulting action *A6()* executed by the object *O4* is also synchronous, stereotyped with ≪CRsynch≫. Consequently, using the concepts defined in Section 5.2.2, the actions *A1()*, *A5()* and *A6()* are the elements of the synchronous set of *A1()*.

The deployment diagram, depicted in Figure 5.17, shows the assignment of the different objects to the concurrent threads and the deployment of the latter on the CPU. This is represented by a node annotated with the stereotype ≪SAEngine≫, and the different threads are represented by the stereotype ≪SASchedRes≫. Both stereotypes are defined in the schedulability analysis sub-profile, *SAProfile*. The deployment of the threads on the CPU is modeled using the stereotype ≪GRMdeploys≫ defined in the UML/SPT *General Resource Model* framework. The assignment of the objects to the threads is modeled using a relationship stereotyped with ≪SAusedHost≫, defined in the *SAProfile* sub-profile.

There are three end-to-end transactions in the system corresponding to the three external events. These transactions are described using the UML sequence diagrams annotated with UML/SPT stereotypes, and are shown in Figure 5.18, Figure 5.19 and Figure 5.20. For

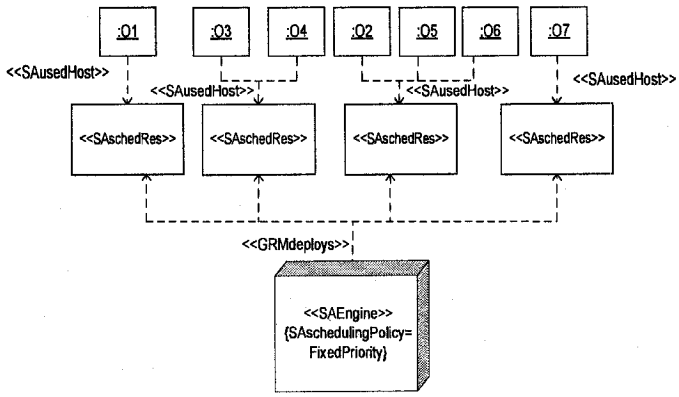

Figure 5.17: UML/SPT-annotated Deployment Model

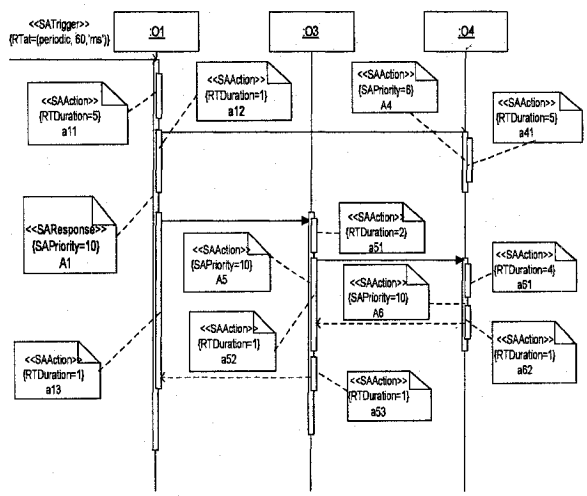

Figure 5.18: Transaction 1 Model

example, the sequence diagram shown in Figure 5.18 models the transaction associated with the first external event. This is modeled using the stereotype ≪`SATrigger`≫ and its arrival pattern (periodicity) is modeled using the tag value {`RAT=(periodic,60,'ms')`}. The stereotype ≪`SAResponse`≫ is used to model the root action in the transaction while ≪`SAaction`≫ is used to model the subsequent actions and/or any of their nested actions (sub-actions). The priorities of the actions are modeled using the tag value `SAPriority` while the computation time of an action is modeled using the tag value `RTDuration` from the *RTtimeModeling* sub-profile of UML/SPT.

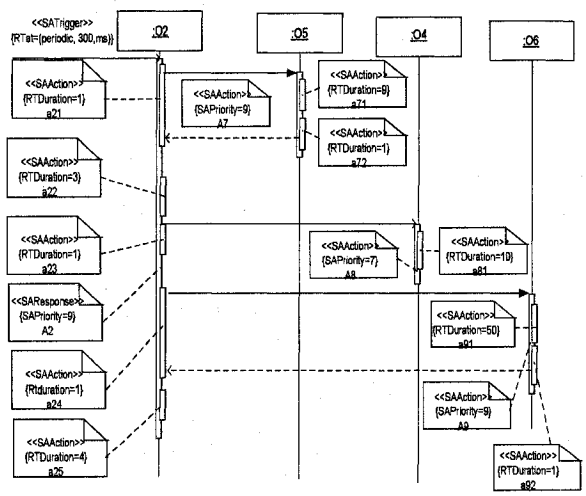

Figure 5.19: Transaction 2 Model

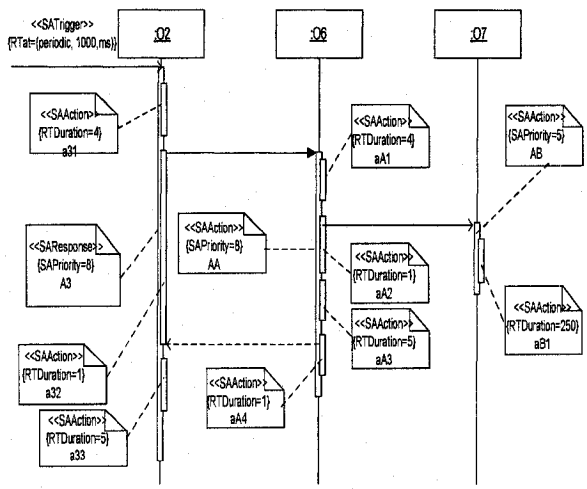

Figure 5.20: Transaction 3 Model

5.4.1 Using the ATL Transformation

The model showed in Figure 5.21 is in the Ecore format. It represents the schedulability analysis information in our example system. This model is valid with respect to the source metamodel described in the previous section. The transformation is then run using an ATL launch configuration in the Eclipse ATL IDE. The execution of the model transformation generates model shown in Figure 5.22. We highlight the different actions along with their sub-actions corresponding to the first transaction and the corresponding synchronous set.

5.4.2 Using the XML-based Transformation

The XML document that gathers the schedulability analysis information corresponding to the first end-to-end transaction of the previous model is shown in Figure 5.23. This XML document has been validated against the schema defined in Section 5.3.5 using the open source Xerces XML parser from the Apache XML project [104].

We have used the Xalan XSLT processor [103] to execute the transformation. Xalan is an open source Java tool that takes as input the previous XML document and the XSLT stylesheet and generates a new XML document. This is valid with respect to the target schema presented in Section 5.3.5. Figure 5.24 shows the XML document generated by the XSLT processor. This XML document represents the model which is expected as input by the schedulability analysis tool. The UML object diagram shown in Figure 5.25 is a graphical representation of a part of the model corresponding to the transaction triggered by the first external event, where the synchronous set, objects, and threads are omitted for simplicity.

5.4.3 Schedulability Analysis Tool

The model represented by the XML document is then provided as input to the schedulability analysis tool. This tool is a Java implementation of the multi-threaded version of the schedulability analysis technique presented in [112]. The output for example is given in Figure 5.26. The results show that the end-to-end response times associated with each

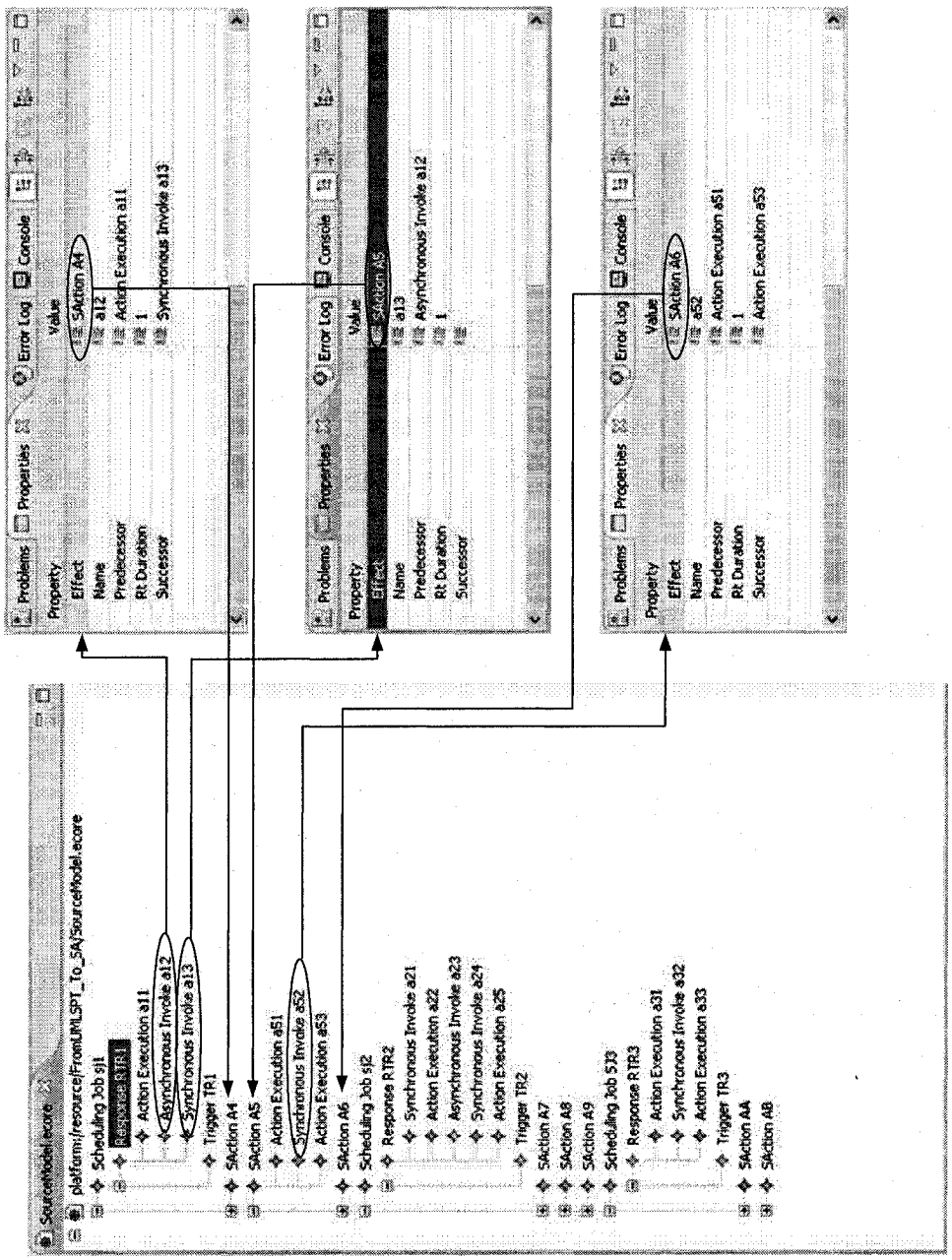

Figure 5.21: Source Model in Ecore

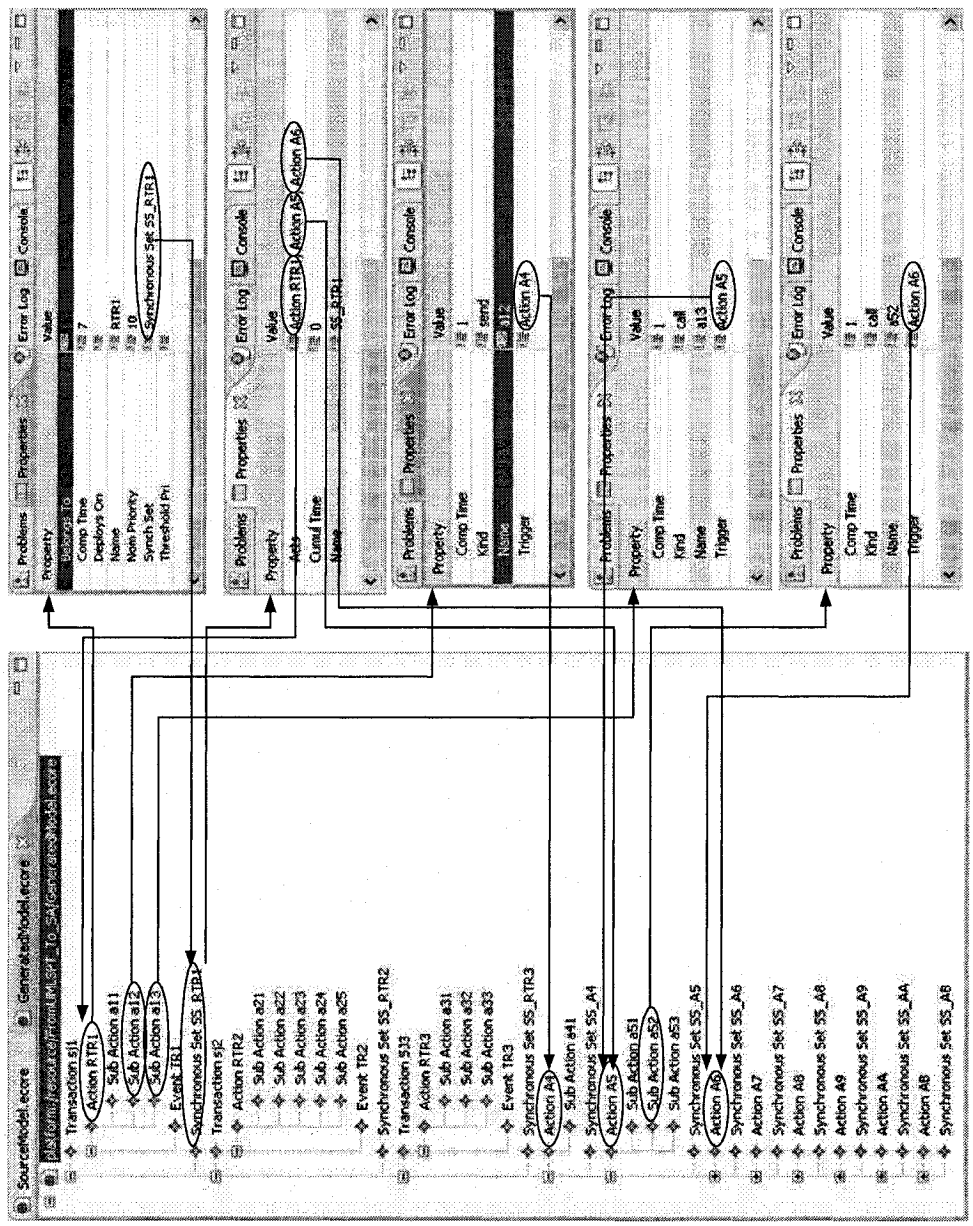

Figure 5.22: Generated Model in Ecore

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <sap:SchedJob name="Job1"
4          xmlns:sap="http://SAProfileSchema"
5          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6          xsi:schemaLocation="http://SAProfileSchema SAProfileSchema.xsd">
7
8      <SATrigger name="Tr1" Period="60"/>
9      <SAResponse name="R Tr1" id="RTr1" priority="10" execHost="O1">
10        <ActionExec name="a11" id=" a11" RTduration="5" succ=" a12"/>
11        <ActionExec name="a12" id=" a12" RTduration="1" succ="_a13"
          matype="asyncInvoke" effect=" A4"/>
12        <ActionExec name="a13" id=" a13" RTduration="1" succ="_a13"
          matype="syncInvoke" effect="_A5"/>
13
14        <SAction name="A4" id=" A4" priority="6" execHost="O4">
15          <ActionExec name="a41" id="_a41" RTduration="5"/>
16        </SAction>
17
18        <SAction name="A5" id=" A5" priority="10" execHost="O3">
19            <ActionExec name="a51" id=" a51" RTduration="2" succ=" a51"/>
20            <ActionExec name="a52" id=" a52" RTduration="1" succ="_a53"
              matype="syncInvoke" effect=" A6"/>
21            <ActionExec name="a53" id="_a53" RTduration="1" matype="reply"
              />
22        </SAction>
23
24        <SAction name="A6" id=" A6" priority="10" execHost="O4">
25            <ActionExec name="a61" id=" a61" RTduration="4" succ=" a62"/>
26            <ActionExec name="a62" id="_a62" RTduration="1" matype="reply"
              />
27        </SAction>
28
29     </SAResponse>
30 </sap:SchedJob>
31
```

Figure 5.23: XML Document for Source Model

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <pox:Transaction xmlns:pox="http://SchedAnalysis"
   xmlns:pix="http://SAProfileSchema" name="Job1">
3          <Event Period="60" name="Tr1" />
4          <Action belongsTo="O1" NomPri="10" name="R Tr1" id="RTr1">
5                  <subAction kind="computation" compTime="5" name="a11"
                   id=" a11" />
6                  <subAction kind="send" trigger="_A4" compTime="1"
                   name="a12" id=" a12" />
7                  <subAction kind="call" trigger="_A5" compTime="1"
                   name="a13" id="_a13" />
8          </Action>
9          <Action belongsTo="O4" NomPri="6" name="A4" id=" A4">
10                 <subAction kind="computation" compTime="5" name="a41"
                   id="_a41" />
11         </Action>
12         <Action belongsTo="O3" NomPri="10" name="A5" id=" A5">
13                 <subAction kind="computation" compTime="2" name="a51"
                   id=" a51" />
14                 <subAction kind="call" trigger="_A6" compTime="1"
                   name="a52" id=" a52" />
15                 <subAction kind="reply" compTime="1" name="a53" id="_a53"
                   />
16         </Action>
17         <Action belongsTo="O4" NomPri="10" name="A6" id=" A6">
18                 <subAction kind="computation" compTime="4" name="a61"
                   id=" a61" />
19                 <subAction kind="reply" compTime="1" name="a62" id="_a62"
                   />
20         </Action>
21 </pox:Transaction>
```

Figure 5.24: Generated XML Document for the target Model

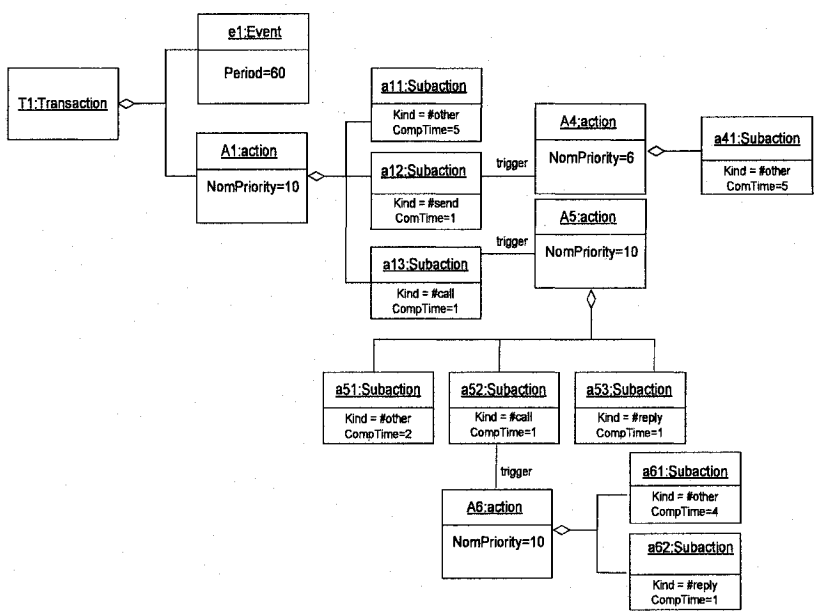

Figure 5.25: Schedulability Analysis Model

Transaction	Job1				
	Instance	Arrival	Finish	Response Time	E2E Response Time
A1	1	0	28	28	28
A4	1	0	161	161	161
	2	60	166	106	
	3	120	171	51	
A5	1	0	21	21	21
A6	1	0	15	15	15
Transaction	Job2				
A2	1	0	59	59	59
A7	1	0	49	49	49
A8	1	0	161	161	161
A9	1	0	108	108	108
Transaction	Job3				
A3	1	0	156	156	156
A10	1	0	146	146	146
A11	1	0	836	836	836

Figure 5.26: Schedulability Analysis Results

transaction.

5.5 Related Work

In this section, we discuss the main related research work. We can distinguish three groups of research. The first group is directly related to the idea of using MDA as an integrating framework for the models used for software design and the models used for the analysis of nonfunctional properties. The second group is focussed on the schedulability analysis of object-oriented design models. The third group includes works on model transformation in general and the languages used to express these transformations.

A significant research work uses MDA as an integrating framework for software modeling, validation and testing. The common objective is to use MDA to bridge the gap between UML design models and several models used by analysis techniques. Gu et al. [46] presented an XML-based transformation of UML models annotated with UML/SPT stereotypes for performance to LQN performance models. Verdickt et al. [124] presented an MDA-based approach where high-level middleware-independent UML models are transformed into middleware-aware UML models. Skene et al. [119] introduced an MDA-based approach

leveraging the UML profiling mechanisms to allow for the derivation of performance analysis models, such as LQN, from UML design models. D'Ambrogio [21] presented a model transformation framework for the validation of software system performance throughout the development cycle. This framework enables transforming software models into layered queuing network models. This approach relies on the principles of the MDA and makes use of the related standards such as MOF, QVT and XML to achieve the objectives of high automation, inter-operability, and finally software system quality. Other MDA-based approaches have been proposed in order to shift traditional testing techniques to the model level [9], [129].

The integration of object-oriented design using UML-RT [116], which is a UML profile that extends UML with concepts defined in ROOM methodology [114], and real-time schedulability analysis has been addressed by Saksena et al. [111] [112]. Gu et al. [47] presented an approach geared to the automatic synthesis of a UML-RT model and its schedulability analysis. The motivation behind these approaches was mainly the application of object-oriented methodologies to manage the increasing complexity of embedded real-time software systems. Although these approaches showed how to adapt and/or apply schedulability analysis techniques on object-oriented models, they did not address the issue of how the task models used in these schedulability analysis techniques could be derived from UML-based design models.

Finally, model transformation is a cornerstone activity in the model-driven development approach and MDA. Many languages can be used to express model transformation, including OCL, scripting languages, XSLT, etc. The object constraint language (OCL) [94] is an integral part of UML that is used to specify constraints on UML models and metamodels. OCL has also been used to express platform independent model transformation [16], [18], [32], [65]. Poress [102] presented a scripting language for model manipulation. This is an extension of Python for the manipulation and transformation of MOF-based models. XML is a common means to represent models mainly through the XMI standard. Poon et al. [101] used XML as an interchange format for real-time data and proposed an XML repository for real-time data in the form of XML documents. This is important for inter-operability between diverse real-time applications. In this work, the premises of a Real-Time Mark Up

language were outlined. This language is defined through an XML schema corresponding to the time-related mechanisms and timed stimulus domain models defined in the time packages of UML/SPT. Consequently, XSLT is an effective way to transform models expressed with XML. XSLT has some drawbacks, however, such as its verbosity, expertise requirement in MOF [93] and XMI [89] and lack of user-friendliness. Peltier et al. [73] presented a model transformation framework, called MTrans, which relies on XSLT to transform models but provides higher abstractions to alleviate XSLT shortcomings. Model transformation technology is maturating with the adoption of OMG QVT standard [88]. ATL it was designed in response to the OMG MOF2.0 /QVT RFP [86]. Although ATL evolved in parallel to the OMG standardization process, it is aligned with the standard [61]. ATL is used in different approaches addressing a variety of issues including model checking [12], model consistency checking [43], architecture migration [8] and web services [11] [98].

5.6 Conclusion

Model-driven architecture is a software development approach that advocates the separation of the business/application logic from the underlying technology used for its implementation. This helps to overcome the complexity in the problem domain and in the technology domain separately. MDA aims at achieving this goal using two key concepts, models and model transformations. In terms of models, MDA distinguishes platform independent models and platform specific models . Model transformations are defined and used to (automatically) generate PSMs from PIMs. Several languages have been proposed and used to express model transformations including OCL, XSLT and QVT.

MDA allows to bridge the semantic gap between the concepts in the problem domain and in the technology domain. This same framework is useful to address the challenging issue of closing the gap between the design modeling concepts and the analysis-specific concepts, which has driven several researchers to use MDA as an integrated framework for software modeling and analysis. Most of this research has been on performance analysis. Our approach targets schedulability analysis, which is important for the validation of real-time

designs. The advantage of an MDA-based approach is to create a tool chain that masks the inherent difficulties of the formal methods used in software validation to the software engineer and consequently helps to increase software quality.

In this chapter, we have discussed a model transformation allowing for the derivation of a task model, suitable for a schedulability analysis technique, from an UML/SPT model. As a proof of concept, we have presented a prototype implementation of this transformation using ATL and XML. The ATL-based implementation consists in a specification of the involved metamodels using KM3 language and the specification of the transformation rules using ATL. As for the XML-based implementation, we have proposed XML schemas corresponding to the metamodels and specified the transformation rules using XSLT. We have illustrated the transformation process with an example using Eclipse ATL IDE for the first implementation and open source tools for the XML-based one.
The solution we have presented in this chapter presents some limitations, however, which can be summarized as follows:

- The transformation takes as input the information modeled using UML/SPT stereotypes. These stereotypes are not models on their own but are closely linked and represented in a UML design model. We assume that the information needed by the transformation is extracted from the UML design model. Very few tools, however, implement the UML/SPT stereotypes, but we expect this to change in the near future with the new UML profile, MARTE [97].

- The feedback given by the schedulability analysis tool is limited to determining if an end-to-end transaction is schedulable or not. This is useful, but a more fine-tuned diagnostic such as which object-to-thread allocation is the cause of a missed deadline would be more informative. In addition, the only way to reflect the results of an analysis into the original design model is through some variables in the UML/SPT stereotypes, which is a UML/SPT intrinsic limitation [10].

Chapter 6

Consistency of UML/SPT Models

UML provides a multitude of diagrams, which can be used to model the structure, the behavior and the deployment of the system under consideration. This is very advantageous to cope with software complexity. On the other hand, these different views may lead to inconsistencies. Moreover, when UML is used to model real-time systems through its profiles for real-time such as the OMG standard, UML/SPT [91] and MARTE [97] for example, new aspects need to be taken into account, namely concurrency, time constraints and schedulability. These aspects may contribute to worsen the consistency issue.

UML's built-in consistency mechanisms are limited to a set of well-formedness rules expressed in the metamodel using OCL. Higher level consistency concepts are, however, not accounted for at the language level. Considering the complexity of a UML/SPT model, which is composed of several UML diagrams and which captures in addition aspects such as concurrency and time constraints using appropriate stereotypes defined in the profile, it is difficult to provide one definition of consistency. An incremental approach to consistency of UML/SPT models, which distinguishes, respectively the syntactic and semantic levels, is more appropriate. The semantic consistency can be further decomposed into behavioral, concurrency-related and time consistency.

In this chapter, we focus particularly on inter-diagram consistency in a UML/SPT model. First, we present a consistency framework for UML/SPT models. In this framework, we define the consistency of UML/SPT models in terms of syntactic consistency and semantic consistency. The latter is defined further in terms of behavioral consistency, concurrency-

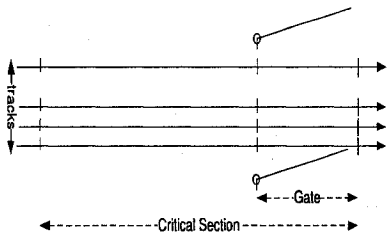

Figure 6.1: Generalized Railroad Crossing System

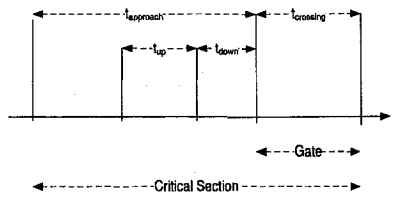

Figure 6.2: Generalized Railroad Crossing Time Constraints

related consistency and time consistency. Second, we focus on time consistency of behavioral diagrams of UML/SPT models, namely statecharts and sequence diagrams. We introduce an approach that relies on schedulability analysis. In order to do so, we show how a UML/SPT-based schedulability analysis model is generated from statecharts and sequence diagrams. Our model transformation approach presented in Chapter 5 is then used to enable appropriate schedulability analysis techniques and consequently check the time consistency of the model.

6.1 Railroad Crossing System Model using UML/SPT

In this section, we introduce a UML/SPT model for the Generalized Railroad Crossing System (GRCS) [50], which we will use throughout this chapter. This system controls a gate in a critical region to protect a railroad crossing as depicted in Figure 6.1. A set of trains can traverse the crossing in parallel using different tracks. The system uses sensors to detect an entering/exiting train to/from the critical region. The GRCS should satisfy certain time requirements as depicted in Figure 6.2. The fastest train takes $\mathtt{t_{approach}}$ to reach the gate after entering the critical section and it takes $\mathtt{t_{crossing}}$ to cross the gate section. A closed gate takes $\mathtt{t_{up}}$ to open fully while an open gate takes $\mathtt{t_{down}}$ to close completely.

Structure View: The class diagram in Figure 6.3 shows the static structure of the design. The system is composed of two concurrent entities `TrackController` and `GateController`, which are annotated with the stereotype ≪`CRconcurrent`≫. They use the passive objects, `TrackHandler` and `Gate`, respectively. `Clock` is a timer, annotated with the stereotype ≪`RTtimer`≫, and it is used by the `TrackHandler` entities to keep track of the time progress.

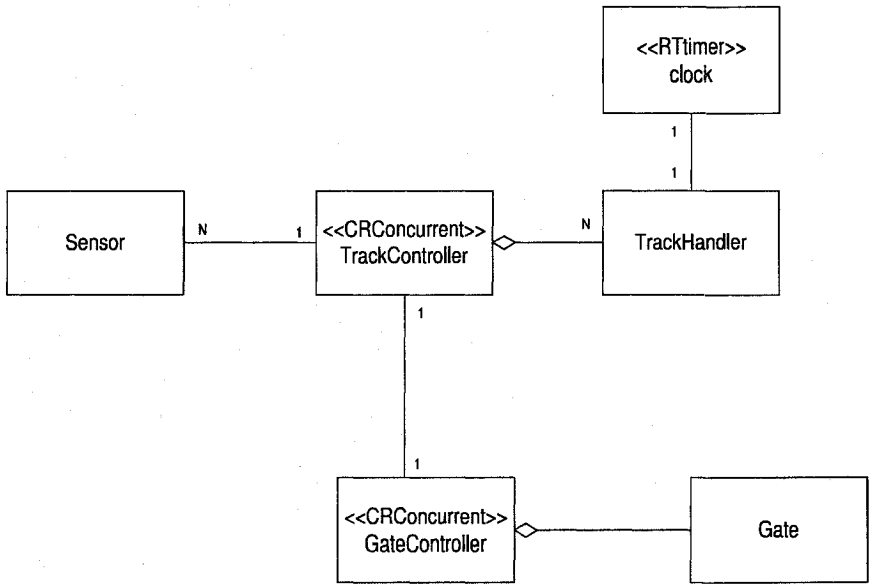

Figure 6.3: Generalized Railroad Crossing Structure View

The entity `Sensor` represents the sensors.

Behavior View: The most important interactions between the entities defined in the system structure along with their time requirements are given in the sequence diagrams shown in Figure 6.4, Figure 6.5, Figure 6.6 and Figure 6.7. The detailed design is modeled using UML statecharts. These describe the internal behavior of each entity. The statecharts corresponding to the `TrackController`, `TrackHandler`, `GateController` and `Gate` are depicted in Figure 6.9, Figure 6.8, Figure 6.10 and Figure 6.11, respectively.

6.2 Framework for Incremental Consistency of UML/SPT Models

An UML/SPT design model of a real-time system is an UML model that captures in addition concurrency and time constraints. As such, it is composed of several UML diagrams annotated with stereotypes to describe concurrency and time constraints. It is not straightforward to provide a single and comprehensive definition for consistency of UML/SPT models. The consistency issue of these models can be summarized as shown in Figure 6.12. We distinguish between intra-diagram and inter-diagram consistencies. Intra-diagram consistency concerns one type of diagrams, also called one view of the system. For such kind of

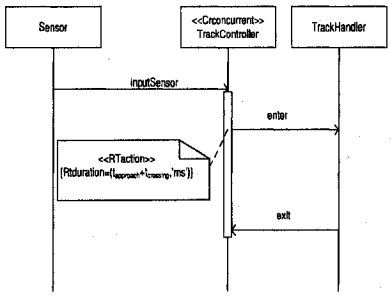

Figure 6.4: Entering Train Scenario

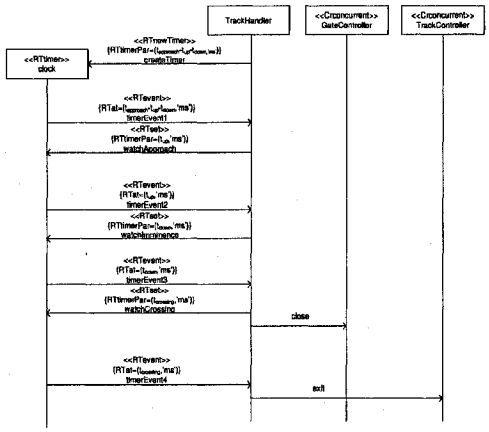

Figure 6.5: TrackHandler Timed Behavior

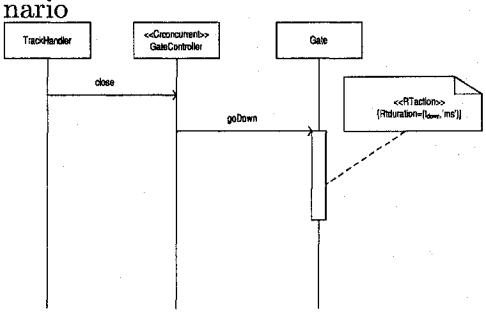

Figure 6.6: Gate Closing Scenario

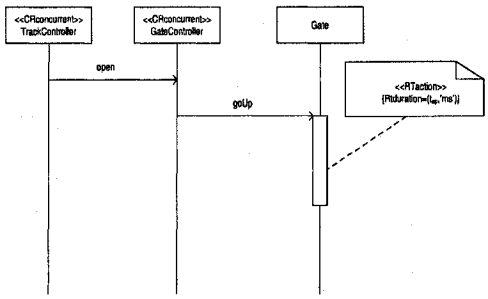

Figure 6.7: Gate Opening Scenario

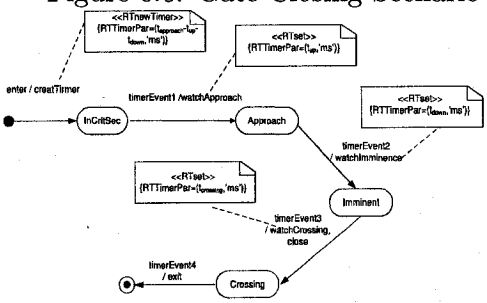

Figure 6.8: TrackHandler State Machine

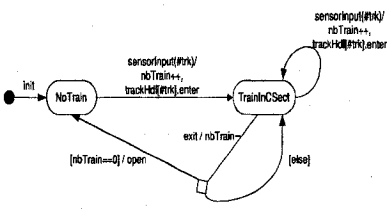

Figure 6.9: TrackController State Machine

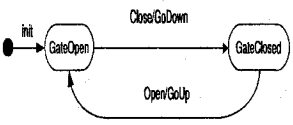

Figure 6.10: GateController State Machine

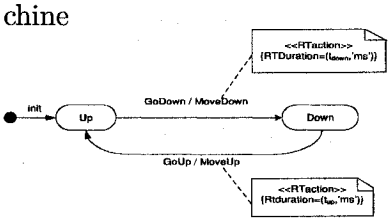

Figure 6.11: Gate State Machine

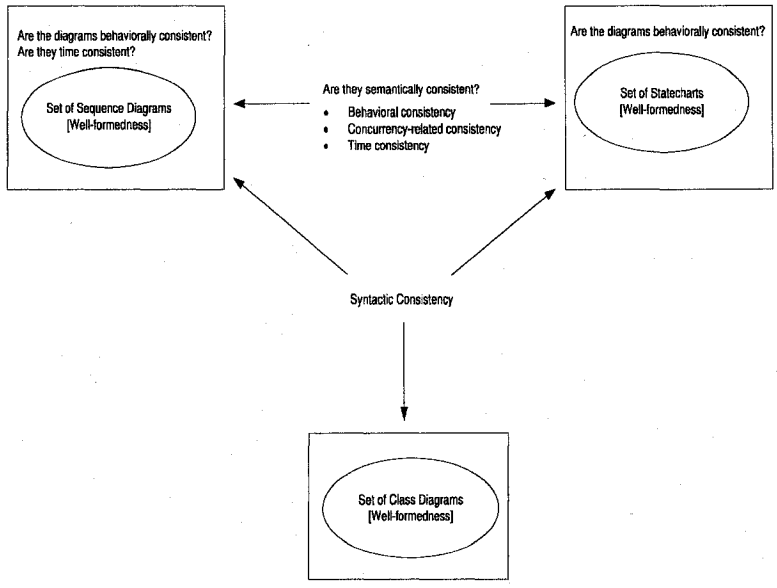

Figure 6.12: Consistency of UML/SPT Models

consistency we are concerned for instance with the well-formedness of the diagrams, which can be checked using UML well-formedness rules expressed in the metamodel using OCL. These rules help in obtaining UML diagrams that are well-formed with respect to the abstract syntax. The semantic consistency of each view is important. This is usually referred to as (semantic) correctness, and is particularly important in the case of behavioral diagrams like sequence diagrams or statecharts. This kind of consistency is well studied by formal verification community that investigated thoroughly the behavioral correctness and the timing correctness of such diagrams. Similarly to intra-diagram consistency, inter-diagram consistency, can be syntactic or semantic. Semantic consistency includes, in addition to the behavioral consistency, concurrency-related consistency and time consistency.

6.2.1 Syntactic Consistency

This consistency includes the well-formedness of each individual diagram in the model, which can be checked using UML built-in well-formedness rules. In addition, syntactic consistency is an inter-diagram static property. The syntactic elements used in the overlapping diagrams should be coherent and compatible. For instance, it can be defined for sequence diagrams and statecharts composing a UML model [67]. Several approaches in the literature [53] address the consistency of a UML model at the syntactical level.

Example

The model described in Section 6.1 is not syntactically consistent. In the sequence diagram depicted in Figure 6.6, `TrackHandler` sends a message `close` to `GateController`. This requires a link between two instances of these classes to enable this communication. There is no association between these classes in the class diagram. Consequently, the class diagram and the sequence diagram in Figure 6.6 are syntactically inconsistent.

6.2.2 Semantic Consistency

The semantic consistency is a dynamic property. We distinguish the behavioral consistency for general-purpose UML models from concurrency-related consistency and time consistency, which are specific to UML/SPT models.

Behavioral consistency

Behavioral consistency is defined for the diagrams used to describe the dynamic behavior of systems. These are mainly the sequence diagrams and statecharts, which capture two different perspectives of the system behavior. Indeed, a sequence diagram describes a partial behavior of the system, which is a particular run/execution of the system. On the other hand, a statechart is a comprehensive description of the behavior of a single object/class. Consequently, a set of sequence diagrams and statecharts model a consistent behavior if the interactions modeled by each sequence diagram can be generated by a particular run of the statecharts associated with the objects involved in the sequence diagram. This can be checked for example by mapping the statecharts and the sequence diagrams to a timed automata formalism [64].

Concurrency-related Consistency

It comes on top of behavioral consistency to capture issues specific to concurrency in UML/SPT models. It is related to the concurrency choices that are expressed in UML/SPT models. Concurrency design choices are important to use efficiently the system resources in order to satisfy time constraints. However, concurrency is likely to lead to issues such

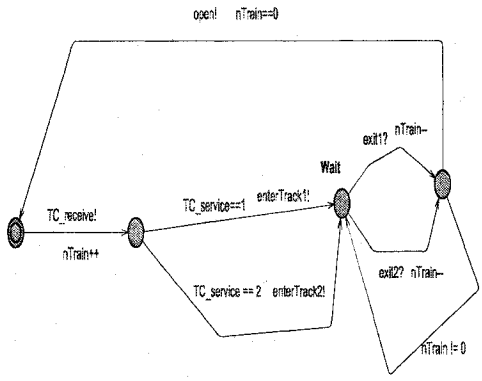

Figure 6.13: Track Controller with Sequential Track Handlers

as deadlock and other race conditions. Concurrency modeling with UML/SPT is done with stereotypes defined in the *RTConcurrencyModeling* package. The semantics of these stereotypes is defined by the concurrency domain model. We have given a formal definition for this domain model using timed automata in Chapter 4 and have shown how this enables the usage of model checking techniques for detecting concurrency related problems in UML/SPT models.

Example Let us consider the model discussed in Section 6.1 with two tracks. Figure 6.13 shows the timed automaton corresponding to the class `TrackController`. This timed automaton is generated using the techniques introduced in Chapter 4. UPPAAL [69] shows that the CTL expression 6.1 is satisfied.

$$\exists\Diamond(\texttt{(TrkHdl1.Crossing or TrkHdl1.Crossing) and gt.Up}) \qquad (6.1)$$

This expression models the possibility that a train is crossing the gate section while the gate is open. This problem is due to a flawed concurrency design choice where `TrackHandler` entities are passive objects. They use the thread of control of their associated concurrent unit, `TrackController`, to proceed and meanwhile the latter is blocked (i.e. in the wait sate). Any *inputsensor* received is then missed and consequently the train can cross the gate after the `TrackController` has send an *open* message to the `GateController`. `TrackHandler` should then be concurrent.

Time Consistency

It comes on top of behavioral and concurrency related consistencies. It is related to time constraints expressed using UML/SPT time stereotypes. We distinguish two kinds of time consistency in UML/SPT models: The logical time consistency of sequence diagrams and the detailed design time consistency. We elaborate on time consistency in Section 6.4.

6.3 Formal Notation and Definitions

We present in this section a formal notation for the UML behavioral diagrams, sequence diagrams and statecharts. Similar ones have been presented in the literature [67] [71]. We use this notation to define formally the behavioral consistency between a set of sequence diagrams and a set of statecharts.

Definition 6.1 *A sequence diagram SeqD is a tuple <O, E, V, Label> where:*

- O *is the set of objects.*
- $E = S \cup R$ *is the set of events.*
- $V \subseteq S \times R$
- $Label : V \rightarrow MNames$ *is a labeling function and MNames is a set of messages names.*

A sequence diagram describes a sequence of message events. Each message m is associated with two causally ordered events, a send event, $send(m) \in S$, and a reception event, $receipt(m) \in R$, respectively. Semantically, a sequence diagram is seen as a partially ordered set of events. In the following $Object : E \rightarrow O$ is a function mapping an event to the object on which it occurs.

Definition 6.2 *The semantics of a sequence diagram SeqD <O, E, V, Label> is defined by the structure* $(E, \preceq)$ *where* $\preceq$ *is defined as follows:*

- $\forall \ (e_i, e_j) \in V \Rightarrow e_i \preceq e_j$
- $\forall e_i, e_j \in E$ *and* $Object(e_i) = Object(e_j)$ *and* $t(e_i) \leq t(e_j) \Rightarrow e_i \preceq e_j$

We define the function $\Pi_o^{SeqD} : E^* \times O \rightarrow E^*$ as the projection of the sequence of events induced by a sequence diagram $SeqD$ on an object $o \in SeqD.O$. This function yields a totally ordered set of events because all the events associated to one object are ordered.
In the following, we consider a formal definition of a simple statechart. This definition omits, for the sake of simplicity, other features of statecharts such as sub-states, pseudo-states, etc.

Definition 6.3 *A statechart SC is a tuple <S, E, A, T> where:*

- *S is the set of states.*
- *E is the set of events.*
- *A is a set of actions.*
- $T : S \times E \times A \rightarrow S$ *a transition relationship.*

The operational semantics of a statechart is defined informally in the UML metamodel [92]. Moreover, there are several proposals in the literature for the formal description of the UML statecharts semantics [99]. In the following, we assume that the predicate $IsARun(sc, se)$ is true if the events sequence se corresponds to a valid transition sequence of the statechart sc.

Using the previous notation, we can define the behavioral consistency between a sequence diagram and a set of statecharts as follows:

Definition 6.4 *A sequence diagram SeqD and set of statecharts* $SC = \{o.sc | o \in SeqD.O\}$ *model a consistent behavior if and only if:*

$$\forall \ o \in SeqD.O,\ IsARun(o.sc, \Pi_o^{SeqD}) = True$$

Consequently, the behavioral consistency between a set of sequence diagrams and a set of statecharts can be defined as follows:

Definition 6.5 *A set of sequence diagrams* $SEQD = \{SeqD_1, SeqD_2, .., SeqD_n\}$ *and a set of statecharts* $SC = \{Sc_1, Sc_2, ..., Sc_m\}$ *define a consistent behavior if and only if each sequence diagram* $SeqD_i \in SEQD$ *and the set of statecharts* $SC' = \{o.sc \in SC | o \in SeqD_i.O\}$ *define a consistent behavior.*

6.4 UML/SPT Time Consistency

In this section, we present an approach for the verification of the time consistency of UML statecharts with respect to a set of sequence diagrams capturing the time constraints. We assume that each sequence diagram models a system end-to-end transaction in response to an external event. The main idea underlying our approach is to use schedulability analysis as a means to check the time consistency. Indeed, a sequence diagram captures an interaction subject to a specific time constraint. As a result, a sequence diagram induces a sequence of state transitions in each statechart. This transition sequence involves a sequence of computations/actions. The statecharts are consistent with a set of sequence diagrams if and only if all the computations executed by the statecharts and induced by the different sequence diagrams are schedulable in the context of a particular deployment environment. This means that in such a deployment environment, in the worst-case scenario, all these computations can be completed within the deadlines resulting from the time constraints.
Our approach is to rely on an appropriate schedulability analysis technique. In order to do so, we generate a UML/SPT-based schedulability analysis model from the statcharts, the sequence diagrams and a deployment model. The latter describes platform-dependent information such as CPU characteristics, shared resources, threads, priorities, etc. This information allows to determine the worst case execution time (WCET) of the different actions in the deployment environment using techniques such as [27]. The generated UML/SPT model is an instance of the schedulability analysis domain model defined in the *SAprofile* package of UML/SPT. This model captures the system external events and the corresponding system responses. These are composed of the actions executed by the different objects and that are allocated to the different available threads. This model can then be supplied for schedulability analysis. We have defined in Chapter 5 an approach for transforming

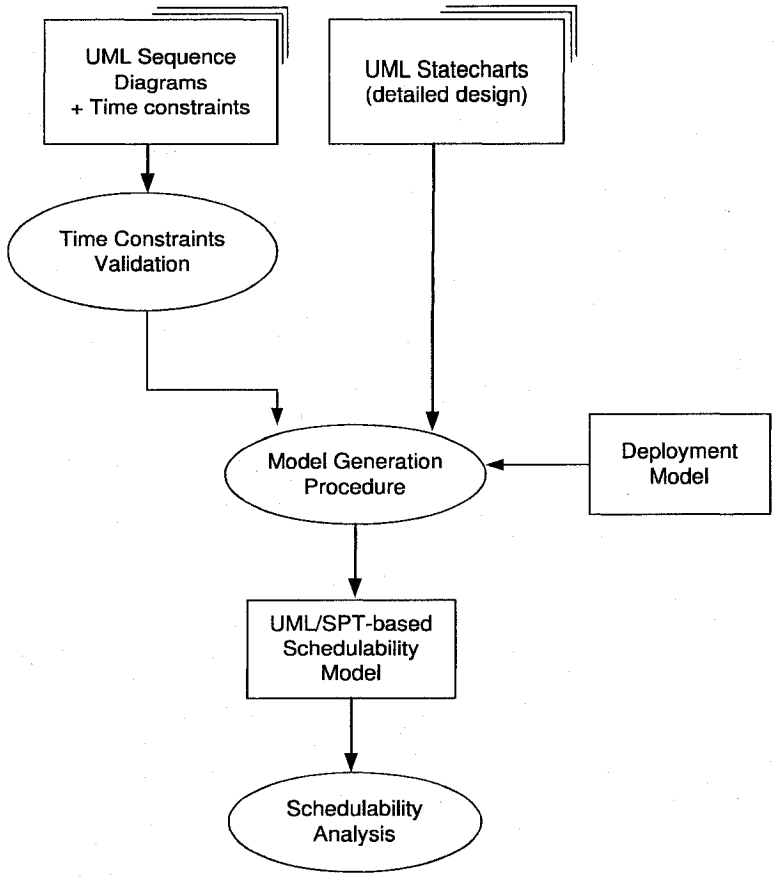

Figure 6.14: UML/SPT Model Time Consistency

UML/SPT models into task models suitable for schedulability analysis. In the following, we elaborate on the main parts of our approach, which is outlined in Figure 6.14

6.4.1 Logical Time Consistency Validation

The time constraints captured using sequence diagrams should be logically consistent. This is necessary otherwise no behavior would satisfy contradictory time constraints and hence no possible implementation. The techniques proposed in [131] can be used for this step. These techniques allow for checking time consistency in MSC specifications. These can be adapted to check the consistency of UML/SPT time constraints modeled with UML sequence diagrams.

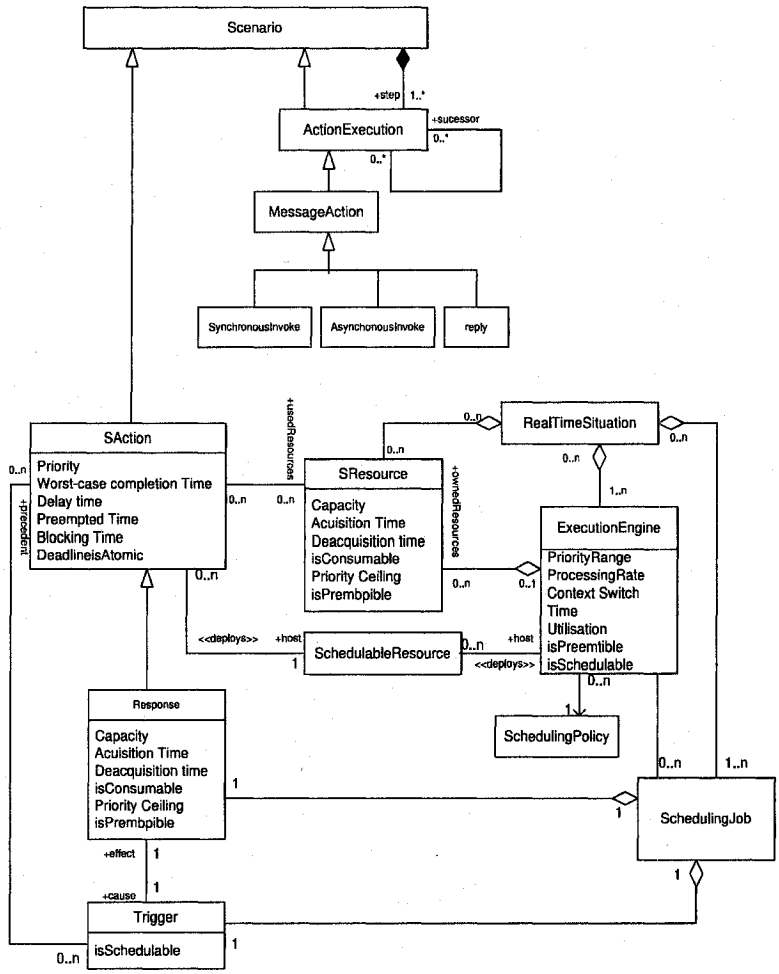

Figure 6.15: Compiled Domain Model supporting Schedulability Analysis from UML/SPT

6.4.2 UML/SPT Model Generation

In this section, we focus on the step of UML/SPT-based schedulability model generation. The generated model is an instance of the domain model illustrated in Figure 6.15. We have compiled this domain model from the dynamic usage model, the concurrency model and the schedulability analysis domain model defined in UML/SPT standard [91].

Our general procedure to generate this model is outlined in Algorithm 4. The objective of Step 1 is to determine the set of computation units executed by the statecharts and triggered by the reception of an event. These computation units are composed of all the actions executed by the statechart in a run-to-completion step. These actions include those executed in entry of a state, the exit of a state and the transition. In order to do so, the set of events in the sequence diagram is partitioned using the projection function $\Pi_{o_i}^{SeqD}$ defined

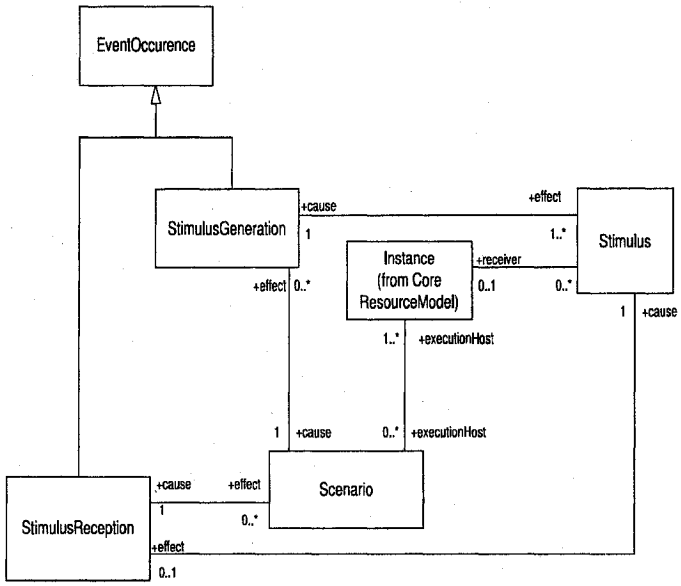

Figure 6.16: Causality Domain Model

earlier. This yields a totally ordered set of events per object, tr_{o_i}. This set is then restricted to the reception events as these are the computations triggering events, $tr^R_{o_i}$. The set of computations units per object, $Action_{o_i}$, is then determined using these reception events by a function $getR2C(Statechart, event)$, which computes for each statechart the different actions executed at the reception of an event. These computation units correspond to the class *SAction* in the domain model shown in Figure 6.15. The corresponding stereotype provided by UML/SPT is ≪SAction≫. In Step 2 and Step 3, the relationship between the determined computation units is established. This relationship is either a sequentiality or a causality relationship. The sequentiality relationship captures the sequence of *ActionExecutions* within a *Scenario/SAction* as shown in the domain model in Figure 6.15. This is determined using the order of the reception events associated to one object $tr^R_{o_i}$. The causality relationship corresponds the causality domain model defined in UML/SPT and shown in Figure 6.16. This is determined using the order relation between the send event and reception event. We assume that the predicate $gen(a, e)$ is true when the execution of the action a generates the event e and the predicate $trigger(e, a)$ is true when the event e triggers the execution of the action a. The final step, Step 4, in this procedure integrates the deployment information in the generated model. This information is provided by a deployment model and includes for example the worst case execution time, the priority of

each action, the deployment of the actions on the available threads.

Algorithm 4 UML/SPT-based Schedulability Model Generation

Input:
  let *SeqD* <O, E, V,Label> be a sequence diagram
  let $SC = \{o_i.sc | \forall o_i \in O\}$ be a set of associated statecharts
Step 1: Actions determination
for all $o_i \in O$ **do**
  Step 1.1: Event partition
    let $tr_{o_i} \leftarrow \Pi_{o_i}^{SeqD} = \{e_{o_{i1}}, e_{o_{i2}}, .., e_{o_{im}}\}$
  Step 1.2: Event restriction to receptions
    let $tr_{o_i}^R \leftarrow tr_{o_i} \cap R = \{e_{o_{i1}}^r, e_{o_{i2}}^r, .., e_{o_{ik}}^r\}$
  Step 1.3: Run to completion steps
    let $Action_{o_i} \leftarrow \cup_{j \leq k}\{getR2C(o_i.sc, e_{o_{ij}}^r)\}$
end for
  let $Actions = \cup_{o_i \in O} Action_{o_i}$
Step 2: Sequentiality relation
$\xi = \{(a_j, a_k) | a_j, a_k \in Actions \wedge \exists o_i \in O \wedge \exists e_{o_{ij}}^r, e_{o_{ik}}^r \in tr_{o_i}^R \wedge e_{o_{ij}}^r \preceq e_{o_{ik}}^r\}$
Step 3: Causality relation
$\zeta = \{(a_i, a_j) | a_i, a_j \in Actions \wedge \exists e, e' \in E \wedge (e, e') \in V \wedge gen(a_i, e) \wedge trigger(e', a_j)\}$
Step 4: Deployment information integration
for all $a_i \in Actions$ **do**
    let $(a_i.wcet, a_i.priority, a_i.thread, ...) \leftarrow deploys(a_i)$
end for

6.4.3 Schedulability Analysis Phase

In Chapter 5, we have defined a metamodel based transformation. This transformation allows to derive a task model expected by the schedulability analysis technique defined in [112] from a UML/SPT model. For this step of our approach, we use this model transformation to enable the schedulability analysis. The analysis allows for computing the worst case response time for each action in each end-to-end system transaction. The design model is schedulable if all the response times satisfy the deadlines. In such a case the statecharts are consistent with the time constraints expressed in the sequence diagrams assuming the deployment environment provided by the deployment model.

6.4.4 Application to Railroad Crossing Model

As an example to illustrate the application of Algorithm 4, we consider three important scenarios, which are the arrival of a train to a critical section (sequence diagram $SeqD1$), a train reaching a point where the gate has to be closed ($SeqD2$) and a train exiting the section ($SeqD3$). Figure 6.17, Figure 6.18 and Figure 6.19 show, respectively, the process of determining the actions executed by the different statecharts and induced by the sequence diagrams. Figure 6.20 shows the results of the causality and sequentiality relationships and the obtained end-to-end transactions in the system. The obtained UML/SPT model after integration of deployment information is shown in Figure 6.21. This model is then supplied for schedulability analysis [112], after its transformation into a suitable task model using our approach defined in Chapter 5, in order to check the time consistency of the design model.

6.5 Related Work

Consistency is an important issue in the context of UML modeling. This led to an extensive research work [6], [53]. A classification of the consistency issues in terms of horizontal/intra-model and vertical/inter-model in UML modeling has been pointed out in [31] and in [53]. The closest work to ours is probably [67], which distinguishes syntactic and semantic consistency and singles out temporal consistency as a particular case of semantic consistency. The focus in this work was put on the time constraints although the modeling language considered, UML-RT, does not have any provision for expressing time constraints. In [72], an approach was presented to check the consistency of real-time system specifications using sequence diagrams. This approach is based on a linear programming algorithm to check the consistency of time constraints in a sequence diagram and a composition of sequence diagrams. Time consistency in MSC based on a formal semantics for MSC has been investigated in [131].

The transformation of UML artifacts used to model dynamic behavior into timed automata for purposes of verification and consistency checking has been the focus of several research works including [28], [64]. Firley et al. consider in [28] an approach to transform sequence

Figure 6.17: Actions Induced by SeqD1

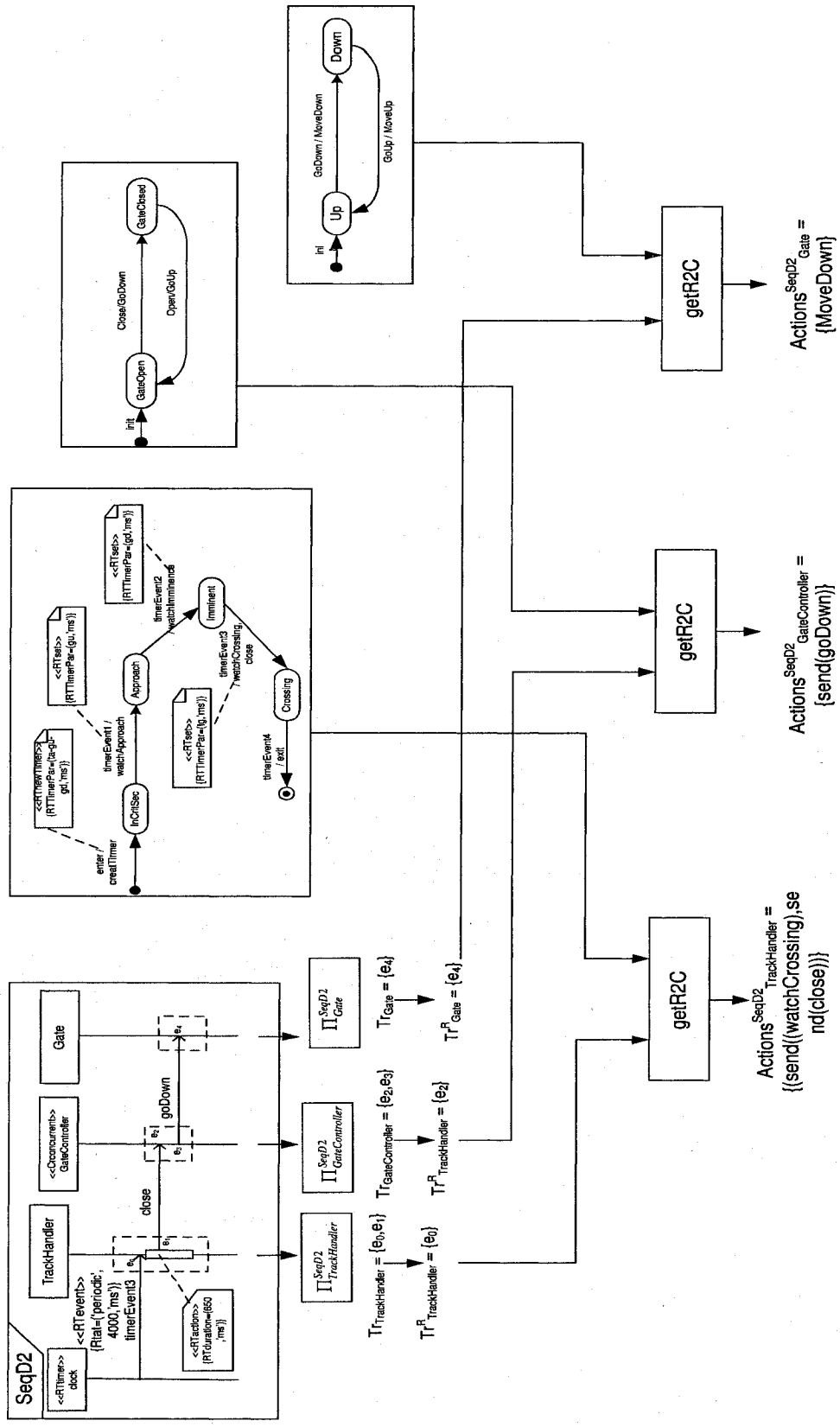

Figure 6.18: Actions Induced by SeqD2

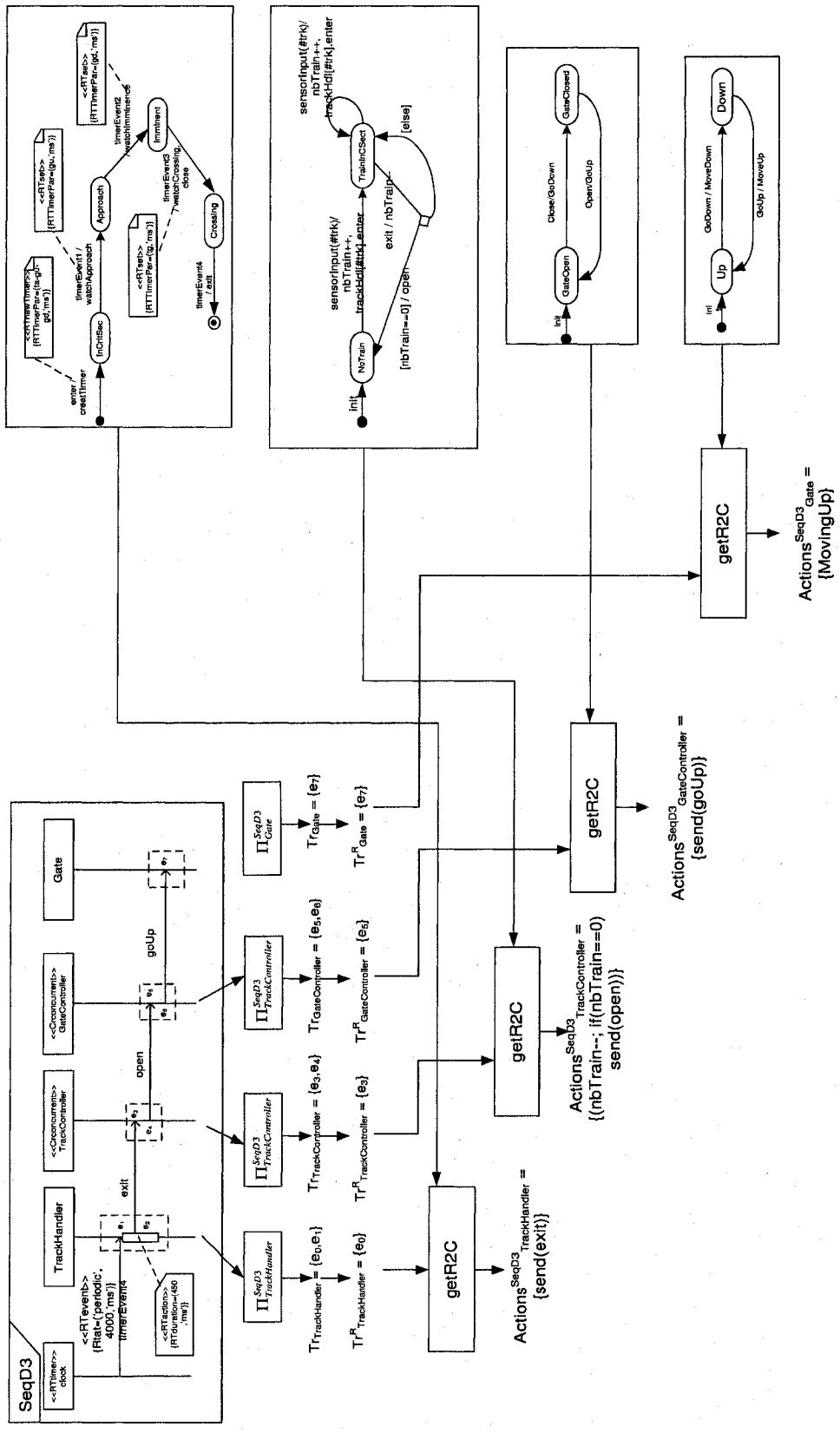

Figure 6.19: Actions Induced by SeqD3

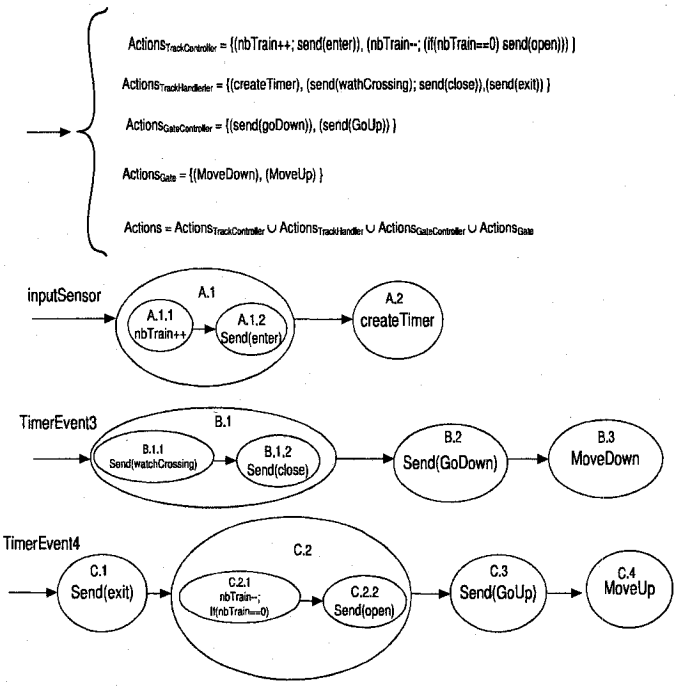

Figure 6.20: End-to-End Transactions Induced by the Sequence Diagrams

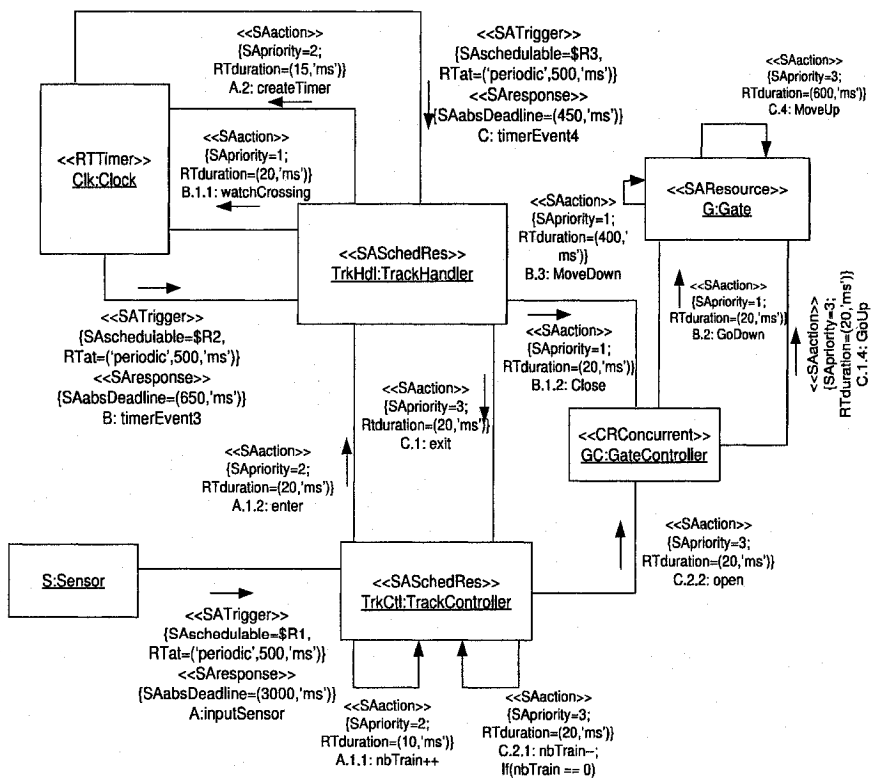

Figure 6.21: Generated UML/SPT-based Schedulability Model

diagrams with time constraints to observer timed automata. Knapp et al. address in [64] the issue of consistency between the main UML artifacts used to model the real-time system dynamic behavior: timed state machines and sequence diagrams with time constraints. The former express the detailed design of the system and the latter specify the main scenarios. This work proposed a technique for the verification of the consistency between the two views based on UPPAAL timed automata. The timed state machines are compiled into timed automata and the sequence diagrams annotated with time constraints are transformed into observer timed automata. The latter transformation is a slight extension to the technique proposed in [28]. The model checker UPPAAL is then used to verify the timed automata with respect to the observer timed automata. This technique is embodied in a prototype tool called HUGO/RT.

6.6 Conclusion

UML Model consistency is a challenging issue. It becomes worse when aspects such as concurrency and time constraints are taken into account. We presented in this chapter a framework for an incremental definition of the consistency in UML/SPT models. Within this framework, we address respectively the syntactic and semantic consistency, which includes in addition to behavioral consistency, the concurrency-related consistency and time consistency as these are important features of UML/SPT models. Considering the time consistency of UML/SPT models, we focused on the consistency of a set of statecharts with respect to time constraints modeled using sequence diagrams. Our approach to address this issue is to use schedulability analysis techniques. We showed how to generate UML/SPT model supporting such schedulability analysis techniques from statecharts and sequence diagrams. This model can then be further transformed to appropriate task model using techniques such as those we presented in Chapter 5.

The approach based on schedulability analysis for checking time consistency between statecharts and sequence diagrams provides however a limited feedback to the designer. There are other important questions that need to be addressed in future work. Indeed, when the analysis shows that a design model is not time consistent, what can be done to fix

the inconsistency? Is it possible to provide more fine-grained feedback in pointing out the origin of the inconsistency? What changes can be operated on the design model and/or the deployment environment that might fix the problem? These remain open questions in this thesis.

Chapter 7

Conclusion and Future Work

Real-time software systems are deployed nowadays in many applications, including home appliances, automotive, avionics, military applications, to mention just a few examples. These systems should be logically correct and should satisfy a set of time constraints. In addition, they are reactive and concurrent in order to handle the different concurrent events that come from the environment in which these systems are deployed. Such characteristics make the design of real-time software systems complex and challenging.

Modeling is an important engineering activity, which relies on using models to raise the abstraction level. This widens the engineers visibility and increases their control over the complexity of the systems they are building/managing. The model-driven approach is therefore adequate to address the complexity issue of real-time software systems. It is very advantageous to use models with rigorous semantics. This enables the verification and the validation of designs and potentially the automatic synthesis of implementations. Taking advantage of the models in the design of systems in general and real-time software systems in particular requires two components of paramount importance: a *modeling language* and *a model-based development methdology/process.*

The Unified Modeling Language is the *de facto* standard modeling language for software-intensive systems. UML is successful because it is a visual language and therefore it is intuitive; it is a multi-view modeling language addressing different aspects of a system which is very importance to deal with systems complexity; and finally, it is adaptable. Indeed, UML is customizable to different domains through its extensibility mechanisms and profiles. The

success of UML led to a surge of the interest in using UML by system engineering community, system-on-chip community, and the real-time software systems community. UML is adapted in these domains through specific and standard profiles such as the UML profile for system engineering (SysML) [95] and the UML profiles for systems-on-chip [96]. The most important UML profile for embedded and real-time systems is the OMG standard called UML profile for schedulability, performance and time [91]. This profile is in the process of a major revamp to lead to MARTE [97].

7.1 Contributions

In this thesis we have focused on the following issues:

- Limitations in UML/SPT expressiveness with respect to some real-time requirements modeling needs.
- Lack of formal semantics which limits the possibility of automatic manipulation of UML/SPT models. The domain models of this profile (i.e. its metamodel) have their semantics defined in English.
- UML/SPT is designed to support schedulability analysis of real-time software systems. There is however a semantic gap between UML/SPT models and the task models expected/used by well-established real-time analysis techniques. It is then important to close this semantic gap in order to enable the application of these techniques for the validation of the schedulability property of a UML/SPT model.
- While the fact that UML provides a multi-view approach, through a multitude of diagrams, is advantageous to cope with software systems complexity, this, on the other hand, may lead to inconsistencies. Moreover, using UML/SPT to model real-time systems, new aspects are taken into account, namely concurrency, time constraints, and schedulability. These aspects may contribute to worsen the consistency issue.

Considering the complexity of a UML/SPT model, which is composed of several UML diagrams and which captures in addition aspects such as concurrency and time constraints

using appropriate stereotypes defined in the profile, it is difficult to provide one straightforward definition of the consistency of a UML/SPT model. We adopt an incremental approach to the verification of the consistency of UML/SPT models. We distinguish, respectively the syntactic and semantic levels. With regard to semantic consistency, we further decompose it into behavioral, concurrency-related, and time consistency.

Our main contributions in this thesis can be summarized as follows:

- A survey of the different UML profiles used to model real-times systems proposed in the academia/industry and an assessment of their capabilities and limitations with respect to a variety of criteria such as formal foundation, supported analysis and tool support.

- A definition of an extension of UML/SPT using the UML extensibility mechanisms and the profile definition methodology. This extension enables to capture in a UML/SPT model multicast communications, which are required to model the behavior of protocols used in distributed real-time systems, such as RMTP2.

- A formalization of the semantics of UML/SPT concurrency domain model using the formalism of timed automata. This formal semantics enables using model checking techniques in order to verify the concurrency-related consistency of a UML/SPT model.

- A definition of an MDA-compliant approach to bridge the semantic gap between task models used by schedulability analysis techniques and UML/SPT models.

- A prototype implementation of this approach using both a high level model transformation language (ATL) and a low-level implementation using XML technologies (XML schema and XSLT).

- A definition of an incremental framework for the verification of the consistency of UML/SPT models. Within this framework we address the semantic consistency of UML/SPT models. Concurrency-related consistency is checked using our timed automata semantics for the concurrency domain model. The verification of the time

consistency is enabled through our schedulability analysis approach of UML/SPT models.

7.2 Future Work

Several issues are left open in this thesis. They can be summarized as follows:

- The metamodeling approach used for the definition of UML profiles in general and which we used for the extension of UML/SPT presents an interesting issue. Indeed, the concepts required for a certain domain could be modeled in a variety of manners, which leads to different metamodels. For instance, two different metamodels have been proposed in [20] and [107] for the reliability prediction domain and used to extend UML/SPT metamodel. Therefore, it is necessary to assess the consistency between the different profiles and extensions.

- The feedback given by the schedulability analysis tool is limited to determining if an end-to-end transaction is schedulable or not. While this is useful, a more fine-tuned diagnostic, such as, which object-to-thread allocation is the cause of a missed deadline would be more informative. In addition, reflecting the results of an analysis into the original UML/SPT design model is limited in UML/SPT to some variables in the UML/SPT stereotypes. This is however a UML/SPT intrinsic limitation [10].

- Our implementation of the model transformation takes as input the information modeled using UML/SPT stereotypes. These stereotypes are not models on their own but are closely linked to the UML design model. We assume that the information needed by the transformation is extracted from the UML design model. Very few commercial tools, however, implement the UML/SPT stereotypes, but we expect this to change in the near future with the new UML profile, MARTE [97].

- Our approach on using schedulability analysis to verify the time consistency of a set of statecharts with respect to a set of sequence diagrams with time constraints provides a limited feedback to the designer. There are other important questions that need to be addressed such as when the analysis shows that a design model is not time

consistent, what can be done to fix the inconsistency? Is it possible to provide more fine-grained feedback in pointing out the origin of the inconsistency? What changes can be operated on the design model and/or the deployment environment that might fix the problem?

- Finally, modeling, metamodeling, UML and its profiles are definitely the key concepts of an effective MDD/MDA-based approach to the development and analysis of software systems including real-time software systems. In this context, the different profiles standardized by the OMG, including, the UML profile for system engineering (SysML), the UML profiles for systems-on-chip and the UML profile for real-time (UML/SPT or MARTE), and the upcoming ones, form a UML-based modeling ecosystem of domain specific languages. It is interesting to investigate a coherent framework using all these different languages for a particular system requiring their modeling capabilities.

Bibliography

[1] Nawal Addouche, Christian Antoine, and Jacky Montmain. UML Models for Dependability Analysis of Real-time Systems. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, pages 5209–5214, The Hague, Netherlands, 10-13 October 2004. IEEE.

[2] R. Alur and D. Dill. Automata for Modelling Real-time Systems. In *Proceedings of ICALP'90*, volume 443 of *LNCS*. Springer, 1990.

[3] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Sciences*, 126(2):183–235, 1994.

[4] Ludivic Apvrille, Jean-Pierre Courtiat, Christophe Lohr, and Pierre de Saqui-Sannes. TURTLE: A Real-Time UML Profile Supported by Formal Validation Toolkit. *IEEE Trans. Software Eng.*, 30(7):473–487, 2004.

[5] Ludivic Apvrille, Pierre de Saqui-Sannes, and Ferhat Khendek. TURTLE-P: a UML Profile for the Formal Validation of Critical and Distributed Applications. *Software and Systems Modeling (SOSYM)*, 5(4):449–466, 2006.

[6] Egidio Astesiano and Gianna Reggio. An Attempt at Analysing the Consistency Problems in the UML from a Classical Algebraic Viewpoint. In Martin Wirsing, Dirk Pattinson, and Rolf Hennicker, editors, *Recent Trends in Algebraic Development Techniques, 16th International Workshop, WADT 2002*, volume 2755 of *LNCS*, pages 56–81. Springer, 2003.

[7] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. Software Eng.*, 30(5):295–310, 2004.

[8] Sven Weber Bas Graaf and Arie van Deursen. Model-driven Migration of Supervisory Machine Control Architectures. *Available online 28 June 2007 in the Journal of Systems and Software*, 2007.

[9] Gabor Batori and Domonkos Asztalos. Using TTCN-3 for Testing Platform Independent Models. In Ferhat Khendek and Rachida Dssouli, editors, *Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference (TestCom 2005)*, volume 3502 of *LNCS*, pages 289–303. Springer, 2005.

[10] Simona Bernardi and Dorina C. Petriu. Comparing two UML Profiles for Non-functional Requirement Annotations: the SPT and QoS Profiles. In *SVERTS*, Lisbone, Portugal, October 2004.

[11] Jean Bézivin, Slimane Hammoudi, Denivaldo Lopes, and Frédéric Jouault. Applying MDA Approach for Web Service Platform. In *Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference, (EDOC'04)*, pages 58–70. IEEE Computer Society, 2004.

[12] Jean Bézivin and Frédéric Jouault. Using ATL for Checking Models. *Electr. Notes Theor. Comput. Sci.*, 152:69–81, 2006.

[13] Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the Large and Modeling in the Small. In Uwe Aßmann, Mehmet Aksit, and Arend Rensink, editors, *Model Driven Architecture, European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004*, volume 3599 of *LNCS*, pages 33–46. Springer, 2005.

[14] Grady Booch. *Object-Oriented Analysis and Design with Applications.* Benjamin–Cummings, Redwood City, Calif., 2nd edition, 1994.

[15] Marius Bozga, Susanne Graf, and Laurent Mounier. IF-2.0: A Validation Environment for Component-Based Real-Time Systems. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, (CAV'2002)*, volume 2404 of *LNCS*, pages 343–348. Springer, 2002.

[16] Lionel C. Briand, Yvan Labiche, and Y. Miao. Towards the Reverse Engineering of UML Sequence Diagrams. In Arie van Deursen, Eleni Stroulia, and Margaret-Anne D. Storey, editors, *10th Working Conference on Reverse Engineering (WCRE 2003)*, pages 57–66, Victoria, Canada, 13-16 November 2003. IEEE Computer Society.

[17] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework : a Developer's Guide.* The eclipse series. Addison-Wesley, 2004.

[18] Eric Cariou, Raphaël Marvie, Lionel Seinturier, and Laurence Duchien. OCL for the Specification of Model Transformation Contracts. In *In the Workshop OCL and Model Driven Engineering of the Seventh International Conference on UML*, pages 69–83, October 2004.

[19] Franck Chauvel and Jean-Marc Jézéquel. Code Generation from UML Models with Semantic Variation Points. In Lionel C. Briand and Clay Williams, editors, *8th International Conference Model Driven Engineering Languages and Systems (MoDELS 2005)*, volume 3713 of *LNCS*, pages 54–68, 2005.

[20] Vittorio Cortellessa and Antonio Pompei. Towards a UML Profile for QoS: a Contribution in the Reliability Domain. In *Proceedings of the 4th international workshop on Software and performance (WOSP'04)*, pages 197–206, New York, NY, USA, 2004. ACM Press.

[21] Andrea D'Ambrogio. A Model Transformation Framework for the Automated Building of Performance Models from UML Models. In *Proceedings of the Fifth International Workshop on Software and Performance (WOSP'05)*, pages 75–86. ACM, 2005.

[22] Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML. In *Formal Methods for Components and Objects, First International Symposium, FMCO 2002*, pages 71–98, 2002.

[23] Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. A Discrete-time UML Semantics for Concurrency and Communication in Safety-critical Applications. *Sci. Comput. Program.*, 55(1-3):81–115, 2005.

[24] Miguel A. de Miguel. General Framework for the Description of QoS in UML. In IEEE Computer Society, editor, *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, pages 61–70, Hakodate, Hokkaido, Japan, May 2003.

[25] Bruce Powel Douglass. *Real Time UML: Advances in the UML for Real-Time Systems.* Addison-Wesley Professional, 2004.

[26] Eclipse. Eclipse Official Web Site. http://www.eclipse.org/.

[27] E. Erpenbach. *Compilation, Worst-Case Execution Times and Schedulability Analysis of Statecharts Models.* Phd thesis, Department of Mathematics and Computer Science of the University of Paderborn, April 2000.

[28] Thomas Firley, Michaela Huhn, Karsten Diethers, Thomas Gehrke, and Ursula Goltz. Timed Sequence Diagrams and Tool-Based Analysis - A Case Study. In Robert B. France and Bernhard Rumpe, editors, *UML'99: The Unified Modeling Language - Beyond the Standard*, volume 1723 of *LNCS*, pages 645–660. Springer, 1999.

[29] Stephan Flake and Wolfgang Mueller. A UML Profile for Real-Time Constraints with the OCL. In Jean-Marc Jézéquel, Heinrich Hussmann, and Stephen Cook, editors, *UML 2002 - The Unified Modeling Language. Model Engineering, Languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings*, volume 2460 of *LNCS*, pages 179–195. Springer, 2002.

[30] Robert B. France, Sudipto Ghosh, Trung T. Dinh-Trong, and Arnor Solberg. Model-Driven Development Using UML 2.0: Promises and Pitfalls. *IEEE Computer*, 39(2):59–66, 2006.

[31] L. Groenewegen G. Engels, J. M. Küster and R. Heckel. A Methodology for Specifying and Analyzing Consistency of Object-oriented Behavioral Models. In Volker Gruhn, editor, *Proceedings of the 8th European Software Engineering Conference (ESEC)*, page 186195. ACM Press, 2001.

[32] Vahid Garousi, Lionel C. Briand, and Yvan Labiche. Control Flow Analysis of UML 2.0 Sequence Diagrams. In *Model Driven Architecture - Foundations and Applications, First European Conference, ECMDA-FA 2005*, volume 3748 of *LNCS*, pages 160–174, Nuremberg, Germany, November 7-10 2005. Springer.

[33] S. Gérard and F. Terrier. UML for Real-Time: Which Native Concepts to Use? In Luciano Lavagno, Grant Martin, and Bran Selic, editors, *UML for Real Design of Embedded Real-time Systems*, pages 17–51. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[34] Sébastien Gérard and Ileana Ober. Parallelism/Concurrency Specification within UML. In *Workshop on Concurrency Issues in UML*, 2001. White paper.

[35] Abdelouahed Gherbi and Ferhat Khendek. On the Design and Schedulability Analysis of Distributed Object-Oriented Real-time Systems. In *Proceedings of the WiP session of 17th Euromicro Conference on Real-time Systems, ECRTS 2005, Also available as IRISA Internal Publication number 1726*, Palma de Mallorca, Balearic Islands, Spain, 2005.

[36] Abdelouahed Gherbi and Ferhat Khendek. Distributed Real-Time Behavioral Requirements Modeling Using Extended UML/SPT. In Reinhard Gotzhein and Rick Reed, editors, *System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006*, volume 4320 of *LNCS*, pages 34–48. Springer, 2006.

[37] Abdelouahed Gherbi and Ferhat Khendek. From UML/SPT Models to Schedulability Analysis: a Metamodel-Based Transformation. In *9th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2006), 24-26 April 2006, Gyeongju, Korea*, pages 343–350. IEEE Computer Society, 2006.

[38] Abdelouahed Gherbi and Ferhat Khendek. UML Profiles for Real-Time Systems and their Applications. *Journal of Object Technology*, 5(4):149–169, May-June 2006.

[39] Abdelouahed Gherbi and Ferhat Khendek. Consistency of UML/SPT Models. In E. Najm E. Gaudin and R. Reed, editors, *13th System Design Language Forum, SDL Forum 2007*, volume 4745 of *LNCS*, page 203224. Springer, 2007.

[40] Abdelouahed Gherbi and Ferhat Khendek. From UML/SPT Models to Schedulability Analysis: Approach and a Prototype Implementation using ATL and XML. *Revision Submitted to Automated Software Engineering Journal*, 2007.

[41] Abdelouahed Gherbi and Ferhat Khendek. Timed-Automata Semantics and Analysis of UML/SPT Models with Concurrency . In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 412–419. IEEE Computer Society, 2007.

[42] Hassan Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley Professional, 2000.

[43] Bas Graaf and Arie van Deursen. Model-Driven Consistency Checking of Behavioural Specifications. In *Fourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'07)*, pages 115–126. IEEE Computer Society, 2007.

[44] Susanne Graf, Ileana Ober, and Iulian Ober. Timed Annotations with UML. In *SVERTS*, San Francisco, USA, October 2003.

[45] ATLAS group. *KM3: Kernel MetaMetaModel Manual - version 0.3* - , 2005.

[46] Gordon P. Gu and Dorina C. Petriu. From UML to LQN by XML Algebra-based Model Transformations. In *WOSP '05: Proceedings of the 5th International Workshop on Software and Performance*, pages 99–110. ACM Press, 2005.

[47] Zonghua Gu and Zhimin He. Real-Time Scheduling Techniques for Implementation Synthesis from Component-Based Software Models. In George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, Clemens A. Szyperski, and Kurt C. Wallnau, editors, *Component-Based Software Engineering, 8th International Symposium, CBSE 2005 Proceedings*, volume 3489 of *LNCS*, pages 235–250, St. Louis, MO, USA, 2005. Springer.

[48] David Harel. On Visual Formalisms. *Communication of the ACM*, 31(5):514–530, May 1988.

[49] Øystein Haugen. Comparing UML 2.0 Interactions and MSC-2000. In *System Analysis and Modeling, 4th International SDL and MSCWorkshop, (SAM' 2004)*, volume 3319 of *LNCS*, pages 65–79, Ottawa, Canada, June 2005. Springer.

[50] Constance L. Heitmeyer and Nancy A. Lynch. The Generalized Railroad Crossing: A Case Study in Formal Verification of Real-Time Systems. In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS '94)*, pages 120–131, San Juan, Puerto Rico, 1994. IEEE Computer Society.

[51] L. Hélouët. Distributed System Requirements Modeling with Message Sequence Charts. *International Journal of Information and Software Technology*, 45:701–714, 2003.

[52] Jon Holt. *UML (Unified Modelling Language) for Systems Engineering.* Institution of Engineering and Technology, 2004.

[53] Zbigniew Huzar, Ludwik Kuzniarz, Gianna Reggio, and Jean-Louis Sourrouille. Consistency Problems in UML-Based Software Development. In *UML Satellite Activities*, volume 3297 of *LNCS*, pages 1–12. Springer, 2004.

[54] INCOSE. http://www.incose.org.

[55] The Open SystemC Initiative. SystemC. http://www.systemc.org.

[56] ITU-T. SDL Combined with UML. 2000. ITU-T recommandation Z.109.

[57] IUT-T. Message Sequene Charts (MSC-2000). *ITU-T Recommendation Z.120*, November 1999.

[58] Ivar Jacobson. *Object-Oriented Software Engineering: a Use Case-driven Approach.* Addison–Wesley, Wokingham, England, 1995.

[59] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. ATL: a QVT-like Transformation Language. In *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, (OOPSLA 2006)*, pages 719–720. ACM, 2006.

[60] Frédéric Jouault and Jean Bézivin. KM3: A DSL for Metamodel Specification. In Roberto Gorrieri and Heike Wehrheim, editors, *Formal Methods for Open Object-Based Distributed Systems, 8th IFIP WG 6.1 International Conference, FMOODS 2006*, volume 4037 of *LNCS*, pages 171–185. Springer, 2006.

[61] Frédéric Jouault and Ivan Kurtev. On the Architectural Alignment of ATL and QVT. In Hisham Haddad, editor, *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC'06)*, pages 1188–1195. ACM, 2006.

[62] Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In Jean-Michel Bruel, editor, *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005*, volume 3844 of *LNCS*, pages 128–138. Springer, 2006.

[63] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise.* Object Technology. Addison-Wesley, 2003.

[64] Alexander Knapp, Stephan Merz, and Christopher Rauh. Model Checking - Timed UML State Machines and Collaborations. In Werner Damm and Ernst-Rüdiger Olderog, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium (FTRTFT 2002)*, volume 2469 of *LNCS*, pages 395–416. Springer, 2002.

[65] Jana Koehler, Rainer Hauser, Shane Sendall, and Michael Wahler. Declarative Techniques for Model-Driven Business Process Integration. *IBM Systems Journal*, 44(1), 2005.

[66] Ingolf Krüger, Wolfgang Prenninger, and Robert Sandner. Broadcast MSCs. *Formal Aspects of Computing*, 16(3):194–209, 2004.

[67] Jochen Malte Küster and Joachim Stroop. Consistent Design of Embedded Real-Time Systems with UML-RT. In *4th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001), 2-4 May 2001, Magdeburg, Germany*, pages 31–40. IEEE Computer Society, 2001.

[68] Sha L., Abdelzaher T. F., Årzén K. E., Cervin A., Baker T. P., Burns A., Buttazzo G.C., Caccamo M., Lehoczky J. P., , and Mok A. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Syst.*, 28(2-3):101–155, 2004.

[69] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[70] Luciano Lavagno, Grant Martin, and Bran Selic, editors. *UML for Real: Design of Embedded Real-time Systems.* Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[71] Xuandong Li and Johan Lilius. Timing Analysis of UML Sequence Diagrams. In Robert B. France and Bernhard Rumpe, editors, *UML'99: The Unified Modeling Language - Beyond the Standard, Second International Conference*, volume 1723 of *LNCS*, pages 661–674. Springer, 1999.

[72] Xuandong Li and Johan Lilius. Checking Compositions of UML Sequence Diagrams for Timing Inconsistency. In *7th Asia-Pacific Software Engineering Conference (APSEC 2000), 5-8 December 2000, Singapore*, pages 154–161. IEEE Computer Society, 2000.

[73] Peltier M, J Bzivin, and G Guillaume. MTRANS: A General Framework based on XSLT for Model Transformations. In *Proceedings of the Workshop on Transformations in UML (WTUML'01)*, Genova, Italy, 2001.

[74] Gabor Madl, Sherif Abdelwahed, and Douglas C. Schmidt. Verifying Distributed Real-time Properties of Embedded Systems via Graph Transformations and Model Checking. *Real-Time Systems*, 33(1-3):77–100, 2006.

[75] G. Martin and W. Muller, editors. *UML for SOC Design.* Springer, 2005.

[76] P. Marwedel. *Embedded System Design.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[77] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5):14–18, 2003.

[78] Miller and J. Mukerji. MDA Guide Version 1.0.1. *OMG Document: omg/2003-06-01*, June 2003.

[79] T. Montgomery, B. Whetten, M. Basavaiah, S. Paul, N. Rastogi, J. Conlan, , and T. Yeh. The RMTP2 Protocol IETF Draft. IETF (Internet Engineering Task Force), April 1998.

[80] Birger Møller-Pedersen. SDL Combined with UML. *Telektronikk*, 4:36–53, 2000.

[81] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, Y. Vanderperren, and Ku Leuven. UML for ESL Design: Basic Principles, Tools, and Applications. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 73–80, New York, NY, USA, 2006. ACM Press.

[82] Iulian Ober, Susanne Graf, and Ileana Ober. Validation of UML Models via a Mapping to Communicating Extended Timed Automata. In Susanne Graf and Laurent Mounier, editors, *Model Checking Software, 11th International SPIN Workshop*, volume 2989 of *LNCS*, pages 127–145. Springer, 2004.

[83] Iulian Ober and Ileana Stan. On the Concurrent Object Model of UML. In Patrick Amestoy, Philippe Berger, Michel J. Daydé, Iain S. Duff, Valérie Frayssé, Luc Giraud, and Daniel Ruiz, editors, *Euro-Par '99 Parallel Processing, 5th International Euro-Par Conference*, volume 1685 of *LNCS*, pages 1377–1384. Springer, 1999.

[84] OMG. Unified Modeling Language (UML) 1.3 Specification. *version 1.3 formal/01-09-67*, March 2000.

[85] OMG. Unified Modeling Language (UML) 1.4 Specification. *version 1.4 formal/01-09-67*, September 2001.

[86] OMG. MOF 2.0 Query / Views / Transformations RFP. *OMG Document: ad/2002-04-10*, April 2002.

[87] OMG. UML Profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms. *OMG Adopted Specification, ptc/2004-06-01*, June 2004.

[88] OMG. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. *Final Adopted Specification ptc/05-11-01*, November 2005.

[89] OMG. MOF 2.0/XMI Mapping Specification, v2.1. *formal/05-09-01*, September 2005.

[90] OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE). *Request For Proposals OMG Document: realtime/05-02-06*, Fubruary 2005.

[91] OMG. UML Profile for Schedulability, Performance, and Time Specification. *OMG Adopted Specification Version 1.1, formal/05-01-02*, January 2005.

[92] OMG. Unified Modeling Language: Superstructure. *version 2.0 formal/05-07-04*, August 2005.

[93] OMG. Meta Object Facility (MOF) Core Specification. *OMG Available Specification Version 2.0 formal/06-01-01*, January 2006.

[94] OMG. Object Constraint Language. *OMG Available Specification Version 2.0 formal/06-05-01*, May 2006.

[95] OMG. OMG Systems Modeling Language (OMG SysML) Specification. *Final Adopted Specification ptc/06-05-04*, May 2006.

[96] OMG. UML Profile for System on a Chip (SoC). *Available Specification Version 1.0 formal/06-06-01*, June 2006.

[97] OMG. A UML Profile for MARTE, Beta 1. *Adopted Specification OMG Document: ptc/07-08-04*, Auguste 2007.

[98] Guadalupe Ortiz and Juan Hernández. Service-Oriented Model-Driven Development: Filling the Extra-Functional Property Gap. In Asit Dan and Winfried Lamersdorf, editors, *4th International Conference on Service-Oriented Computing - ICSOC 2006*, volume 4294 of *LNCS*, pages 171–185. Springer, 2006.

[99] Ivan Paltor and Johan Lilius. Formalising UML State Machines for Model Checking. In Robert B. France and Bernhard Rumpe, editors, *UML'99: The Unified Modeling Language - Beyond the Standard, Second International Conference*, volume 1723 of *LNCS*, pages 430–445. Springer, 1999.

[100] Sanjoy Paul, Krishan K. Sabnani, John C.-H. Lin, and Supratik Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal On Selected Areas In Communications*, 15(3):407–421, April 1997.

[101] Polly M. S. Poon, Tharam S. Dillon, and Elizabeth Chang. XML as a Basis for Interoperability in Real Time Distributed Systems. In *2nd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (WSTFEUS 2004)*, Vienna, Austria, 2004. IEEE Computer Society.

[102] Ivan Porres. A Toolkit for Model Manipulation. *Software and System Modeling*, 2(4):262–277, 2003.

[103] Apache XML Project. Xalan Java Version 2.7.0. http://xml.apache.org/xalan-j/.

[104] Apache XML Project. Xerces2 Java Parser. http://xerces.apache.org/xerces2-j/.

[105] Eclipse Modeling Project. Eclipse Modeling Project Web site. http://www.eclipse.org/modeling.

[106] M2M Project. Eclipse M2M Project Web site. http://www.eclipse.org/m2m/.

[107] Genana Nunes Rodrigues, David S. Rosenblum, and Sebastian Uchitel. Reliability Prediction in Model-Driven Development. In *Proc. ACM/IEEE 8th Int'l Conf. on*

Model Driven Engineering Languages and Systems, volume 3713 of *LNCS*, pages 339–354, 2005.

[108] RTL. Software and Tools for Communicating Systems. http://www.laas.fr/RT-LOTOS/.

[109] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice–Hall, Englewood Cliffs, NJ, 1991.

[110] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition).* Pearson Higher Education, 2004.

[111] M. Saksena, P. Karvelas, and Y.Wang. Automated Synthesis of Multi-Tasking Implementations from Real-Time Object-Oriented Models. In *3rd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)*, pages 360–367, Newport Beach, CA, USA, March 2000. IEEE Computer Society.

[112] Manas Saksena and Panagiota Kervelas. Designing for Schedulability: Integrating Schedulability Analysis with Object-Oriendted Design. In *The 12th Euromicro Conference on Real-Time Systems*, June 2000.

[113] Manas Saksena and Bran Selic. Real-Time Software Design- Sate of the Art and Future Challenges. *IEEE Canadian Review*, pages 5–8, Summer 1999.

[114] B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modeling.* John Wiley and Sons, 1994.

[115] B. Selic and L. Motus. Using Models in Real-Time Software Design. *IEEE Control Systems Magazine*, 23(3):31–42.

[116] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. March 1998. Whitepaper Available from www.objectime.com.

[117] Bran Selic. On the Semantic Foundations of Standard UML 2.0. In *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the*

Design of Computer, Communication and Software Systems, SFM-RT 2004, volume 3185 of *LNCS*, pages 181–199. Springer, 2004.

[118] Bran Selic. A Systematic Approach to Domain-Specefic Language Design Using UML. In *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2007)*, pages 2–9. IEEE Computer Society, 2007.

[119] James Skene and Wolfgang Emmerich. A Model-Driven Approach to Non-Functional Analysis of Software Architectures. In *18th IEEE International Conference on Automated Software Engineering (ASE 2003), 6-10 October 2003, Montreal, Canada*, pages 236–239. IEEE Computer Society, 2003.

[120] OMEGA ST. Project. http://www-omega.imag.fr/.

[121] John A. Stankovic. Misconceptions About Real-Time Computing. *IEEE Computer*, 21(10):10–19, 1988.

[122] Telelogic. Telelogic Products. http://www.telelogic.com/products/.

[123] TTool. A Toolkit for Editing and Validating TURTLE Diagrams. http://www.eurecom.fr/ apvrille/TURTLE/index.html.

[124] Tom Verdickt, Bart Dhoedt, Frank Gielen, and Piet Demeester. Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models. *IEEE Trans. Software Eng.*, 31(8):695–711, 2005.

[125] W3C. XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999. http://www.w3.org/TR/xslt, 1999.

[126] W3C. Validator for XML Schema REC (20010502) version. http://www.w3.org/2001/03/webdata/xsv, 2005.

[127] Yun Wang and Manas Saksena. Scheduling Fixed-Priority Tasks with Preemption Threshold. In *6th International Workshop on Real-Time Computing and Applications Symposium (RTCSA '99), 13-16 December 1999, Hong Kong, China.* IEEE Computer Society.

[128] Gwang Sik Yoon and Yong Rae Kwon. Extending MSC for Reactive Systems. In *IEEE CS International Symposium on Human-Centric Computing Languages and Environments (HCC'2001)*. IEEE Computer Society, 2001.

[129] Justyna Zander, Zhen Ru Dai, Ina Schieferdecker, and George Din. From U2TP Models to Executable Tests with TTCN-3 - An Approach to Model Driven Testing. In Ferhat Khendek and Rachida Dssouli, editors, *Testing of Communicating Systems, 17th IFIP TC6/WG 6.1 International Conference(TestCom 2005)*, volume 3502 of *LNCS*, pages 289–303. Springer, 2005.

[130] Tong Zheng and Ferhat Khendek. An Extension for MSC-2000 and Its Application. In Edel Sherratt, editor, *Telecommunications and beyond: The Broader Applicability of SDL and MSC, Third International Workshop, (SAM 2002)*, volume 2599 of *LNCS*, pages 221–232. Springer, 2003.

[131] Tong Zheng and Ferhat Khendek. Time Consistency of MSC-2000 Specifications. *Computer Networks*, 42(3):303–322, 2003.