ENSMA : Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
LIAS : Laboratoire d'Informatique et d'Automatique pour les Systèmes

THESE

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AÉROTECHNIQUE

(Diplôme National — Arrêté du 7 août 2006)

Ecole Doctorale : Science et Ingénierie pour l'Information, Mathématiques
Secteur de Recherche : INFORMATIQUE ET APPLICATIONS

Présentée par :

Yassine OUHAMMOU

****

Model-based Framework for Using Advanced Scheduling Theory in Real-Time Systems Design

-----------------------------------

Cadre fondé sur les modèles pour une utilisation avancée de la théorie de l'ordonnancement dans la conception des systèmes temps réel

****

Directeurs de Thèse : **Emmanuel GROLLEAU** et **Pascal RICHARD**
Co-encadrant : **Michaël RICHARD**

Soutenue le 12 Décembre 2013
devant la Commission d'Examen

JURY

Rapporteurs :	**Marco DI NATALE**	Associate Professor, Scuola Superiore Sant'Anna, Pise
	Frank SINGHOFF	Professeur, Université de Bretagne Occidentale, Brest
	Yves SOREL	Directeur de Recherche, INRIA, Rocquencourt
Examinateurs :	**Emmanuel GROLLEAU**	Professeur, ENSMA, Futuroscope
	Jérôme HUGUES	Maître de Conférences, ISAE, Toulouse
	Pascal RICHARD	Professeur, Université de Poitiers, Poitiers

ENSMA : **E**cole **N**ationale **S**upérieure de **M**écanique et d'**A**érotechnique
LIAS : **L**aboratoire d'**I**nformatique d'**A**utomatique pour les **S**ystèmes

THESE

pour l'obtention du Grade de

**DOCTEUR DE L'ECOLE NATIONALE SUPERIEURE
DE MECANIQUE ET D'AEROTECHNIQUE**
(Diplôme National – Arrêté du 7 août 2006)

Ecole Doctorale : Sciences et Ingénierie pour l'Information, Mathématiques
Secteur de Recherche : Informatique et Applications

Présentée par :

Yassine OUHAMMOU

****

Model-based Framework for Using Advanced Scheduling Theory in Real-Time Systems Design

---

Cadre fondé sur les modèles pour une utilisation avancée de la théorie de l'ordonnancement dans la conception des systèmes temps réel

****

Directeurs de Thèse : Emmanuel GROLLEAU et Pascal RICHARD
Co-encadrant : Michaël RICHARD

****

Soutenue le 12 Décembre 2013
devant la Commission d'Examen

****

JURY

Marco DI NATALE	Associate Professor, Scuola Superiore Sant'Anna, Pise	Rapporteur
Frank SINGHOFF	Professeur, Université de Bretagne Occidentale, Brest	Rapporteur
Yves SOREL	Directeur de Recherche, INRIA, Rocquencourt	Rapporteur
Emmanuel GROLLEAU	Professeur, ENSMA, Futuroscope	Examinateur
Jérôme HUGUES	Maître de Conférences, ISAE, Toulouse	Examinateur
Pascal RICHARD	Professeur, Université de Poitiers, Poitiers	Examinateur
Yamine AIT-AMEUR	Professeur, ENSEEIHT, Toulouse	Invité

Remerciements

J'aimerais tout d'abord exprimer mes vifs remerciements à mon directeur de thèse Monsieur Emmanuel GROLLEAU, actuellement directeur du LIAS, pour m'avoir guidé pendant ces trois années de thèse. Je lui suis également reconnaissant pour l'encadrement qu'il m'a fait bénéficier. Sa disponibilité, sa gentillesse et ses qualités pédagogiques et scientifiques ont été d'une aide inestimable.

Je tiens également, à remercier mes co-directeurs de thèse Messieurs Michaël RICHARD et Pascal RICHARD pour leur encadrement, leurs discussions intéressantes et leurs encouragements qui m'ont été déterminants pour ma formation de chercheur sur les plans scientifique et technique.

Je tiens à remercier Monsieur Marco DI NATALE, Monsieur Frank SINGHOFF, et Monsieur Yves SOREL d'avoir accepté d'être les rapporteurs de ce travail et à qui je souhaite ma profonde reconnaissance. Un merci également aux Messieurs Jérôme HUGUES et Pascal RICHARD pour l'honneur qu'ils m'ont fait d'être examinateurs de cette thèse et de l'importance qu'ils ont accordée à mon travail.

J'aimerais adresser un remerciement particulier à Monsieur Yamine AIT-AMEUR, ex-directeur de LIAS, de m'avoir accueilli au sein du laboratoire et de m'avoir convaincu de faire une thèse autour des systèmes temps réel. Je le remercie également d'avoir accepté de participer à ce jury autant qu'invité.

Un merci particulier à Ladjel, Mickael, Stéphane, Laurent, Allel, Henri, Fred et Zoé pour leur présence et leur soutien cordial. Je remercie tout le personnel de l'ENSMA, je pense particulièrement à Frédéric CARREAU et Claudine RAULT pour leurs aides techniques et administratives.

Mes vifs remerciements vont également à tous les amis, Youness et sa famille, Chedlia, Idir, Ahmed, Ilyès, Selma, Amira, Loé, Nabil, Georges, Christian, Valéry, Linda, François, Ahcene, Kaoutar, Thomas, Moustapha pour leurs encouragements et leur bonne humeur.

Mes dernières (mais pas les moins méritantes) pensées vont vers ma famille, plus particulièrement à mes parents et mes frères qui m'ont toujours soutenu. Leur présence et leurs encouragements sont pour moi les piliers fondateurs de ce que je suis et de ce que je fais.

Maman,Papa, je vous dédie cette thèse.

Contents

Introduction

I. Motivation

Real-time computing is spreading more and more in our daily lives through avionic systems, automotive systems, multimedia, telecommunications, robot controllers, etc. A real-time system operates in a dynamic environment and must constantly be adapted to changes in that environment. The correctness of critical real-time systems is determined partly by the functional results of computations, and partly by the time at which results are produced.

The model driven development of real-time systems can take place over several years, and designers have to cope with new hardware/software/standards during the design process. Moreover, software development costs are sharply impacted by wrong design choices made in the early stages of development, in particular during the design phase, but often detected after the implementation. Timing vulnerabilities may result in catastrophic failures, hence they are among vulnerabilities which must be detected at the design phase of the life-cycle development, to ensure the feasibility of the system under development over a reasonable time-to-market. This kind of prediction is based on the schedulability analysis.

The schedulability analysis is one of the main analyses required to ensure the timing correctness of a real-time system. The real-time scheduling theory has been devoted to propose different models providing several levels of expressiveness, and different analytical methods with different levels of accuracy.
Recently, different approaches taking benefits from the model driven engineering ease the use of real-time scheduling theory during the design phase. Real-time designers can operate during the analysis phase, since different analysis tools based on different input analytical models facilitate the utilization of the scheduling theory.

Due to the criticality of real-time systems, the choice of the appropriate validation tests and/or dimensioning techniques has a relevant impact on the development life-cycle. The utilization of the real-time scheduling theory in practical cases could be profitable. Unfortunately, it is not sufficiently applied and research results have been exploited in industry only to a modest extent to date. The steep learning curve behind many of the current analysis methods has been one of the major impediments to their adoption. The utilization of current solutions is driven by the real-time designer's experience. However, experts in both design and analysis of real-time systems are uncommon. Determining what type of an analysis technique or a model to use for a given system-design situation is difficult. Yet, collecting the relevant information to define the analysis context may be laborious.

By taking the criticality of hard real-time systems into consideration, the lack of scheduling analysis background can lead to an imprecise analysis result due to a wrong choice of analysis contexts or analysis tests. Indeed, the conception process which starts from the modeling to analysis phase provided by current solutions may lead to a potential gap (measured as pessimism and over-dimensioning) due to the estrangement between the abstract model and the practical application.

II. Thesis objectives and Solution overview

The thesis contributions are driven by two essential purposes.

The first objective is to help designers to cope with the analysis difficulty, then to orient them in order (1) to choose the most appropriate analysis tests and (2) to ease the design modification due to refinement or dimensioning actions. Therefore, designers will be guided to build scheduling-aware models.

The second objective is to enhance the applicability of the real-time scheduling theory. Nowadays, only few works are adopted and give a good satisfaction due to their impact on the development life-cycle of real-time systems. Therefore, it is needful to provide an easy way for transferring the research studies from academia to industrial practice.

To achieve the above objectives, we propose a flexible way to unify modeling and analysis efforts for supporting schedulability analysis. We use the model driven engineering to ensure a good utilization of existing solutions and to complete their provided results by exploiting different research studies. Consequently, we propose a framework called MoSaRT (Modeling-oriented Scheduling analysis of Real-Time systems). It is an intermediate framework between real-time design languages and schedulability analysis tools.

The MoSaRT framework offers dual capabilities in order to have a chain of transformation and verification starting from design languages down to analysis tools.

First capability is the MoSaRT design language that is a specific modeling language dedicated to designers. It supports designers during the design phase to dimension their models, and to analyze their applications with several third-party tools, without requiring a deep understanding of the schedulability analysis techniques.

Second capability is the MoSaRT analysis repository which is dedicated to be fed by real-time analysts in order to play an advising role and to help designers. The goal of the analysis repository is to identify which analysis tests are applicable to the design model.

III. Thesis structure

The remainder of the manuscript is composed as follows.

The research foundations part contains three chapters. It starts by Chapter 2, which introduces several generalities related to the real-time systems and highlights the general structure of the systems on which we are interested in this thesis. Chapter 2 also gives an overview about the software design area and its importance throughout the development process.

Chapter 3 presents briefly different aspects related to the real-time scheduling theory and discusses the common timing properties and constraints determining the behavior of real-time systems. An overview of different kinds of analyses studying the feasibility and the schedulability of real-time systems is also presented.

Chapter 4 presents some capabilities related to the use of model driven engineering and their impact on the design of real-time systems. It discusses and presents a comparison of some standard modeling languages and analysis tools.

The contribution part starts by Chapter 5, which highlights the work positioning by presenting different problem statements and illustrating them via some motivating examples. Chapter 5 describes the objectives we aim to achieve in this thesis, and gives a brief idea about our proposals.

Chapter 6 is devoted to present the MoSaRT design language. It introduces our contribution referring to the research foundations and describe both functional and operational elements, and their semantics. Rules enabling a good usage of the MoSaRT design language are presented.

Chapter 7 highlights different scheduling characteristics helping designers and easing the use of real-time scheduling theory. It presents an approach based on the analysis repository (i) to improve the way designers check their system designs, and (ii) to increase the applicability of real-time scheduling theory.

Chapter 8 introduces the objectives of the MoSaRT framework, describes the structure of the MoSaRT framework and its various capabilities. The framework is illustrated through several case studies.

Chapter 9 summaries the thesis and discusses obtained results and perspectives.

Part I

Research Foundations

CHAPTER

2

Real-Time Systems: Generalities and Definitions

Contents

Abstract

This chapter is devoted to discuss several generalities related to the real-time systems. Section I presents briefly the definition of real-time systems and their classifications. The general structure of the systems on which we are interested in this thesis has been highlighted in Section II. Finally, Section III is devoted to give an overview about the software design area and its importance throughout the development process.

I. Real-time systems

The main intent of this section is to describe the domain of interest for the current work.

I.1. Definition of real-time systems

I.1.a. Control/command systems

Control/command systems are usually dedicated to handle the execution of a process of the physical world. The command synthesis serves to react and to adjust the system regarding to given requests. Since commands are produced through actuators, requests are generated after measures done on one or several sensors.

I.1.b. Embedded systems

An embedded system is a set of cooperating hardware and software elements dedicated to accomplish a specific mission. The architecture of an embedded system is often composed of different subsystems that are distributed on several equipments. They communicate with their environments by taking decisions, performing calculations and reacting accordingly. Yet, embedded systems are subjected to different environmental and resource constraints like power consumption, size, cost, resources and memories, etc. One of the key points of embedded systems is that the energy is also embedded (batteries, fuel, ambient energy production, etc.).

I.1.c. Real-time systems

A real-time system is any information processing system that must interact with a correct behavior to input events within specified timing bounds [BW09]. In other words, a real-time system has to satisfy both the following requirements types [Sta88]:

- Logical correctness: the result produced and computed by the system has to be correct.
- Timing correctness: the correct behavior of the system is also defined by the timing of the output. A result, that is functionally correct, but not temporally correct (e.g. not respecting the deadline), is considered as a wrong behavior. Moreover, a delay is considered as an error that can lead to severe consequences. Note that this is different from requiring the interaction to be as fast as possible. The timing correctness is usually modeled as deadlines which have to be met.

When embedded control/command systems are subjected to timing constraints, they are considered as real-time systems. Indeed, embedded control/command systems are often real-time systems because the embedded computing resources have to fit the computational requirements of the control. For instance, an over powerful central processing unit is requiring a lot of energy (e.g. battery), while for an adapted central processing unit, the duration of the execution of the treatments cannot be neglected.

I.2. Classification of real-time systems

The classification of real-time systems may be based on different criteria. In the following, we classify the real-time systems according to their criticality in two categories:

- The soft real-time systems are systems which tolerate timing failures, possibly because those failures do not lead to a catastrophe, like telecommunication and multimedia systems [AMK98].
- The hard real-time systems have to meet the required deadlines. Otherwise, something unacceptable can occur. According to the definition provided by Buttazzo [But11], a hard real-time system has to be capable of executing tasks - if missing their deadlines can cause catastrophic consequences - on the environment under control. When the real-time system is used to supervise a critical environment, it is called a safety-critical system (e.g. automotive [AUT], avionics [DH92], supervision of nuclear power plants [NSST98], etc.).

Real-time systems are implemented to be used in different contexts: they can be used as calculating and processing systems, or as system controllers. Most soft real-time systems are used as calculating-oriented systems like video games. Most hard real-time systems are used as control-oriented systems like the anti-lock braking system given as an example below.
In this work we are interested in hard real-time systems which are not necessarily control-oriented and/or safety-critical.

Example: Anti-lock Braking System

When the car driver uses the brake, the anti-lock braking system (ABS) as a controller has to use the environment information (like the wheels speed and the brake pedal) and control the brake operation at the right instant (fractions of a second). Both the functional result (brake activation) and the timing behavior (time at which the functional result is produced) are important in ensuring the car and passengers safety. Then, we can say that the ABS is an embedded, hard real-time, control-oriented and safety-critical system.

Synchronous and asynchronous approaches

Note that this approaches classification contains several notions (like RTOS, preemption and semaphores) which will be presented later.
The synchronous approach has been developed to integrate the determinism into concurrent programing [BB91, Hal92]. The underlying implementation of synchronous approaches is either event based or time-unit based. In event-based implementations, an event is triggering a function, and its execution duration is supposed to be shorter than the inter-arrival delay between successive events. This hypothesis supposes that the computational resources are over powerful. Therefore, in embedded systems, this can be achieved in a time-unit-based implementation: the system considers, at each time unit, the events that have arrived since the previous time unit. It executes the functions related to these events. The synchronous hypothesis is then that the execution requirements to handle all events do not take more than the time unit. Yet, delaying the response to an event by a time unit is not harming the behavior of the system. In both cases, the system can be non-preemptive. Then, there is no need for all

the functionalities (such as semaphores, mailboxes, etc.) of a real-time operating system (RTOS). The advantage of the synchronous approach is its easy implementation and its design verification. However, the synchronous hypothesis is not fitting for several real-time applications.

The asynchronous approach takes the time of the system execution into account. Moreover, tasks may be preempted (interrupted) due to the arrival of events. The system behavior (e.g. event interceptions, task preemptions, execution priorities, etc.) is managed by a RTOS. Events can be produced by the environment via sensors. In this case we talk about event driven programming (also called event triggered) [Kop91]. Nonetheless, a real-time system can be also conceived to be dependent of the internal clock instead of being dependent of the controlled environment. This kind of systems is called the time driven programming [Kop91]. In the current work, we are focusing on the asynchronous approach.

II. General Structure of Real-Time Systems

While most embedded real-time systems are executed on electronic circuits, implementing several protocols and executing various algorithms, a generic architecture for the notion of a real-time system is illustrated in Figure 2.1. This architecture represents a computer-based system interacting with its environment. The control system acts on the process through actuators and gets information about the environment using sensors.

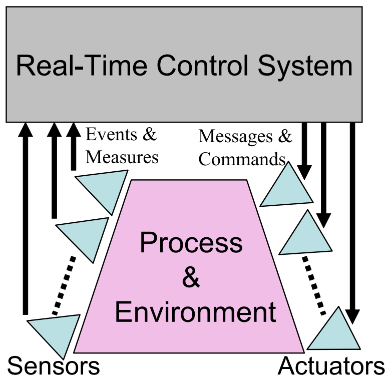


Figure 2.1: A generic architecture of a real-time system

A real-time system consists of a software structure executing on a hardware platform.

II.1. Hardware Structure

The hardware structure has an important impact on the real-time system's execution and also on the system's analyzability. A hardware platform is a composition of a set of elements which can be processors, memories, networks, input/output cards, storage supports, etc. (See Figure 2.2).

The interaction between different hardware elements leads to various types of architectures. We can classify these types in three categories:

- Uniprocessor architecture: the hardware structure is composed of a unique processor. This processing unit is a common resource for software elements composing the software application. A processor or a central processing unit (CPU) is an electronic circuit characterized by several properties like the internal-clock frequency, the energy consumption, etc.
- Multiprocessor or multicore architecture: in this case, the hardware structure contains several processors (or cores) sharing memories. So, by multiprocessor architecture we mean the MIMD (Multiple Instructions on Multiple Data) architecture category [Fly72]. The MIMD architectures can have a shared memory or distributed memories:
  - A shared memory multiprocessor platform enables access from any processor of the system to any location within a common memory, through the interconnection network or via a hierarchical way involving cache memories.
  - In case of a distributed memory multiprocessor platform, each processor accesses its own memory and communicates with the memory of other processors via an interconnection network.
- Distributed architecture: the hardware structure is composed of a set of nodes communicating via networks. Each node may be represented as a uniprocessor or a multiprocessor architecture (see Figure 2.2). Furthermore, the networks are exploited by remote software elements to exchange messages. In real-time systems domain, networks can be classified according to the size (e.g. local area network, wide area network, etc.), the physical support and topology [EP93] (e.g. optic fiber, wireless, bandwidth, etc.) and the access protocol (like Controller Area Network [DBBL07], Avionics Full DupleX Switched Ethernet [Sch07], Asynchronous Transfer Mode [Gor95], etc.).

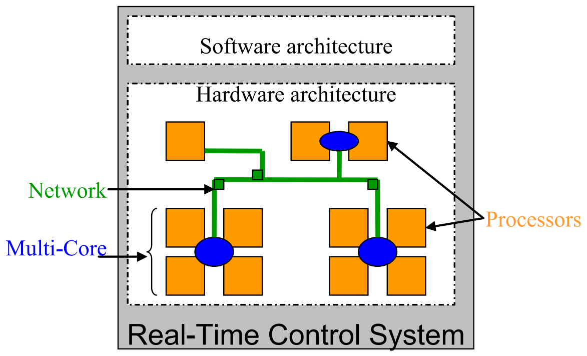


Figure 2.2: A hardware structure of a real-time control system

II.2. Software Structure

The software structure of a real-time system consists of two main parts: the real-time operating system and the application program (see Figure 2.3).
The real-time operating system is the bridge that links the application program to the hardware structure. In the literature, the definition of the real-time operating system is often related to "kernel" and "executive" terms. Hence:

- Kernel: it manages the hardware resources access, and provides a scheduler (in case of an on-line scheduling) or a dispatcher (in case of an off-line scheduling)[1] deciding which part of program has to be executed on which processor.

- Executive: it provides a set of drivers modules and libraries facilitating the files management, the communication management, etc. The real-time executive includes the real-time kernel.

- Real-time operating system (RTOS): it provides a set of means ensuring the system maintenance and the multi-tasking through several services. The RTOS plays the role of a middle-ware between end-users or developers and the real-time executive.

Some RTOS and executives possess their proprietary API (Application Programing Interface), while other RTOS are conforming to standards like Osek/VDX [G+], POSIX [Wal95] defining specifications which have to be respected by RTOS implementations.

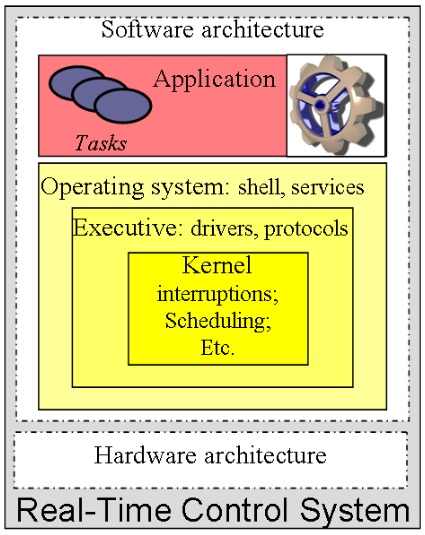


Figure 2.3: A software structure of a real-time control system

The application program corresponds to the software that executes different functionalities enabling the system to control its environment. The application is often structured in several tasks, where every task ensures a sequence of operations in order to perform treatments, to respond to events, to send messages, etc. Some programming languages handle tasks directly by encapsulating the operating systems layer (like Ada language [BW01]). However, other languages (like C or C++) require the utilization of the real-time operating system for tasks management. The notion of "task" represents the basic entity considered as the stepping stone of the development of real-time systems using RTOS. The real-time operating systems provide various services enabling different kinds of interactions between tasks like task communications, mutual exclusions, etc.

[1]On-line and off-line scheduling notions will be discussed later

II.3. Towards a real-time system design

For obtaining a real-time application structured in different tasks, the development of real-time systems has to be elaborated through a set of services ensuring the respect of system's specifications, such as, the timing constraints (e.g. the execution time of a task has to not miss the deadline). However, any incoherent development may cause a non-compliance and may lead to a severe violation of the timing constraints. Whereof, a special attention is dedicated to the design phase which plays a key role in the real-time system's development flow.

III. Real-Time Systems Design

To understand the importance of the design during the life-cycle of a real time system, it is preferable to start by highlighting the whole development process. Then, we emphasize the specificities of the design phase (which is a step of the development process) of real-time systems.

III.1. Real-Time Systems Development

Embedded real-time systems have been increasingly used in different domains. For instance, nowadays, a car is made of an assembly of many systems ensuring the various functionalities of the vehicle. Hundreds of functions are realized or controlled by softwares (e.g. Anti-lock Braking System, Global Positioning System, Airbag, etc.).
The development of real-time systems is difficult, because they are part of a physical environment and they must meet timing requirements. Different characteristics should be considered through the development of a real-time system. Some of these characteristics are [TKK+98]:

- Anticipation of the problems and the risks.
- Easing the cooperation of multiple actors.
- Shortening the development time.
- Controlling the complexity.

To understand the needs in terms of system engineering, we choose the automotive sector to illustrate the growing utilization of real-time embedded systems.

Illustration: real-time systems development in the automotive domain

In the automotive domain, developing real-time embedded systems engages two stakeholder categories: carmakers (or manufacturers) and their tier suppliers. The objective of the carmaker is to market vehicles which satisfy the needs and the customers expectations. Yet, a carmaker has to respect the manufacturing standards and norms. Then, they aim to enrich vehicle by a set of real-time systems in order to fulfill the needs of the customers. Since manufacturers have to ensure the sustainability and the assembly of embedded systems which require a very high-knowledge, their suppliers have the expertise enabling to concretize the expected systems (see Figure 2.4).

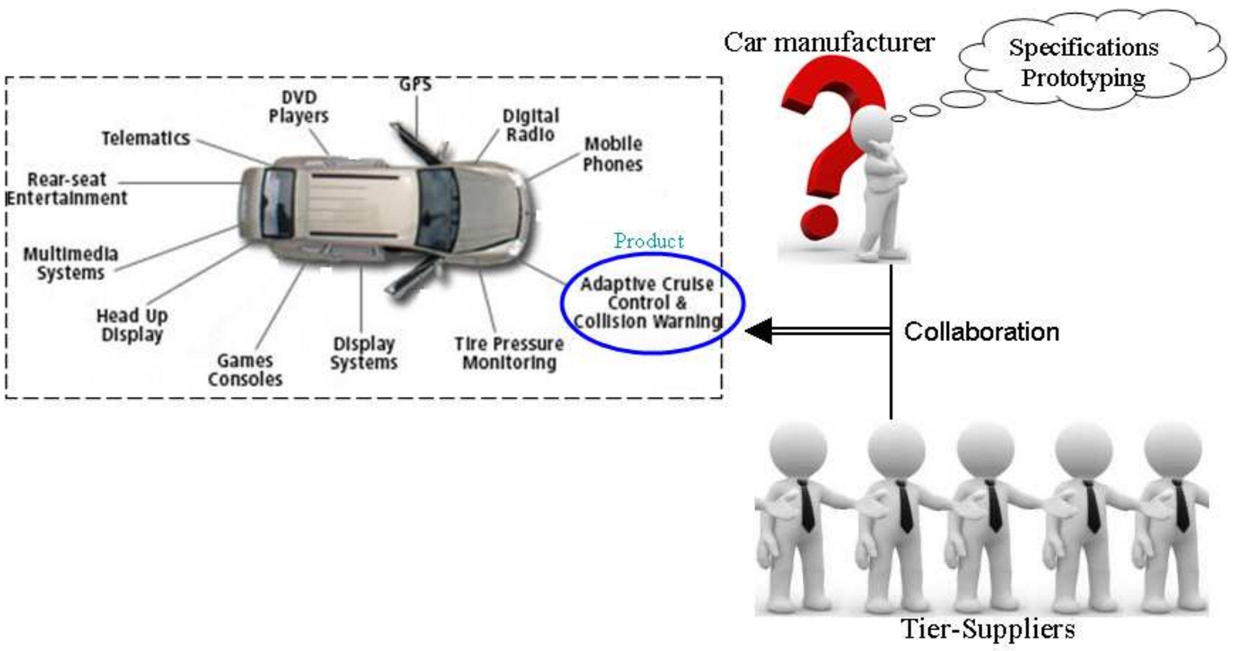


Figure 2.4: Interaction between car-manufacturers and their suppliers

Since, software and hardware technologies become a dominant factor for the system complexity, sophisticated development approaches for complex embedded systems have been developed. Then, as different types of computer systems, the development of real-time embedded systems may follow different life-cycles starting from the requirement collections to the system integration, its utilization and maintenance. In the sequel, we consider the development of embedded software which is a component of the whole embedded system development elements.

III.1.a. Software development of real-time systems

The software engineering is using methodologies based on the scientific knowledge. In other words, the software engineering consists in the development of techniques and tools allowing to produce a software respecting its constraints.
A software product is a solution suggested to address some problems. Although, finding a solution does not mean the end of problems, because, there are quality criteria that may be satisfied globally or partially by a software development [Bro87]. Some of those criteria are:

- Determinism: the most important quality criteria. The software has to provide the same results under the same conditions.
- Efficiency: even if a software optimizes the time, the size, etc., the optimization has not to have a negative impact on the software, and the efficiency characteristic has to meet the user needs. This criterion is very important in embedded applications.

- Re-usability: the ability of the software to evolve or to be used at least partially in order to develop another program, without modification or with minor modifications.
- Robustness: the ability of the software to maintain its performance despite changes of operating conditions or the presence of uncertain parameters. Indeed, the software must react well when it deviates from its normal use.
- Portability: the ease of the software utilization in different implementations. We distinguish between the portability and the compatibility. The compatibility is a term rather used to describe the ability of computers to exchange data or to execute the same program.

Generally, the software development may be realized by following a cycle called the software life-cycle. The software development cycle is a set of steps transforming the requirements to an assisted treatment process. This latter is derived from users and utilization's environments to satisfy these requirements (entry points). The treatment process is decomposed to a set of steps in order to facilitate the monitoring, the verification and the validation of the project.
Despite the differences in the development cycles, all of them consist of several phases which are staggered by time:

- Requirements specifications;
- Preliminary Design;
- Detailed design;
- Implementation;
- Integration;
- Validation;
- Exploitation and utilization.

Each phase ends with a set of deliverables that have to be validated by the responsible actors (i.e. designers for design phase, developers for implementation phase, customers for exploitation phase, etc.). Then, the development process moves to the next phase by using the deliverables of the previous phase as inputs.
There are many kinds of software development cycles, like the V-shaped approach, the spiral-shaped approach, the waterfall approach, etc. Hereafter, we introduce the V-shaped approach as an example of the software development cycle.

III.1.b. Example of a development-cycle: V-shaped approach

The V-shaped approach is one of the most famous software development life-cycle. It has been sharply adopted in industrial domains, especially via the technical processes defined by ISO 15288/NF Z 67-288 standard [dS13].
The V-shaped life-cycle is composed of two slopes: a downward slope and an upward slope. The downward slope shows the progressive stages from the specification to the implementation. The upward

slope depicts a set of tests to verify and/or validate the stages of the downward slope. Moreover, every step from the downward slope is matched with a step from the upward slope (see Figure 2.5). So, we can say that the V-shaped life-cycle focuses on the verification (e.g. testing) and validation.

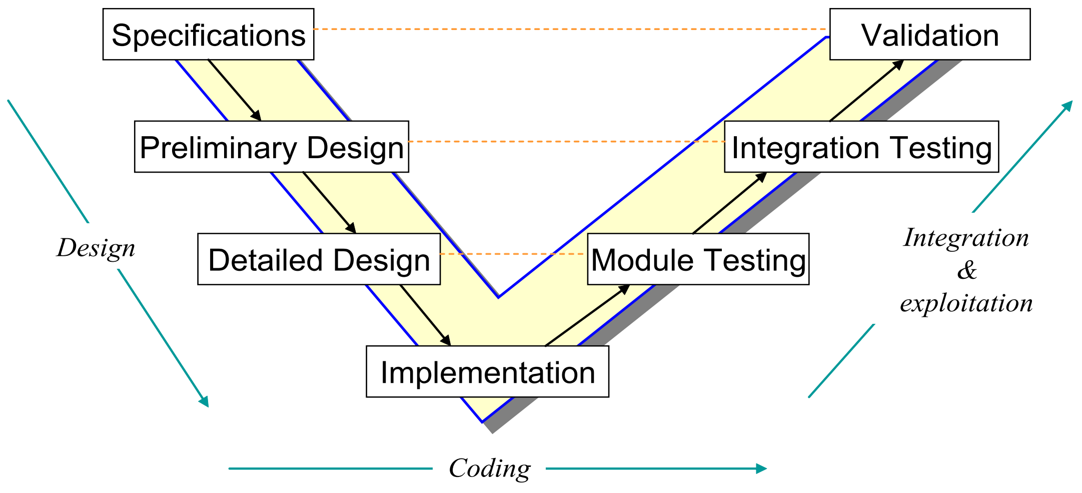


Figure 2.5: V-shaped approach

The steps composing the V-shaped life-cycle are detailed as follow:

- Specification: this phase is defining the system requirements. The specification is expressing what the system must carry out and the qualities that the system must have. We separate the requirements into two categories: functional and non-functional (also called extra-functional) requirements. While functional requirements define the system capabilities, non-functional requirements focus on performance, design and quality constraints. Indeed, for real-time systems, the timing constraints represent a sub-set of the non-functional requirements. The specification can be expressed informally in natural language, but it can be also expressed in a formal or semi-formal language.

- Design: this phase can be divided in two steps, preliminary design and detailed design. The design phase defines the internal structure of the system referring to the requirements that are previously imposed. The preliminary design leads to a general system architecture composed by a set of components, modules and concurrent tasks. Due to the specificity of the real-time systems, dividing the architecture into modules and concurrent tasks enables to obtain static and dynamic system views. Detailed design permits to detail the preliminary design of the system until getting a detailed description of each component and specifying data structures, communication protocols, tasks content, etc. At the design phase, it is possible to use many design formalisms.

- Implementation: this step is used to implement the concepts proposed via the design phase. Through the implementation step, every component is encoded in a programming language. Indeed, for real-time systems, the programming language can be a concurrent language (e.g.

Ada) or a sequential language used in conjunction with a real-time operating system (e.g. a real-time executive with C).

- Module Testing: during this phase every independent component is checked in order to verify the respect of its specifications and the correct behavior under all circumstances.
- Integration Testing: this phase regroups tests according to the module testing phase. Moreover, the integration testing step checks the components interactions, the component relationships, the task functionalities, the communication mechanisms, the task synchronization, etc.
- Validation: it is a phase for checking that the developed software respects the requirements expressed in the specification phase. Furthermore, the end-customers can check if the developed software responds to their expectations.

The development efficiency differs from an approach to another one. Although, if an error occurs in one of the early phases, it would spread to the other successive phases, and changes may not be taken into account until the end of the last phase. This leads to an additional cost of the software development especially in terms of time-to-market.

III.1.c. Time-to-Market concept

"Time-to-market" is a concept representing the duration required to develop and commercialize a product. Nowadays, with the rapid development of technology and the growth of users demands, "Time-to-market" becomes an important criterion contributing in the product success, particularly in areas where demands and norms evolve rapidly.

III.1.d. Hardware and software parts

Since a real-time system is composed from the real-time software and the hardware components, the hardware part has to be taken into consideration during the life-cycle development. We recall that a part of real-time constraints is related to the architecture platform on which the software will be executed. Indeed, considering the hardware part during the development enables to get a final product respecting its real-time constraints.
The narrow link between the software and the hardware means that the software development of real-time systems is often influenced by the hardware. Although we have focused on the software development of real-time systems, the complexity of hardware part and software part of a real-time system requires more suitable development approaches. As a solution, more sophisticated development approaches called hardware/software co-design methodologies have been proposed [LHG98]. Figure 2.6 depicts one of the traditional concurrent development process. The hardware/software co-design approach focuses on the combination of hardware and software prospects at an early development stage. The needs are met by exploiting the synergy between hardware and software through their concurrent development.

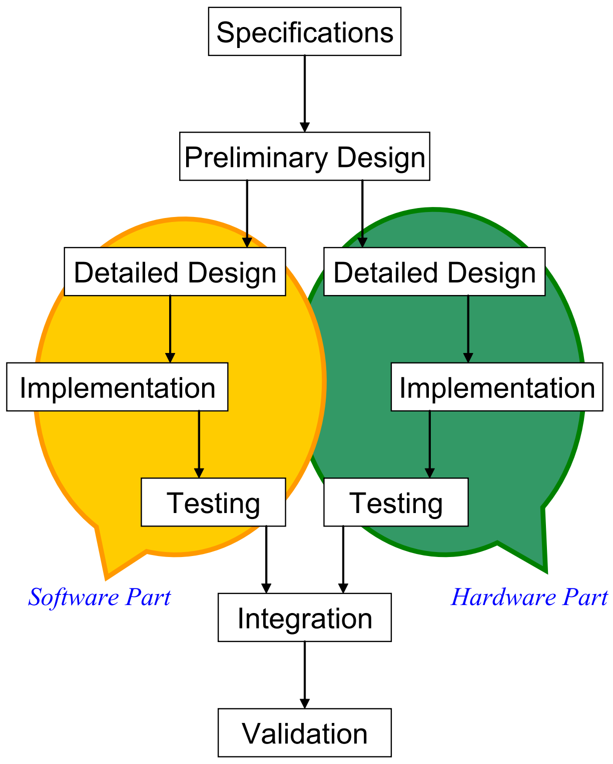


Figure 2.6: General hardware/software development approach

III.1.e. Discussion

Recently, major advances have been announced. They are related to several methodological aspects regarding the system design and its correctness referring to its requirements. Conventionally, the verification is done at a very late stage of the system's life-cycle. However, in case of an early development flaw, the cost of backtracking in the cycle is usually very high. In a number of cases, the malfunction is due to the design errors.
Moreover, formally, the integration of the hardware/software subsystems was a very hard task to be managed by manufacturer's designers. However, in complex industrial systems (e.g. avionics, automotive, etc.), many modules developed for a special product are reused for another ones. Therefore, at the design stage, designers already have a number of information for a primary system analysis. Yet, the time partitioned systems that have appeared in the aerospace domain due to the Integrated Modular Avionics (IMA) [C$^+$97] need an early dimensioning during the design phase [CRS11].
Hence, currently, several actors of real-time systems field apply the verification methods at an early stage of the development, in particular during the design phase [HS09, APK$^+$11, ZBG$^+$08].

III.2. Software Design of real-time systems

Experiences on real-world applications shows that software development costs are sharply impacted by wrong design choices made in the early stages of development but often detected after implementation.

Several studies have reported that 80% of the money spent in development goes to correcting defects [NIS02], and most timing-related vulnerabilities are detected very late in the development process (i.e. during the implementation phase or the system integration phase). Accordingly, the design phase of real-time systems has been recently treated as one of the earliest stage where the validation of timing properties can save over 50% of the costs required for the developement process [SZP$^+$03].
Consequently, in the current work, by the software design of real-time systems we mean (i) the modeling step and (ii) the real-time scheduling analysis step of those systems. The real-time design is an iterative process where modeling and analysis steps complete each other[2].

III.2.a. Design methodologies of real-time systems

The modeling is an abstraction level in the development flow. So, we can say that a model is an abstraction, a simplification of a system that is sufficient to understand the modeled system and answer the questions that are raised about its performance. Since real-time software is typically a set of communicating concurrent processes, through models and the modeling process, designers (or modelers - to be more accurate -) extract the basic system parameters, especially the timing ones. Several methodologies have been proposed for modeling real-time systems [Gom08]. Below, we introduce some methodologies items:

- Component based-design [HNN05]: a software component is a module or object which performs a specific function according to a set of specifications. Components are software units with provided and required interfaces. Every software component is used to encapsulate some functionalities which are only accessed via the component's interfaces. In case of real-time system's design, the component contains also introspective interfaces, that are used to retrieve non-functional properties like timing requirements. The whole system is built by assembling the set of components and connecting their interfaces. Hence, the application can be split into clearly separable and reusable blocks, improving the organization of the product as well as its reusability and modularity.

- Data-flow design [Gom84]: various approaches based on the data flow methodology aim to model the system functions as well as the data-flows and data stores between functions. Several methods serve designers to decompose the system into modules with a high cohesion and low coupling. Typically, a design of a real-time system following a data-flow methodology is composed of a high level control oriented sub-system which executes different data processing corresponding to each state of the system. Furthermore, some researches consider that it may be important to study separately control and data parts. This separation gives a more structural view of the model and facilitates the modification and the re-use of different parts.

- Platform-based design [SVN07, SVCBS04]: foundations of platform-based design consist in providing a high level of abstraction by modeling only the important system parameters, and limiting the design space exploration to a set of available components. Platform-based design elicits identified abstraction layers. This is allowing a separation of concerns between functional architectures and abstractions of possible implementations. For instance, a platform-based design methodology decouples software layer from changes in hardware layer. Indeed, platform-based

[2]In the literature, the terms “design” is often used to only indicate the modeling process.

design methodologies are carrying out the design as a sequence of refinement steps by using platforms at various level of abstraction. In other words, modeling goes from an initial abstract model related to the application towards the final specific model containing detailed properties about the platform. Then, the whole system modeling is a process consisting of mapping the application model to the architecture platform.

Each design approach following one or several above-mentioned methodologies has its advantages and its drawbacks. At this stage, we do not emphasize on the utility of each approach rather than what we expect from a design approach during the modeling phase. Thus, whatever the methodology is, we expect the following characteristics:

- A separation between the design level and the implementation level, defining semantics of abstract descriptions.
- A close connection between software and hardware parts (see Figure 2.7) by saving the independence of each part to ensure the maintenability and extendibility of each part. That can be achieved by using, for instance, intermediate interfaces.
- Taking the quality-of-service (QoS)/non-functional properties into account, in particular, the timing properties in order to obtain an analyzable design (i.e. a design which provides all necessary elements to check the timing constraints).

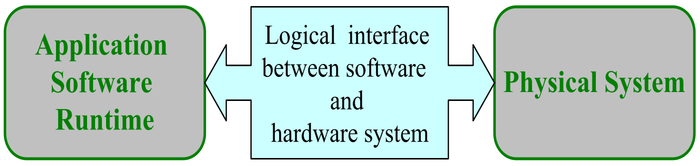

Application Software Runtime
Logical interface between software and hardware system
Physical System

Figure 2.7: Real-time system's design: elements of software system architectures

We have just introduced some traditional families of methodologies without comparing their characteristics. Due to the diversity of the modeling approaches, a need is perceived and consists of performing the temporal verification of real-time systems independently of the modeling approach used. We devote Section III of Chapter 4 to highlight some standard modeling languages in which we are interested. Once the modeling is made, the analysis step may be launched in order to detect inconsistencies, errors, and omissions in the requirements. In our work, we are interested solely in the real-time scheduling analysis.

III.2.b. Temporal verification of real-time systems

To verify a system, testing can be used. Testing [Hoa69] is the process of executing a program to check that it satisfies specified requirements, and also to identify differences between expected and obtained results. However, testing can be used to show the presence of bugs but never to show their absence. In addition, bugs may be found during later developing process stages. So, this kind of techniques is not

sufficient in the context of hard real-time systems. In order to avoid any impact on the time-to-market and the reliability of critical system, the early verification can be used.
Referring to the context of real-time systems (when the verification is a part of the real-time scheduling analyses), the verification is based on three main approaches: model-checking approach, simulation approach and analytical approach.

- Model-checking methods represent an approach using the formal verification. Model-checking methods are based on an exhaustive exploration of all the system states defined by an abstract design. The design is expressed in a formal language such as Petri nets [CEP03], temporal logic [CES86], timed automata, etc. While, the model checking methods are dedicated to finite-state systems, a system with a very high number of states may lead to a state explosion problem. Generally, the analyses via model-checking methods are safe (i.e. conservative).
- Simulation consists in studying the behavior of the system. Generally, a simulation is to run under some specifications in order to verify the performance in the most significant system conditions. Nevertheless, it is impossible to perform exhaustive simulations for every possible system's state. The simulation is verifying the most representative cases. So, the analyses via the simulation are not safe in general, but they can be considered as safe under some conditions.
- Analytical methods consist in analyzing the system with a set of equations. Mathematical equations are used to calculate various criteria like the processor utilization, tasks response-time, etc. Thus, giving the results computed by these equations, the system's behavior is described. When a design complies with the conditions of some analytical methods, the scheduling analysis may be performed by using these convenient methods. Generally, the analytical methods are safe.

Whereas we are interested in all these kinds of temporal verifications, in this thesis, we emphasize the analytical methods based on the scheduling theory. The scheduling theory proposes several analytical methods and simulations in order to perform the analyses efficiently. Hence, the scheduling analysis might lead the analyst to rewrite some specifications, to reassess the initial analysis, or to correct and refine the design's requirements. Therefore, Section 3 is devoted to present an overview of the real-time scheduling theory.

III.2.c. Illustration: real-time systems design in automotive domain

The quest for a rapid new system introduction has induced most car-makers to change the scope and the structure of their development process. In general, the relationship between the car-maker and its suppliers can be as [Vol04]:

- Supply proprietary which is a set of standard items based on a proposition made by the supplier.
- Black box: in which the car-maker defines the functional features, but the product development responsibility is totally left to suppliers.
- Concurrent engineering is a mutual exchange of competences and information between the car-maker and a tier-supplier for a common system's development. Concurrent engineering development is considered a very promising way in order to achieve relevant results in time-to-market shrinkage and cost reduction.

In the context of real-time systems, the manufacturers adopt increasingly the introduced design methodologies for improving the quality and the re-usability. They intervene during two particular development steps: the design of systems and their integration into vehicles. Moreover, the vehicle development may take more than 24 months and the cost of embedded systems represent about 25% of the car manufacturing cost [PBKS07]. Hence, due to the concurrency pressure, manufacturers may specify the design and delegate the implementation of real-time systems (or subsystems) to various suppliers that ensure the software coding by respecting the specified design. Manufacturers may also follow some standard methodologies like AUTOSAR [AUT] in order to maximize the product margin.

III.2.d. Conclusion

To summarize the importance of multi-partner development (especially in industrial domain), Figure 2.8 depicts actors and views related to the real-time design phase. So, Figure 2.8 shows different roles which can be played by a system designer depending on the nature of tasks and the intervention stage. Generally, the system designer can be considered as a model designer when modeling functional parts and platform architectures of systems. Yet, the system designer can be viewed as an analyst during the analysis phase.
Indeed, referring to different elements tackled until this stage (real-time notions and design methodologies), and regardless of actors affiliation (manufacturer or supplier), it is helpful to introduce three main actors and their design views related to the design of real-time systems.

Software-Hardware Architecture Generally, architects focus solely on the non-functional modeling and the system analysis. Thus, architects have architectural views of the real-time systems. When the software and hardware modeling processes begin with the immature specifications of the system, a parallel development of the incomplete specifications requires a close collaboration in particular with the analysts.

Temporal Verification: Scheduling Analysis Before starting the timing verification of a real-time system. Analysts have first to extract the system requirements from the system modeling (done by the architects). Then, this latter has to be implemented using a specification formalism. Since the architect may not be knowledgeable in the scheduling analysis area, the analyst collaborates with him/her to write the requirements and system specifications. Both actors work closely to ensure that the specifications reflect the real requirements and system's behavior. Once these specifications are taken into consideration by architects during the modeling phase, the analyst can verify whether the system satisfies the requirements.

Function Modeling The existence of a function modeler depends on the domain, the competence and the strategies followed for developing real-time systems. It is known in industrial sectors using real-time systems that suppliers are characterized by high specialized technical skills. So, the function modeling (based on functional requirements) depends on the followed design methodology and the relationship between the architect and the function modeler (e.g. the relationship between manufacturers and their suppliers). For instance, in case of a top-down methodology (i.e. from architecture to business model), the function modeler depends on the provided architecture. Otherwise, the architecture

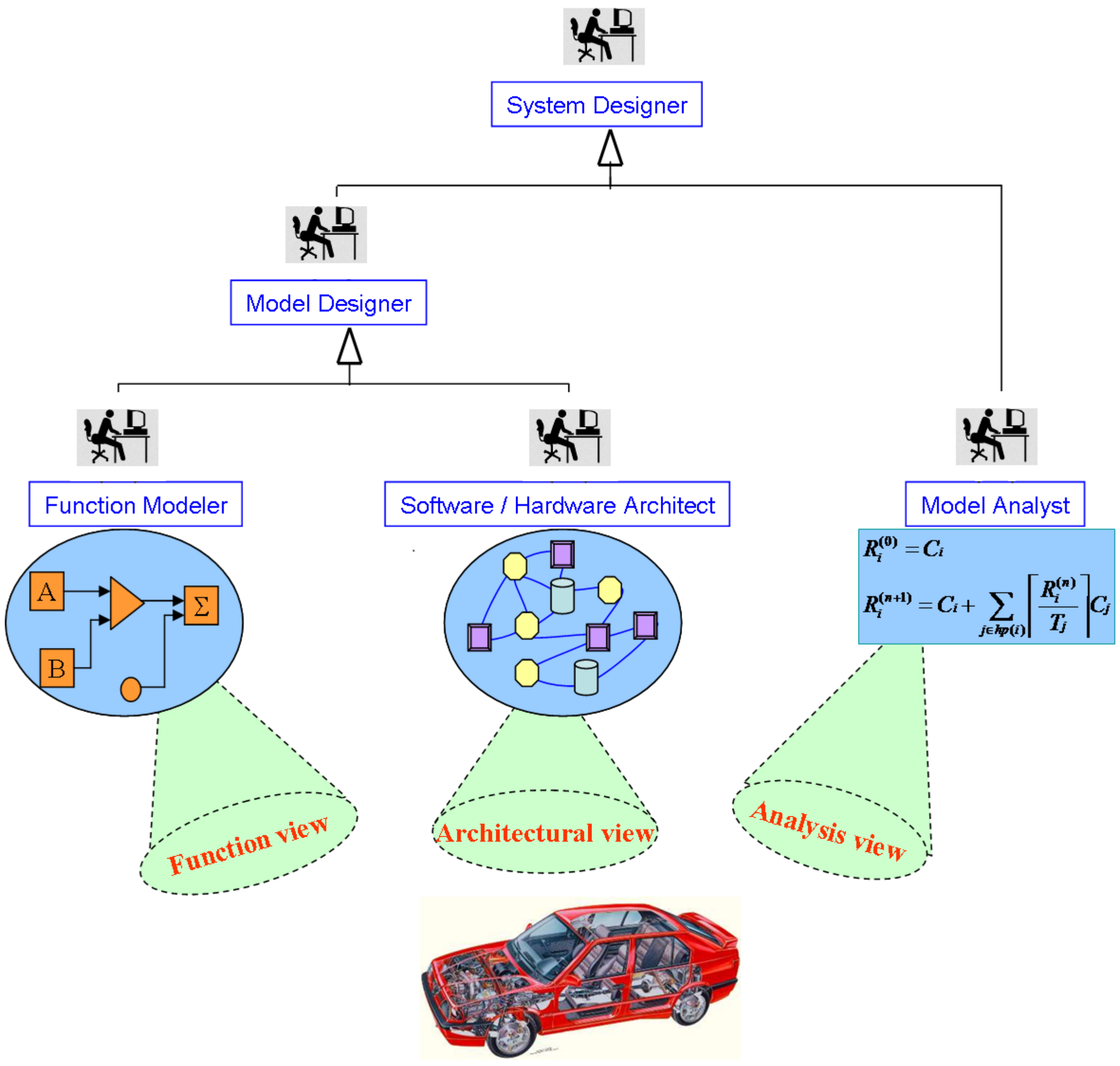


Figure 2.8: Different real-time system's views depending on each actor

modeling depends on the functional model proposed by the function modeler.

Chapter

3

Real-Time Scheduling

Contents

Abstract

This chapter presents briefly different aspects related to the real-time scheduling theory. The common timing properties and constraints determining the behavior of real-time systems are tackled in the first section. The second section gives an overview of different kinds of analyses studying the feasibility and the schedulability of real-time systems.

I. Introduction

Our goal is not to provide an exhaustive study about analysis models and methods, but to emphasize guidelines leading to the analysis situation of a real-time system and its corresponding tests.
Elements aiding to identify the context of real-time systems are tackled in Section II. Section III gives an overview of different kinds of analyses studying the feasibility and the schedulability of real-time systems.

II. Real-Time Scheduling-Aware Context

In order to enable real-time scheduling analyses we need to have system models that facilitate such analyses. In this work, we refer to such models as analytical models. The analytical model of a real-time system supplies the information necessary for performing the required analyses. As we are interested in the temporal behavior of real-time systems, this section is devoted to introduce some aspects related to the analytical models.

II.1. Real-time tasks: description, characteristics and definitions

II.1.a. Real-time tasks description

A task, also called thread, is an active entity of the real-time application. It represents the sequential execution of a set of instructions corresponding to one or many functions. A task is executed when a specific event occurs. Each time a task recurs is called an “instance” of the task or a “job” of the task. According to the type of the hardware architecture, jobs may be run on one or different processors. However, in a given instant, a processor (i.e. a processing core) is able to execute only one job. In the sequel, by the word “processor“ we mean a processor with one core.

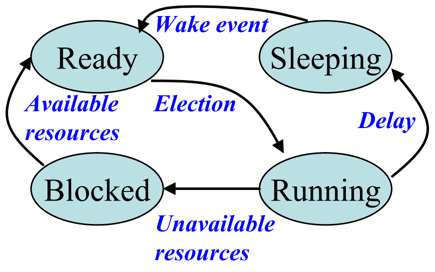


Figure 3.1: Different possible task states

Every job is related to a data structure containing its runtime state. The transition from a job state to another is ensured by the real-time operating system following a state transition diagram (see Figure 3.1). The state of a tasks may be among the following common states (states may differ according to real-time operating systems):

- Ready: the task is ready to run when obtaining a processor.
- Blocked: the task is waiting for one or more events or resources (shared resource, message, interrupt, etc.) to be available.

- Sleeping: the task is sleeping and waiting for a wake event.
- Running: the task is active and executing on a processor.

II.1.b. Real-time tasks characteristics

In the literature, the execution of a task τ_i is frequently depicted by a time-line or a Gantt diagram as shown in Figure 3.2. This latter contains a part of the main properties, like:

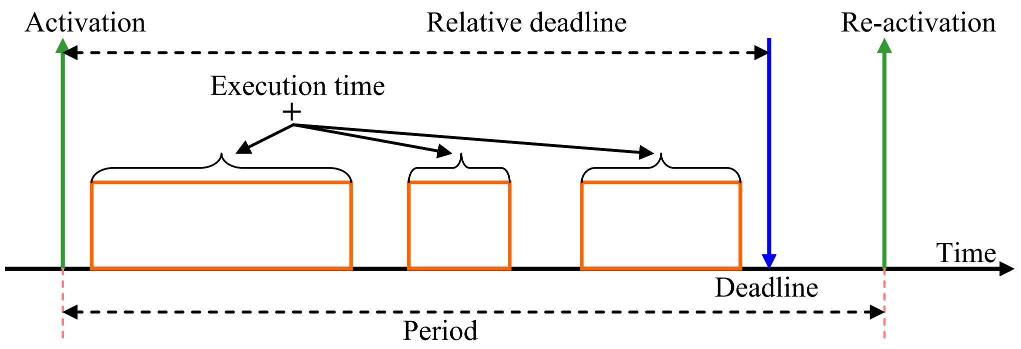


Figure 3.2: Usual graphic representation of a real-time task

- Release time r_i: time of creation of task τ_i, it corresponds to the activation instant of the first job. If the release time is a known value then the task is said concrete.
- Activation pattern (Periodicity): it represents the execution frequency of the jobs of task τ_i. Generally, three periodicity kinds are known:
  - Periodic task: periodically, a new job is released referring to a fixed time interval T_i. Periodic tasks are usually activated by clock-based interrupts. Let $\tau_{i,j}$ be the j^{th} instance of the periodic task τ_i, and $r_{i,j}$ its release time, hence $r_{i,j} = r_i + (j\text{-}1)T_i$.
  - Sporadic task: task that recurs at uncontrolled instants, but two successive releases are characterized by a minimum separation interval T_i. The fast activation pattern of a sporadic task corresponds to a periodic activation. Indeed, every job $\tau_{i,j}$ is characterized by a release time $r_{i,j}$, where $r_{i,j+1} \geq r_{i,j} + T_i$, and $r_{i,1} = r_i$.
  - Aperiodic task: task which may arise at any instant, without any minimum separation duration between two consecutive instances. The aperiodic tasks are often activated by external events. Thus, $r_{i,j+1} \neq r_{i,j}$, and $r_{i,1} = r_i$.
- Execution time C_i: represents the time needed for executing each job of τ_i on a processor. The execution time of tasks and the hardware execution resources are strongly coupled. Generally, the execution-time of a task is represented by the worst-case execution time (WCET) which is the upper bound of all possible execution times of any job. The execution time is an important information for analysis methods and tools. Some research studies consider also the best-case and the average execution time values (BCET and AET). In case of mixed criticality systems

[Ves07] (appeared in avionic systems according to the safety standards like RTCA DO-178B), a task may be characterized by L execution time values, where L represents the number of the criticality levels considered by the system. Although we are not interested in the estimation of the worst-case execution time in this work, there are two types of the estimation methods (also known as the timing analysis):

- Static analysis: it is performed by analyzing the code and counting the number of clock cycles that a job needs to execute. Moreover, a task is represented as a graph flow. Then, the worst-case behavior of a task leads to know its worst-case execution time [CP00].
- Dynamic analysis: it is based on extensive testing. The dynamic analysis is based on the measures performed during task executions or simulations [RS04]. The major drawback of the dynamic analysis is that it does not produce safe results.

Calculating the worst-case execution time is a non-trivial problem and had led to many works. The execution time evaluation may be very pessimistic because of some factors, like the existence of hardware caches or system interruptions. The knowledge of this parameter is indispensable for the hard real-time systems analysis, which is using it as an input.

- Relative deadline D_i: is the time devoted to task τ_i to finish its execution. Every job $\tau_{i,j}$ of task τ_i must finish its execution before D_i time units after its activation.
- Utilization: it is the average time for which a task executes per unit time interval. For periodic task τ_i, the utilization $U_i = C_i \ / \ T_i$, where C_i is the worst-case execution time and T_i is the period of τ_i. The greatest utilization for a sporadic task is also $U_i = C_i \ / \ T_i$.

II.1.c. Real-time characteristics related to the system execution

While the behavior of the task-set is characterized by several properties, Figure 3.3 shows some of them. They are related to the tasks execution and/or derived from the properties already presented.

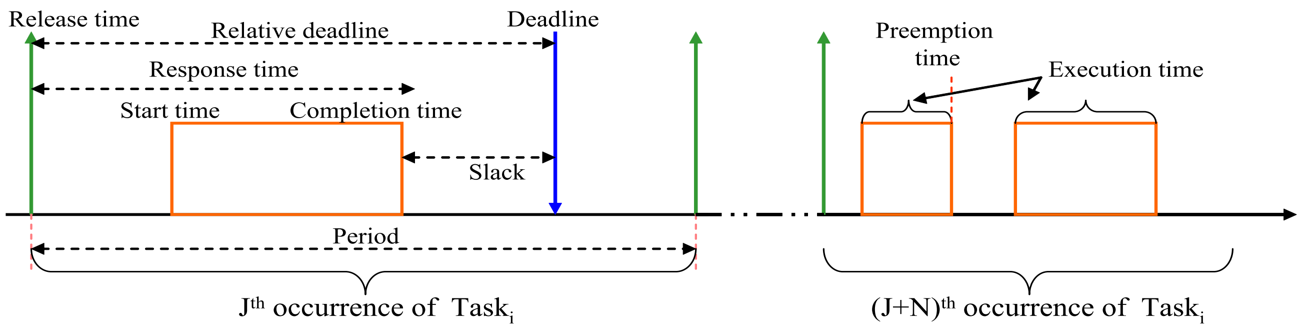


Figure 3.3: Usual graphic representation of a task execution

Jobs properties

- Start time: time at which the job starts running.

- Completion time: time at which the job finishes the execution.
- Preemption time: instant where the job is interrupted in order to execute for example a higher priority job (i.e. instance of another task). A job can be preempted several times during its execution.
- Deadline of a job $d_{i,j}$: end of the allowed time frame where the job has to be executed. Usually, the deadline is obtained from the job release date plus a relative deadline. So, $d_{i,j} = r_{i,j} + D_i$. A typical timing constraint is that the completion time must occur before or on the deadline.
- Lateness of a job: it is the interval between the completion time of a job and its deadline. A negative lateness signifies that the job respects the deadline.
- Laxity of a job $L_{i,j}(t)$ (or slack time): remaining time to deadline minus the amount of remaining execution. If this parameter is null at a given time t, the job should be executed without interruptions, otherwise the deadline will be missed.
- Response time of a job $TR_{i,j}$: duration required for the job to produce its results. The response time is obtained by the completion time minus the release time, and contains also the duration related to the blocking on shared resources and the duration of the preemptions. The job respects its deadline $d_{i,j}$ if $TR_{i,j} \leq D_i$.

Tasks properties

- Laxity $L_i(t)$: it is the duration when the task cannot be executed without missing its deadline.
- Response time TR_i: The worst-case response time of a task is the greatest value among its jobs response times. Thus, $TR_i = \max_{\forall j} (TR_{i,j})$. A task is schedulable if $TR_i \leq D_i$.
- Practical factors: it is important to take into consideration several practical factors related to the system execution, for instance:
  - Execution modes: we distinguish four possible execution modes of a real-time task according to the preemption and the suspension. Then, the first three modes concern the preemption, and the last one concerns the suspension.
    - Preemptible task: every job can be preempted by the scheduler at any time.
    - Non-preemptible task: every job cannot be preempted once it starts executing.
    - Co-operative task: the task can only be preempted at specific instants within its execution. Then, each job execution is composed by series of non-preemptible sections.
    - Self-suspending task: the task may suspend itself. The self-suspension delays in the task behavior represents essentially the waiting time due to the execution of external operations.
  - Jitter: the term jitter means the variability in time. A task that is expected to be periodic can have a release jitter. Then, the value of the release jitter is used as a bound on the maximum deviation from the ideal task period. The release jitter can also be related to other factors such as network congestions.

- Precedence: a job must wait for the result provided by another job (instance of another task) before being executed. Generally, the precedence relationships of a task-set are represented by a directed graph where the vertices represent tasks and the edges represent the precedence relations.
- Shared resources: tasks often need to share some software resources (e.g. memories access) or hardware resources (e.g. sensors). We talk about a critical shared resource requiring a mutual exclusion, when this resource cannot be used by more than one task simultaneously. Thus, the critical section is the piece of code belonging to a task that is executing under mutual exclusion constraints enforced by a synchronization mechanism (e.g. semaphores introduced by Dijkstra in 1965).

II.2. Real-time tasks system: the task-set characteristics

A tasks system is the execution of a set of tasks subjected to several timing constraints. In other words, the tasks system is a Cartesian product of all temporal properties characterizing the tasks of the software application. Therefore, tasks may be classified using different characteristics such as: deadlines, release-times, interdependency relationships, etc.

Deadlines Referring to the relative deadlines, task-set may be among one of the following categories:

- Implicit-deadlines case: deadlines of the tasks are exactly equal to periods;
- Constrained-deadlines case: deadlines of the tasks are less than or equal to periods;
- Arbitrary-deadlines case: deadlines of the tasks and periods are unrelated;

Thus, the implicit-deadlines case is a particular case of the constrained-deadlines case which is also a particular case of the arbitrary-deadlines case.

Release times A task-set may be concrete or non-concrete depending on its release times:

- Concrete/Non-concrete task-set: if the release-time of every task is known, then the task-set is concrete. Otherwise, the task-set is non-concrete.
- Synchronous/Asynchronous task-set : when the task-set is concrete, if all the tasks are released at the same time, then we talk about a concrete simultaneous (i.e. synchronous) task-set. Otherwise, the task-set is concrete and asynchronous.

Execution times Every task forming the task-set may be characterized by several types of execution times. For instance, the execution time may be:

- An interval of values between BCET and WCET values;
- A set of values for each criticality level;
- Probabilistic set of values;

- An unknown value which must be calculated. So, the value has to be computed depending on the other task-set parameters [ZBB11].
- etc.

Interdependency relationships According to the precedence relationships and the existence of shared resources, a task-set is categorized as follows.

- Independent task-set: all tasks are independent from each other.
- Dependant task-set with precedence relationships: some tasks have to expect messages or synchronization signals coming from other tasks.
- Dependant task-set due to the existence of shared resources.

Preemption control A task-set can also be classified according to the preemptivity of tasks.

- Preemptible task-set: every task may be preempted at any time.
- Non-preemptible task-set: one task or more cannot be preempted at all.
- Limited preemptible task-set [BBY13]: an alternative between the two extreme cases of fully preemptible and non-preemptible task-set, where tasks can be preemptible, non-preemptible or co-operative.

Models hierarchy To design the real-time task systems, several expressive models were proposed over the years and many of them allow an efficient analysis.
In the 1970s, Liu and Layland proposed a model which is characterized by few behavioral task parameters (execution time and period of task activations). It allows a simple efficient analysis, but it is very limited in terms of realistic tasks. Since then, many models have been proposed.
Early works have proposed their models as a generic model. The generalized multiframe model (GMF) [BCGM99] was proposed as a generalization of the sporadic model [Mok83] and of the multiframe model of Mok and Chen [MC96][1]. This latter is generalized in that the deadlines of the internal frames are allowed to differ from the minimum frame separation, all the frames do not have the same deadlines, and the minimum frame separations are not identical. The transaction model is another generalization of the sporadic model. In that model, the transaction release times are not known a priori [PGH98, RGRR12]. The recurring branching model is a further generalization of GMF [Bar98]. It allows to model some restricted forms of conditional real-time code ("if-then-else" and "case"). This type of model was further generalized to recurring model [Bar03] by proposing different job types that can be released by a task. Another generalization of the GMF model is the non-cyclic GMF model [TMNLM10] allowing to release the jobs in any order, thence the behavior is not strictly cyclic. The two last generalizations were unified to introduce the non-cyclic recurring model [Bar10]. The non-cyclic recurring model was also generalized by the digraph model (DRT), which is one of the most general models [SEGY11] currently known. It models each task using an arbitrary directed graph for

[1]Readers can see Appendix A showing some examples of multiframe and GMF models

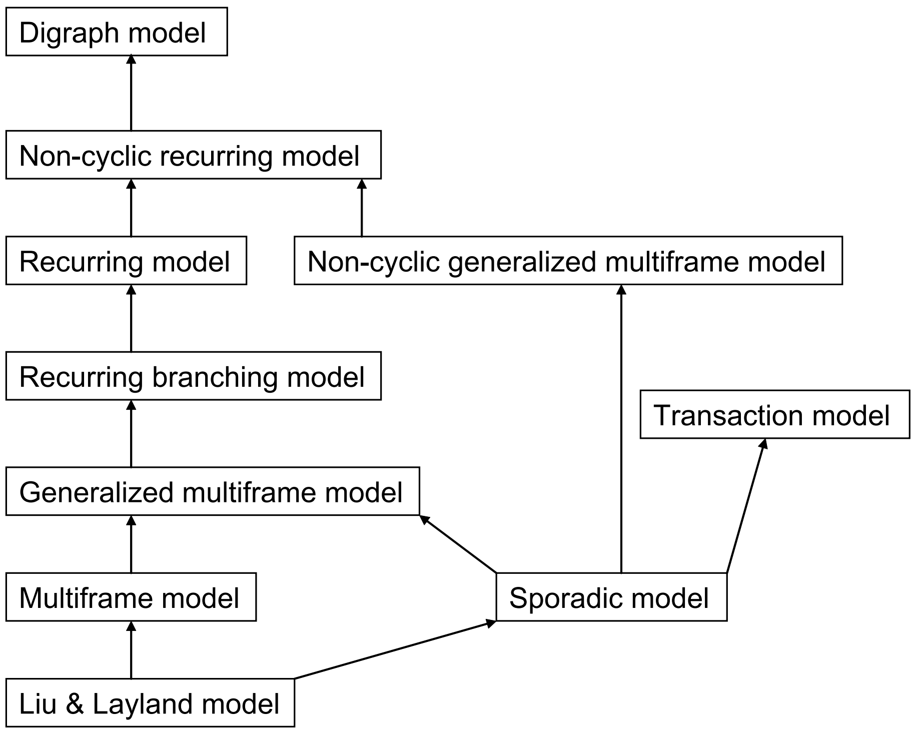

Digraph model
Non-cyclic recurring model
Recurring model
Non-cyclic generalized multiframe model
Recurring branching model
Transaction model
Generalized multiframe model
Multiframe model
Sporadic model
Liu & Layland model

Figure 3.4: A hierarchy of various models

job releases.

Figure 3.4 shows the generalization relationship between various models indicated by arrows (i.e. A $\rightarrow$ B means that B is a generalization of A). These models were introduced in the uniprocessor case, and most of them with independent tasks, unable to suspend themselves. Some of these models are or have been investigated in larger architectures such as in the distributed systems, or in multiprocessor systems for partitioned, semi-partitioned and global scheduling [QFGM13].

II.3. The hardware execution resources

The physical part of a real-time system contains a set of hardware elements like processors, memories, inputs/outputs, etc. As we are interested in the execution resources, the classification below depends on the number and the type of processors, and it is driven by a scheduling analysis point of view.

- Uniprocessor architecture: the architecture contains only one processor.
- Multiprocessor architecture: according to the nature of the available processors, the multiprocessor architecture can be:
    1. Homogeneous: in this case, the processors are identical, hence the processors are interchangeable and they have the same speed. Then, the execution rate of all tasks is the same on every processor.

2. Uniform: the processors can execute the same tasks. Moreover, the execution rate of a task depends only on the speed of the processor. For example, a processor of speed 2 will execute all tasks at exactly twice faster than a processor of speed 1.
3. Heterogeneous: the processors are different and the execution rate of a task depends on both the processor and the task, because every task has different computational requirement on each processor.

- Distributed architecture: the distributed systems are defined by a set of tasks running on processors and exchanging messages. The network is the only way of communication between processors. It represents a shared resource for tasks communication.

II.4. Schedule and Scheduling

Basically, a *scheduler* is a module implementing an algorithm (or a policy) ordering the execution of the outstanding tasks on processors according to some criteria.
A *schedule* produced by a scheduler is an assignment of a subset of ready jobs to available processors. Then, a *schedule* for a task-set is one where every job is assigned to at most one processor at a time, every processor executes to at most one job at a time. A schedule is called a *feasible schedule*, only if every job in the schedule starts at or after its release time and completes by its deadline and all job constraints are satisfied (e.g. resource constraints, precedence constraints, etc.).
Scheduling algorithms can be classified into several categories according to the knowledge they require to operate, the type of parameters they use, and their ability to interrupt tasks during their executions.

II.4.a. On-line and off-line scheduling

Definition 1. *(Off-line scheduling) An off-line scheduling algorithm provides a feasible schedule in advance. It presupposes the knowledge of tasks characteristics, and in particular their exact activation patterns.*

The major drawback of this kind of schedulers is their stiffness and their inability to be adapted to the variations of the environment. The advantage of these methods is that the schedule is correct by construction.

Definition 2. *(On-line scheduling) An on-line scheduling algorithm is a strategy executed by a scheduler, in order to share the processor(s) between the active jobs.*

On-line algorithms are flexible, but in order to validate a system, a suitable schedulability test has to be checked at the design stage (see Section III).

II.4.b. Feasibility and Schedulability

Definition 3. *(Feasibility) A task-set is said to be feasible with respect to a given system if there exists at least one scheduling algorithm that produces a feasible schedule.*

Definition 4. *(Schedulability) A task is said to be schedulable according to a given scheduling algorithm, if its worst-case response time under that scheduling algorithm is less than or equal to its deadline.*

Likewise, a task-set with respect to a given system is referred to as schedulable according to a given scheduling algorithm if all of its tasks are schedulable.

II.4.c. Priority-based algorithms

Many on-line algorithms are priority-based to define an order of ready tasks. Thus, the task(s) with the highest priority can be elected by the scheduler. The specification of such algorithms is to define a priority assignment strategy. The priority assignment can be made once and for all or progressively over time. Hence, we introduce the following classification:

- Fixed task priority: the priority of each task is calculated from its static parameters (like the period). Then, the priority assignment is done before starting the application. Examples of fixed priority scheduling algorithms are: Rate Monotonic (RM) [LL73], Deadline Monotonic (DM) [LW82], etc.
- Fixed job priority: the jobs of a task may have different priorities, but each job has a single static priority. Example of a fixed job priority scheduling algorithm is: Earliest Deadline First (EDF) [Der74].
- Dynamic job priority: priorities of jobs may change between their release times and their completion times. Example of a dynamic priority scheduling algorithm is: Least Laxity First (LLF) [Mok83].

II.4.d. Migration-based algorithms

In case of multiprocessor systems, scheduling algorithms may also be classified according to the allocation problem (e.g. on which processor a job/task should execute). Carpenter et al. [CFH$^+$04] have proposed the following classification:

- No migration: each task is allocated to a processor and the migration is not authorized.
- Task-level migration (also known as restricted migration): the different task instances (i.e. jobs) can execute on different processors, but each instance executes only on a single processor.
- Job-level migration (also known as full migration): each job can migrate and execute on different processor, but the parallel execution of the job is not authorized.

Scheduling algorithms where no migration is permitted are referred to as partitioned, those where migration is permitted are referred to as global.

II.4.e. Synchronization protocols

Synchronization protocols manage concurrent accesses of tasks to shared resources.
Sharing resources may cause some problems like:

- The deadlock which is a situation where two or more tasks are waiting for each other to finish, but neither ever does.

- The priority inversion is a situation where a high priority task is waiting for a low priority task which is waiting for a medium priority task. For instance, let L, M, and H be tasks with priorities Low, Medium, and High. M is running and H is blocked waiting for some resource that is held by L. So long as any task with a priority higher than L is runnable, it will prevent task L, and thus task H, from running.

In order to avoid these problems, several synchronization protocols have been proposed. For example, the following protocols are widely used in the uniprocessor scheduling contexts:

- The Priority Ceiling Protocol (PCP) (also called Original Ceiling Priority Protocol OCPP) is a protocol for shared critical resources avoiding the synchronization problem especially the deadlock. In this protocol, each resource is assigned a priority ceiling, which is a priority equals to the highest priority of any task that may lock the resource [SRL90].

- The Immediate Ceiling Priority Protocol (ICPP) is identical to PCP from a scheduling view point, but it is easier to implement than OCPP.

- The Priority Inheritance Protocol (PIP) [SRL90] is a method for eliminating the priority inversion, but not the deadlocks. Thus, if a job blocks one or more high priority jobs, it would ignore its original priority assignment and it would execute its critical section at the highest priority level of all the jobs it blocks. Then, the job would return to its original priority level after its critical section.

II.4.f. Preemptive and non-preemptive algorithms

Scheduling algorithms are also characterized by the kind of events impacting their behaviors. Thus, the preemptive and non-preemptive algorithms can be distinguished:

- Preemptive scheduling algorithms: a preemptive scheduling algorithm can interrupt and suspend the execution of a task at any moment, in order to proceed to its execution later. Such behavior occurs every time a change impacts the list of ready and blocked tasks, or the priorities of the ready tasks for priority based schedules.

- Non-preemptive scheduling algorithms: once a task is elected for execution, it runs without yielding the processor.

II.4.g. Optimality

Definition 5. *(Optimal scheduling algorithm) A scheduling algorithm is said to be optimal with respect to a system if it can schedule all the task-sets that are feasible on the system.*

In other words, if an optimal scheduling algorithm with respect to a system (i.e. task model, hardware model and resource model) cannot schedule some task-sets, then no other scheduling algorithm with respect to the same system should be able to produce a feasible schedule for those task-sets. Furthermore, it is possible to find several optimal algorithms with respect to a system.

II.5. Discussion: Real-time scheduling contexts

The temporal behavior of the real-time systems is impacted by the choices of the scheduling algorithms, the task-set characteristics and the hardware execution resources. Every choice represents a system timing characteristic. In this section, we have presented several timing characteristics, which represent a set of assumptions forming the real-time system's context. This latter is needed to be known by the systems designers in order to ensure a consistent process of the real-time scheduling analysis.
To summarize, modeling the real-time system by focusing on the timing properties and the resource requirements is composed of three elements:

1. Tasks system: it describes the software application.
2. Hardware/resources model: it describes system resources available to applications.
3. Algorithms: it defines how the software application uses resources at all times (scheduling algorithms, synchronization protocols, network protocols, etc.).

III. Real-time Scheduling Analysis

III.1. Principles

The scheduling analysis should provide evidence that the system behaves as expected and produces results at the correct time. Several scheduling problems are induced due to the real-time system characteristics. They are related to the systems validation or the systems design when this latter is not set yet. Therefore, the problematic of real-time scheduling concerns three aspects:

- Decision: this aspect is requested by systems with strict constraints to verify their feasibility. The decision problem is checking if a set of tasks can be executed on a given architecture according to a given scheduling algorithm. A decision problem is a mathematical question that is defined on given parameters enabling to obtain a yes/no answer.
- Repartition: this is specific to multiprocessor and distributed architectures. The repartition consists in the allocation of a set of tasks to a set of processors.
- Optimization: it is related to the decision and the repartition aspects. For example, minimizing the number of processors needed to respect the timing constraints, or minimizing the execution-time of tasks, etc. Some optimization problems may be also viewed as decision problems.

III.2. Schedulability analysis

Schedulability analysis decides for a given task-set under certain scheduling policy, whether all deadline requirements associated with each task will be satisfied. When the scheduling policy is given (e.g. it may be determined by the used RTOS), and the question is: "is the system schedulable with the chosen policy?", hence the test used to respond to the question is called a *schedulability test.* In order to perform the schedulability test for a task-set, one has first to determine the task model corresponding to the timing behavior of the analyzed system.

Sustainability and Predictability A schedulability test is used to check the schedulability of a task-set according to a given real-time scheduling context. While schedulability tests are based on the worst-case behavior of the analyzed system, they should be *sustainable* and *predictable*. The sustainability concept has been introduced by Baruah and Burns in 2006 [BB06] and generalizes the concept of predictability. The sustainability allows to ensure that the practical system in fact would be schedulable, even if during execution the system behaves better than the worst-case behavioral task parameters considered for analysis.

Definition 6. *(Sustainability) A schedulability test with respect to a system is sustainable, if the task-sets deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual jobs are changed in any, some, or all of the following ways: (i) decreasing execution times, (ii) increasing periods or inter-arrival times, (iii) decreasing jitters and (iv) increasing relative deadlines.*

The sustainability requires that the schedulability is preserved in situations in which it should be easier to ensure schedulability, e.g. the opposite of the worst-case execution. Yet, the predictability is a specific case of the sustainability. The schedulability test is predictable when the analyzed system deemed schedulable even after decreasing execution times.

Computational complexity of the schedulability tests The computational complexity theory is developed to determine the practical computational limits of a mathematical algorithm aiming to solve a problem [Coo71]. Thus, the computational complexity is measuring the solving-time which is relative to the size of the input problem. The algorithm's complexity is often quantified by the number of instructions. Then, the complexity of an algorithm is O(F(n)) where F is the complexity function, n is the size of the input taken into account, and the (big-O) O notation is used to bound the complexity. When F is a polynomial function, the algorithm is called polynomial. A particular case of the polynomial function is when F(n) = n. Hence, the algorithm's complexity is called linear. If F(n) = n.log(n), then the complexity is quasi-linear. Yet, if F(n) = n^p, the complexity is polynomial when p is a constant. Furthermore, if F(n) = 2^n, the complexity is exponential.
To ensure the scalability of the analysis tests, the complexity is a very important aspect. Indeed, it is preferable for a test complexity to be polynomial or pseudo-polynomial at worst.

Sufficient/Necessary/Exact conditions A schedulability test is defined to be a *sufficient* condition if all of the task-sets that are deemed schedulable according to the test are in fact schedulable. A test can also be referred to as *necessary* condition if failure of the test will indeed lead to a deadline miss at some point during the execution of the system. Schedulability test that is both sufficient and necessary is labeled as *exact* condition, then it is in some sense optimal. So, a sufficient but not necessary test is pessimistic, but for many situations an exact test is computationally intractable.
In the literature, there are several complexity classes related to the decision problems, the well-known are:

- The complexity class P is the class of problems that can be solved in a polynomial time by a deterministic algorithm. Class P is the class of the so-called easy problems.

- The NP-complete is the class of hard problems, which are solvable in a polynomial time using a non deterministic algorithm.

- The complexity class NP (Non-deterministic Polynomial) includes the P and NP-complete complexity classes.

III.2.a. Examples of validation methods

The temporal validation of real-time systems requires the schedulability analysis, and it needs to have a complete knowledge about the system configuration (e.g. the scheduling algorithm, hardware architecture, task model, etc.). The validation may be performed through the simulation over a simulation duration and under some system hypothesis (e.g. the worst system behavior correspond to executing the tasks with their worst-case execution times) [Dec03].
The validation may also be performed via algebraic methods, which may be sufficient or exact schedulability conditions depending on the system conception which needs validation. Four main kinds of schedulability tests are widely used:

- Processor utilization analysis: this method is related to the utilization factor that represents the time the processor spends to execute the task-set. The processor utilization has to be under a specific utilization threshold in order to avoid the unschedulability of the system.

- Processor demand analysis: it is based on the demand bound function which corresponds to the maximum amount of tasks executions that can be released and must be finished in a time interval [LSD89, JS93].

- Response time analysis (RTA): this method consists in calculating the worst-case response time of each task in order to be compared with the deadline [JP86]. If the worst-case response time of a task exceeds its deadline, hence the system is not schedulable. In case of distributed systems, an alternative of the response time analysis technique may be used. This technique is called the holistic analysis [TC94]. It transforms the distributed system to a set of uniprocessor systems, then that eases the computing of the response times.

- Simulation: the scheduling is related to an infinite duration. Nonetheless, the objective of the simulation is to simulate the behavior of a real-time system in order to detect any temporal fault within a finite duration. This latter is related to the periodicity of the task-set and it is called the *feasibility interval* [LM80]. Generally, the feasibility interval depends on the system properties and it is based on the *simulation interval.*

  Definition 7. *(Simulation interval) The simulation interval of a real-time system is an exact or upper bound of the time interval for the schedule to repeat in a cycle.*

  Definition 8. *(Feasibility interval) The feasibility interval is a finite interval such that if all the deadlines of jobs released in the interval are met, then the system is schedulable.*

III.2.b. Examples of validation methods utilization

The choice and the utilization of each test to prove the schedulability of a real-time system depends on the real-time context of that analyzed system. As the examples below are based on the concept of worst-case scenario, hence we first provide the following notions for a good understanding. They are valid according to examples contexts.

Worst-case scenario The worst-case scenario is the execution scenario of a specific task-set considered for a validation purpose. The specificity of this type of scenario is that it represents the case when tasks have the longest response-times. The underlying idea of using the worst-case scenario is to define the worst-case behavior of a real-time system. Then, the validation of the worst-case behavior implies the validation of the system regardless of its practical executions (this is correct under some conditions presented hereafter). Traditionally, the critical instant notion is often used to well determine the worst-case scenario for basic task models.

Definition 9. *(Critical instant [LL73]) A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

Context 1:
▷ Tasks are independent, sporadic and with constrained-deadlines.
▷ The Deadline Monotonic (DM) is used as a fixed priority scheduling algorithm.
▷ The architecture is uniprocessor.
Worst-case timing behavior:
The worst-case behavior of a task is produced when the task is released simultaneously with other higher priority tasks. Moreover, the task-set is executed as fast as possible (i.e. periodically). The instant when the task is released is known as *critical instant*.
Schedulability analysis:
The test used in this case is the response time analysis which is known as an exact schedulability condition for critical instant. Furthermore, as the task-set is sporadic, the critical instant may effectively be produced during the task-set execution. Therefore, the test is exact and test complexity is pseudo-polynomial.

Context 2:
▷ Tasks are independent, periodic, asynchronous and with constrained-deadlines.
▷ The Deadline Monotonic (DM) is used as a fixed priority scheduling algorithm.
▷ The architecture is uniprocessor.
Worst-case timing behavior:
As the release times are deferred and tasks are periodic, the critical instant may not occur.
Schedulability analysis:
As it is not sure that the task-set execution achieves the critical instant, the response time analysis is a sufficient schedulability condition. In the current context, it is possible to use the simulation as an exact test since it is sustainable in this context. However, its complexity is exponential.

Context 3:
▷ Tasks are periodic, synchronous and with constrained deadlines.
▷ Tasks are sharing critical resources.
▷ The Deadline Monotonic (DM) is used as a fixed priority scheduling algorithm.
▷ The Priority Ceiling Protocol (PCP) is used as a resources access protocol.
▷ The architecture is uniprocessor.
Worst-case timing behavior:
The worst-case behavior of a task may happen if when the task is released, a shared resource is used by a task with lower priority, while achieving the longest critical section. At the moment of the release, other higher priority tasks are released.
Schedulability analysis:
It is possible to use the response time analysis with the worst blocking durations, but it is not sure that the worst-case behavior described earlier would happen. Consequently, the response time analysis is a sufficient schedulability condition, else, in the best of our knowledge, it does not exist any exact and sustainable schedulability test for the current context.

III.3. Dimensioning analysis

Verifying the temporal behavior of a real-time system may have an impact on the system dimensioning (also know as sizing) and the development cost. Even when not exact, tests are always safe. If the schedulability tests used during the validation generate some pessimisms, the system can be over-dimensioned (i.e. over-sized). In other words, to overcome the uncertainties caused by validation techniques, real-time system designers may replace the processor by faster ones or add new cores to comply with the timing constraints of the real-time application.
Feasibility/Schedulability tests produce results that are of binary nature (i.e. schedulable or not). In the design phase, it is interesting to measure the impact of the system parameters on its schedulability. Increasing the system cost is not acceptable in the industry. So, the objective is to address the dimensioning problem as early as possible in the development cycle to reduce design and prototyping costs.
The dimensioning analysis tests have the same characteristics as the schedulability tests. Every dimensioning test is related to a specific context. Essentially, the contexts which need dimensioning tests are known to be incomplete: some timing parameters may be unknown and/or the scheduling algorithm is not defined and/or tasks are not yet allocated and/or the hardware architecture is not well defined, etc. As the dimensioning test is related to a specific context, then it may be characterized by several properties related to the feasibility, the complexity and the sustainability.

III.3.a. Examples of dimensioning methods

Hardware dimensioning The objective of the hardware dimensioning of a real-time system is to set the resources (processor number and processing capacities) required for a task-set in order to respect the timing constraints. A rudimentary way to perform this kind of analysis is to investigate the resource levels within a given interval by using successive dichotomies. Minimizing the number of processors [DRGR08] is an example of hardware dimensioning.

Sensitivity analysis The sensitivity analysis of a real-time system aims to study the variation domains of the system parameters while maintaining the system schedulability. These domains are called the feasibility regions. Several research studies have discussed the sensitivity analysis like [BDNB08, Ves94].

Priority assignment and task allocation In order to enforce the schedulability of the system, the assignment of the priorities to the tasks and the messages is very important. Choosing the priorities for a simple system can be a simple job not requiring any computation tool. Nonetheless, defining the set of priorities for complex systems (like automotive systems) becomes very difficult. Indeed, any bad choice may lead to an un-schedulable system, hence it enforces the designers to over-dimension components. The Audsley's algorithm [Aud91] is one of the research studies related to the priority assignment. When the hardware structure contains more than one processor, we talk also about the task allocation. Then, several methods have been proposed to simultaneously allocate tasks to processors and assign fixed priorities to tasks and messages [RRC03].

IV. Conclusion

The main goal of designing a real-time application before its implementation on a target machine is to validate both logical and temporal behaviors. Hence, to study the schedulability and the feasibility of a real-time system, it is necessary to focus on the modeling process to integrate the different timing characteristics for analysis purposes. In order to master the system complexity and to improve the system-level quality, model-driven engineering is gaining momentum in industrial domains. The challenging rubs are how to integrate real-time architecture models and how to perform different kinds of scheduling analyses based on formal and mathematical aspects. Therefore, the big challenge is to study the possibility to get correct-by-construction products due to an incremental design process. Next Chapter is devoted to present works related to model-based engineering.
In the sequel, we use the term "scheduling analysis" to regroup different analysis categories tackled in this chapter.

Chapter 4

Technological Background

Contents

Abstract

Scheduling tests often require different information of the analyzed system as input, and models provide a good means of capturing this information in a structured way. In this chapter, we present some capabilities related to the use of model driven engineering, and their impact on the design of real-time systems. Furthermore, some standard modeling languages and analysis tools will be presented, discussed and compared to each other.

I. Introduction

Schedulability/feasibility methods need several data about the systems that require analysis. Model-driven engineering becomes increasingly used to propose solutions and tools for modeling and analyzing real-time systems. The remainder of this chapter is organized as follows. Section II is devoted to present principles of model-driven engineering. Section III presents some works based on model-driven engineering and related to the design and the analysis of real-time systems. The last section concludes this chapter.

II. Basics of Model-Driven Engineering

Abstraction representations have been required due to the growth of the systems complexity. Indeed, the system engineering has evolved around the notion of model. An engineering model is a selective representation of some system that specifies, accurately and concisely, all of its essential properties of interest for a given set of concerns. Nowadays, multifarious disciplines are based on computer models in order to use equipped tools and automatize their development activities. Then, computer models have increasingly taken a leading role in the development cycle of complex systems thanks to model-driven engineering (MDE).

II.1. Models and Meta-models

Model-Driven Engineering (MDE) has a significant influence on the systems development by focusing on the abstract concern more than the traditional programming concern.

Definition 10. *(Model) A model is an abstraction and a systems's simplification enabling to understand and provide answers related to the modeled system. Then, a system can be described by different models related to each other.*

In MDE context, the roles of models are numerous as they are used during the whole development cycle. So, models can be used to describe the requirements, to express the design choices independently of domains requirements, and to explore several solutions based on several criteria. Yet, models may also be used to store information. In other words, model-driven engineering is a form of generative engineering in which one or several parts of computer application are generated from the models.
To highlight the nature of the different models used in computer systems, we will identify two main relationships. The first one, called "represented by", indicates a representation of an object which is modeled through a model. The second relationship, called "conforms to", defines the dependence of a model to a modeling language (see Figure 4.1). In model driven engineering, these relationships have a special attention since domain modeling languages are described by the models. These models are called meta-models. A modeling language is a computer language intended for constructing models of systems and the contexts in which these systems operate. Figure 4.1 shows the modeling of some geometric shapes. The proposed model (model shown in model layer) is an engineering model representing the "system" from a particular viewpoint. The model can be specified and composed by using the set of components and relations provided by the metamodel (model shown in meta-model layer). So, every model element conforms to a metamodel element. Moreover the modeling viewpoint also

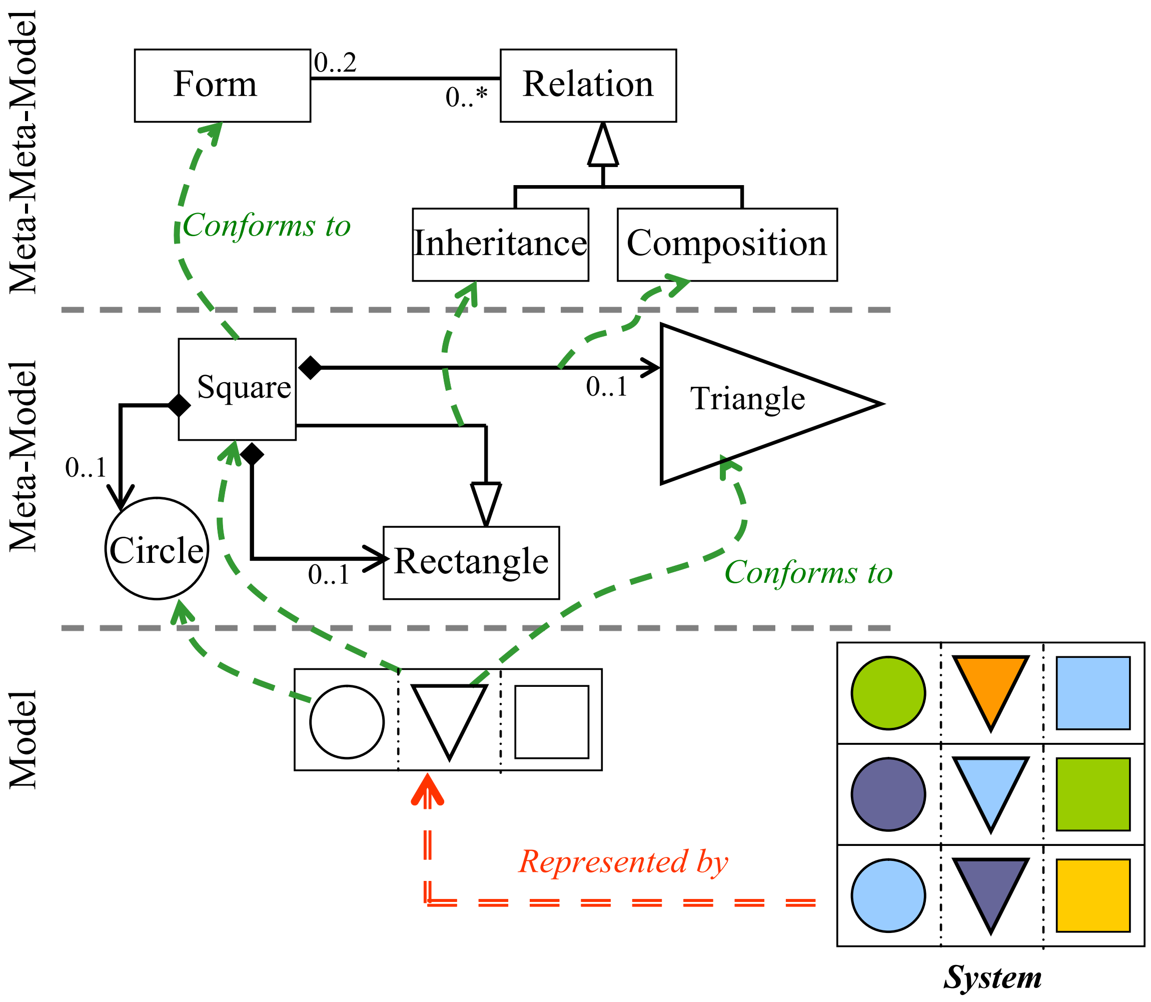


Figure 4.1: Illustration of model driven engineering concepts

depends on the meta-model (e.g. the color characteristics are not taken into consideration by the meta-model). Likewise, the elements composing the metamodel and their arrangements conform to the meta-metamodel (model shown in meta-meta-model layer). Accordingly, it is helpful to provide the following definitions.

Definition 11. *(Metamodel) A meta-model is an abstraction for highlighting properties of the model itself. A meta-model describes the different types of model elements and how they are arranged [GETS07].*

A model conforms to the metamodel as a program conforms to the programming language's grammar in which it is written.

Definition 12. *(Meta-metamodel) A meta-metamodel is a model of a modeling language for describing meta-models.*

In other words, the meta-metamodel contains adequate elements to define modeling languages and it has also the ability to describe itself.

II.2. Model Driven Architecture

The concept of MDE emerged as a generalization of the Model Driven Architecture (MDA) which is a trade mark proposed by the Object Management Group (OMG) in 2000. The MDA approach is based on a set of OMG standards including MOF [OMG11a], UML [OMG04], XMI [OMG11b], OCL [OMG06], etc. Where:

- The MetaObject Facility (MOF) defines an abstract and extensible language to describe, define and manipulate meta-models. It has the characteristic of auto-definition.
- The Unified Modeling Language (UML) is a modeling language for general purposes. Historically, UML is an object oriented language. It includes a set of graphical notations representing different views of a system.
- The XML[1] Metadata Interchange (XMI) complements the modeling languages by defining an interchange format based on XML. Interoperability and serialization techniques of models are based on the XMI.
- The Object Constraint Language (OCL) is an expression language enabling to precise the specification which may be ambiguous due to the graphical notation of modeling languages.

II.3. Meta-Modeling

The meta-modeling is the process of defining the metamodel of a modeling language. The meta-modeling aims to model the language, which enables to express the designed system. A modeling language is any artificial language which may be used to express information of systems in a structure. Modeling languages can be graphical or textual. Graphical modeling languages use diagram techniques with (i) named forms which represent concepts, and (ii) links connecting the forms to express the relationships. Textual modeling languages typically use standardized grammars to make computer-interpretable expressions. Examples of a graphical modeling language and its corresponding textual modeling language are UML [OMG04] and XMI [OMG11b]. There are several ways related to the meta-modeling process permitting to obtain a modeling language. We solely highlight two manners related to our thesis topic.
1) On the one hand, UML was proposed as a universal modeling language. It is one of the main standards on which the MDA approach is based. Nonetheless, UML is a language dedicated to the software systems based on the object paradigm. Yet, UML does not design the non-functional information. So, to overcome the fore-mentioned limits, the UML profile mechanism has been proposed to extend the language with new concepts. Recently, several MDA standards have been developed following the UML profile mechanism.
2) On the other hand, MDE favors the construction of modeling languages dedicated to a particular domain (i.e. Domain Specific Modeling Language - DSML). The last mechanism provides a flexible way to its users in order to conceive their needs and requirements.

[1]eXtensible Markup Language

II.3.a. Domain Specific Modeling Languages (DSMLs)

A Domain Specific Modeling Language (DSML) is a specification language that provides - through appropriate notations and abstractions - expressive power focused on - and usually restricted to - a particular problem domain. In other words, a DSML is used to describe a problem and its solution in concepts which are familiar to people who work in the domain.
Defining the structure of a DSML consists of the design of an abstract syntax, concrete syntax and semantics.

- Abstract syntax describes the basic structure of the language. Indeed, the abstract syntax is based on a set of concepts and constructs that can be exchanged and specifies how they can be linked together to form valid expressions. The abstract syntax is also a set of rules for creation of well-formed underlying structures of the language.

- Concrete syntax means defining graphical and/or textual representation(s) dedicated to end-users in order to manipulate models (instances of the DSML). For instance, the model of Figure 4.1 depicts some shapes (model) representing a part of the graphical representation of the DSML (the DSML abstract syntax corresponds to its metamodel).

- While syntax is concerned with the form of a valid model, we need to describe the meaning in terms of some well-known semantic domain. There are two kinds of DSML semantics [AvdBEV12]. The static semantics define the structural properties of models that can be determined without considering either input or execution. The dynamic semantics (also know as execution semantics) are concerned with the execution and the behavior of specified models. Moreover, semantics may be defined by a natural language or by describing syntactical elements in terms of a formal approach using, for example, a mathematical framework, logical rules, execution rules on an abstract machine [CMTG07], etc.

Sometimes, it is preferable to construct a DSML by reusing and specializing parts of existing metamodels. This way alleviates the development process of new modeling languages from scratch. Generally, there are two major ways to extend modeling languages [Sel07]:

- The heavyweight approach is a extension based on the refinement of new language constructs from an existing modeling language. Then, designers can customize the source language as required via the extension, refinement and modification mechanisms in order to create a new modeling language.

- The lightweight approach is a restricted metamodel extension of an existing modeling language without modifying its abstract syntax or its semantics.

When the lightweight mechanism is used to extend MOF-based languages (see Section II.3.b), that leads to build profiles [FFVM04]. Indeed, profiles are usually used to customize UML (which is a MOF-based language) for a specific domain. Then, UML profiles are also considered as domain specific modeling languages.

UML profile UML can be easily customized by using the lightweight mechanism provided by UML itself. The recent version of UML (including the Profiles package) defines a set of UML artifacts that allows a specification dealing with concepts and notation required in particular application domains (e.g., real-time systems).

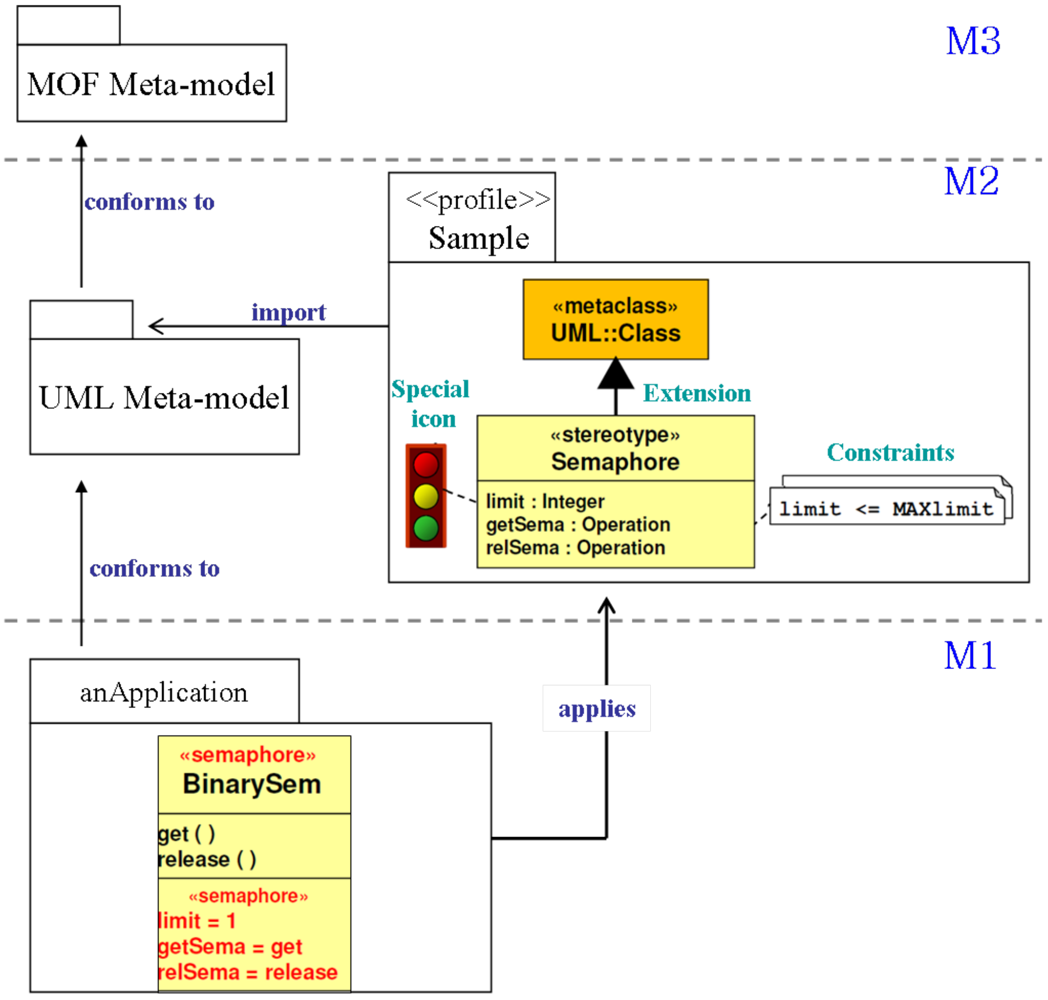


Figure 4.2: The profile mechanism: example of an UML profile

Figure 4.2 represents an example of an UML profile. This latter is dedicated to design semaphores. UML profiles are based on three aspects:

- Stereotype: a stereotype extends a metaclass (i.e. class of UML metamodel) in order to enrich the semantics without changing the original meaning. For example, Figure 4.2 shows an extension of the UML metaclass "class". The concrete syntax of the stereotype respects that of the metaclass (i.e. the class shape), but the stereotype can also have an optional one (e.g. icon). Moreover, for semantical reasons, the stereotype may also have properties called "tagged values" and be subjected to many constraints.

- Tagged value: it is a way for adding attributes (e.g. concurrency limit in Figure 4.2) and operations (e.g. getSemaphore() in Figure 4.2).
- Constraint: adding constraints allows to precise semantics. Generally, a constraint is expressed by an OCL rule. For instance, according to the rule shown in Figure 4.2, the constraint means that the maximum number of concurrent accesses cannot exceed a special constant `MAXlimit` which equals to 1 in case of a binary semaphore (i.e. mutex).

Referring to the growth of the system's complexity, DSMLs qualities become undeniable. However, as they are used by small groups of specialists, it is economically difficult to develop and maintain the DSML appropriate tools (e.g. Integrated development environments, code generation tools, etc.). Fortunately, various platforms have been created ensuring a good DSMLs maintainability. Among these initiatives, we can find the Eclipse Modeling Project (including EMF, GEF and GMF) [Ecl] which is a popular integrated development environment.

II.3.b. Towards a meta-modeling with MOF

The notion of the metamodel has become widely used as a description language in different domains. Then, many meta-models have emerged to provide their characteristics in particular domain (web services, data bases, software development, etc.). To face the incompatible emergence of metamodels, the Object Management Group (OMG) has proposed a generic framework supporting different metamodels. Since the objective is to curb the number of abstraction levels, the solution was to provide a common language defining all metamodels. Accordingly, the OMG has provided the MOF as metametamodel. This latter is the cornerstone of MDA. Therefore, the MDA approach depicts the four-layered architecture (see Figure 4.3):

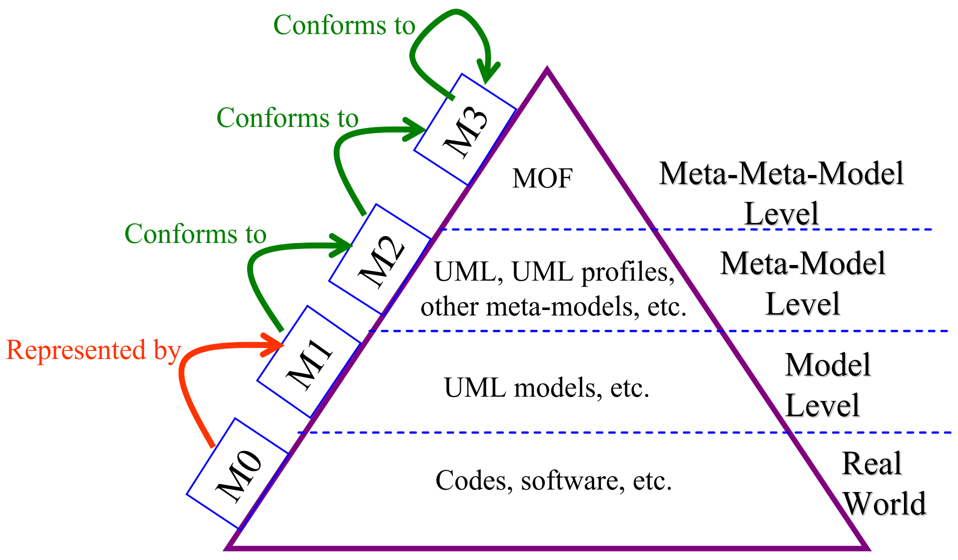


Figure 4.3: Four-layered architecture of MDA [BB02]

- M3. The meta-metamodel layer contains MOF language. The MOF is used to define the various modeling formalisms and also to describe the MOF itself.
- M2. The metamodel layer contains metamodels built by the M3-model. The most prominent example is the UML metamodel [OMG04]. So, all metamodels, standardized or not, defined by MOF are positioned on the M2 level.
- M1. The model layer contains the structure of elements that conform to a model of level M2, for example, models written in UML.
- M0. The real-world layer contains concrete objects represented by models of level M1, for example, executable codes corresponding to UML models.

II.4. Model Transformation

The model transformation is among the main advantages of modeling layers (i.e. models, metamodels and meta-metamodels). In addition to the meta-modeling, the model transformation is a central operation in model driven engineering allowing to get different target models by treating different source models [REP12].

Definition 13. *(Model Transformation) A model transformation is a function, ϕ: $S \rightarrow T$, such that: ϕ takes as input a set of source models S, and produces as output a set of target models T. S and T are models sets conforming to two sets of metamodels. If the two metamodels sets are identical, then the model transformation ϕ is called endogenous, otherwise it is called exogenous.*

Figure 4.4 shows an overview of the model transformation mechanism as it is used via model driven engineering. In general, a model transformation is a program that is composed of a set of rules and based on a corresponding metamodel. This latter represents an abstract definition of the used transformation language. A transformation rule is a description of how one or more elements in the source language can be transformed to one or several elements in the target language [KWB03]. Then, the transformation rules are specific to the source and target metamodels. The execution of the transformation rules is ensured by a transformation engine. Furthermore, the rules description can be declarative (i.e. the execution order of rules is not defined by users) or imperative (i.e. the execution order of rules is defined by users).
There are many classifications criteria related to model transformation [CH06, REP12]. In the current work, we are interested in two kinds of model transformation approaches:

- model-to-model transformation produces models conforming to the target metamodels. For example, ATL (Atlas Transformation Language) [JK05] is one of the famous transformation languages. It contains a mixture of declarative and imperative constructs. Also, QVT (Query-View-Transformation) [GGKH03] is the model-to-model transformation language provided by OMG. Figure 4.5 shows an example of the model-to model transformation ensured by an ATL program called “transfo.atl”. This program transforms any instance of the `Person` class (part of the metamodel `MMA`) with a `function` property equals to “Manager” to an instance of the `Contact` class (part of the metamodel `MMB`).

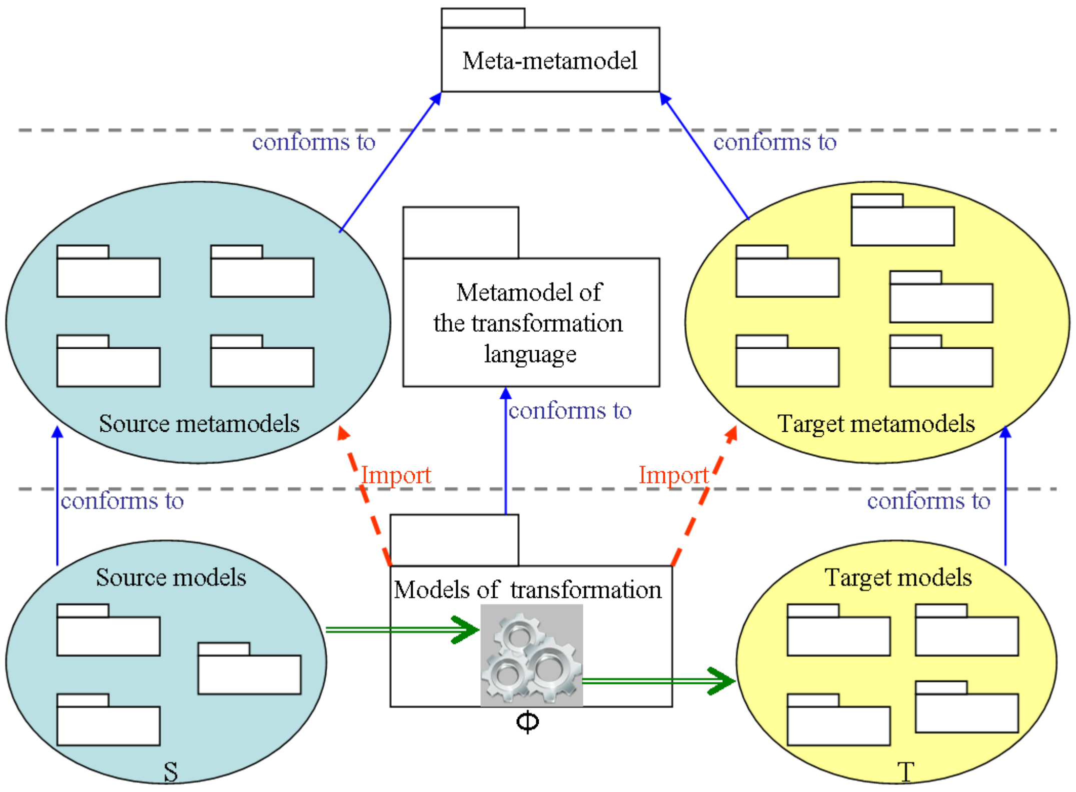
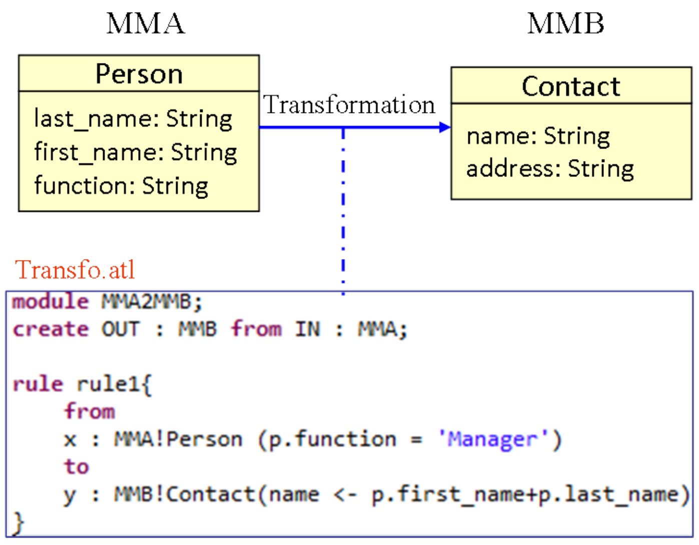


Figure 4.4: Models transformations in model driven engineering



```
module MMA2MMB;
create OUT : MMB from IN : MMA;

rule rule1{
    from
    x : MMA!Person (p.function = 'Manager')
    to
    y : MMB!Contact(name <- p.first_name+p.last_name)
}
```

Figure 4.5: Example of a transformation program with ATL language

- model-to-text transformation permits to obtain target models which essentially consist of strings. The model-to-text transformation is usually used to perform the code generation, like transforming UML models to a Java programs. Acceleo [MJL06] and Xpand[Kla07] are among template-based languages ensuring model-to-text transformation.

II.5. Model-driven development of real-time systems

Models occupy a prominent place among the artifacts of systems development. When applied to the embedded systems domain, the main idea is to use models to highlight functional and non-functional aspects of the system (behavior, structure, timing, etc.) prior to any implementation. Thus, models must be sufficiently precise in order to be understood and processed by machines. During last decade, several research studies have discussed the integration of the model-driven development into real-time systems design by taking advantage of the expressive power of modeling languages. That allows to explore different candidate architectures according to the non-functional requirements. The next section is devoted to discuss some model-based approaches of real-time systems design.

III. Model-based Approaches and Scheduling Analysis

The objective behind proposing model-based approaches of real-time systems is to get a design in a language that is commonly understood by different actors (e.g. modelers, architects, analysts, etc.). Then, the common language represents a manner that designers use to discuss many candidate design solutions. Moreover, it is necessary for the solution to be unambiguous in order to perform the analyses which are the scheduling analyses in our context (as it is mentioned in Chapter 3).

III.1. Standard design languages

Several design languages have been proposed, however this section is solely devoted to present some standard ones.

III.1.a. Architecture Analysis and Design Language (AADL)

Description The Architecture Analysis and Design Language (AADL) is a domain specific language standardized by the Society of Automotive Engineers (SAE) in 2004. The first version of AADL was developed for avionics field, it was formerly known as the Avionics Architecture Description Language. In 2009, the SAE published the second version of AADL which is used to model the software and the hardware architecture of embedded real-time systems. AADL is aimed at modeling the architecture of distributed real-time embedded systems, cyber-physical systems, and other mission-critical software-reliant systems. The description of a system in AADL consists in describing an architecture as a hierarchy of components with their interfaces and their interconnections.
There are three component categories: software, hardware and system (see Figure 4.6). Interactions between components are expressed through predefined features (like ports, access to bus, data, etc.). Furthermore, the resulting architectural models can be used for various activities such as the analyses (e.g. schedulability analysis, flow control) or the code generation.

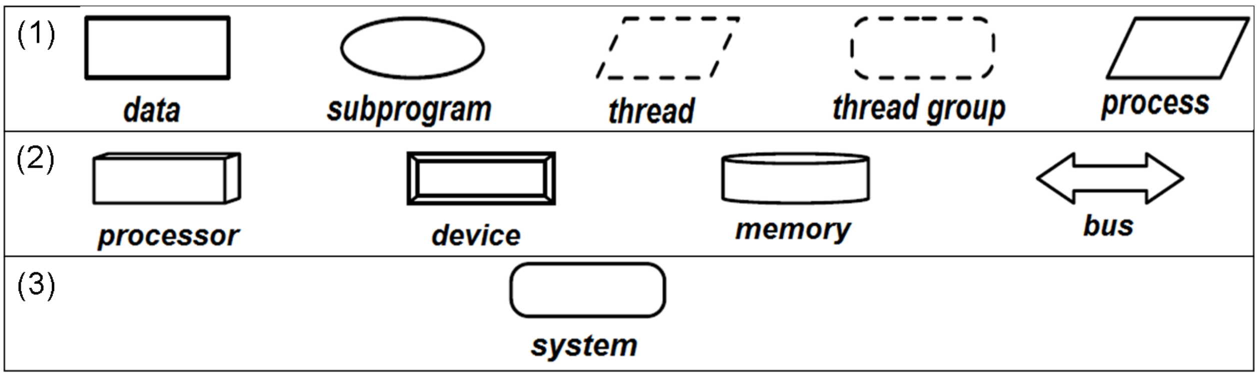


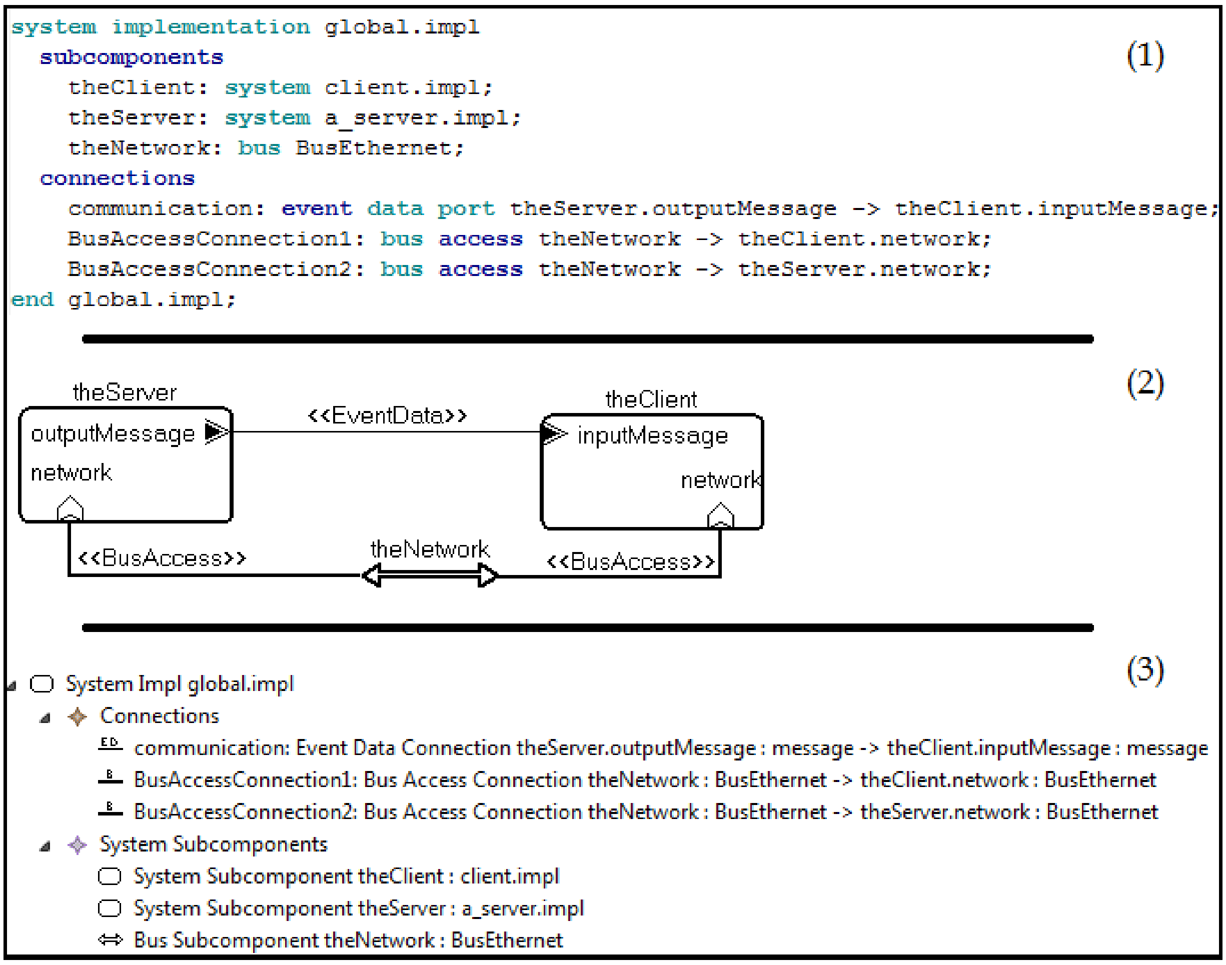
Figure 4.6: The main AADL components. (1): part of software components, (2): part of hardware components, (3): system component

Utilization OSATE [Uni13] is one of the tools supporting AADL. It is an open-source Eclipse plug-in and it suggests three possible formalisms to construct AADL models. The graphical formalism, the textual formalism, and XML formalism called AAXL. Figure 4.7 shows different possible formalisms of an AADL model where the designed system contains two sub-systems communicating through a network.
AADL (version 2) provides two meta-models. A metamodel of the AADL models and property sets, and a meta-model of AADL model instances. Both kinds of meta-models conform to the Meta Object Facility (MOF) (see Figure 4.8). In other words, if one would like to model a real-time system using AADL, one has first to do a model conforming to the meta-model of the AADL models. Then, this model has to be instantiated to get the model instance which conforms to the meta-model of the AADL model instances. So, the obtained model instance represents the practical system.
AADL is extensible by the use of annexes or property-sets without modifying the meta-models. There are among others annexes available for specifying error handling and behavior. AADL provides also the ability to specify non-functional properties of the components separately.
Then, designers can instantiate the corresponding metamodel to specify some properties as needed. Listing 4.1 show an example of a property-set containing the declaration of *Period* as a property related to *thread* elements.

Listing 4.1: Part of a property set

```
property set My_Properties is
Time_Units: type units (Us, Ms => Us * 1000, Sec => Ms * 1000, Min => Sec * 60, Hr => Min * 60);

Time: type aadlinteger 0 Us .. value(Max_Time) units Time_Units;

Period: inherit Time applies to (thread);
        .
        .
        .
end My_Properties;
```

For various purposes, an AADL model may be completed by adding properties. The utilization of an ad-hoc property is done as shown in Listing 4.2, where the model is enriched by the *Period* property



(1)

```
system implementation global.impl
  subcomponents
    theClient: system client.impl;
    theServer: system a_server.impl;
    theNetwork: bus BusEthernet;
  connections
    communication: event data port theServer.outputMessage -> theClient.inputMessage;
    BusAccessConnection1: bus access theNetwork -> theClient.network;
    BusAccessConnection2: bus access theNetwork -> theServer.network;
end global.impl;
```

(2)

(3)

```
System Impl global.impl
  Connections
    communication: Event Data Connection theServer.outputMessage : message -> theClient.inputMessage : message
    BusAccessConnection1: Bus Access Connection theNetwork : BusEthernet -> theClient.network : BusEthernet
    BusAccessConnection2: Bus Access Connection theNetwork : BusEthernet -> theServer.network : BusEthernet
  System Subcomponents
    System Subcomponent theClient : client.impl
    System Subcomponent theServer : a_server.impl
    Bus Subcomponent theNetwork : BusEthernet
```

Figure 4.7: Example of an AADL model which contains two sub-systems communicating through a network. (1): textual formalism, (2): graphical formalism, (3): AAXL formalism

already declared in Listing 4.1. In addition, every component specified in the architecture inherits all the properties from its parents except if the property is defined in the component itself.

Listing 4.2: Example of a property-set utilization by an AADL model

```
. . .
thread implementation threadExample.impl
  properties
        . . .
    My_Properties::Period => 15 Ms;
                . . .
end threadExample.impl;
. . .
```

While AADL is proposed as a standard, the SAE provides a predefined set of properties.

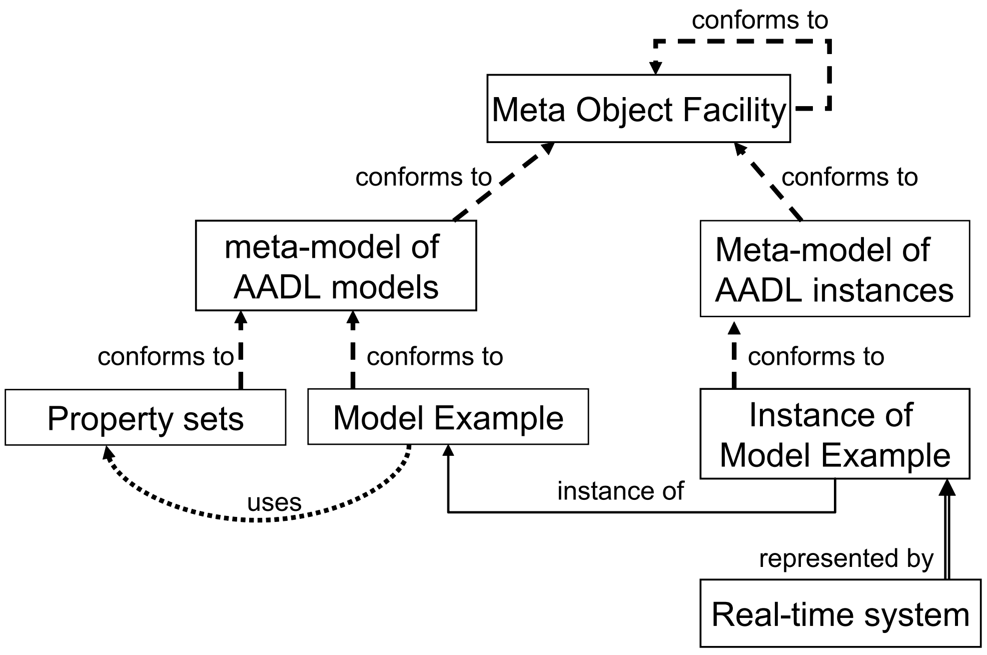


Figure 4.8: AADL metamodels

Advantages/Drawbacks The main advantage of AADL is that it is an industrial standard. Moreover, AADL proposes a practical way to be extended by using annexes and property-sets. However, the extension and the customization of AADL - to represent for example some timing based properties - may lead to a lack of semantics related to the schedulability analysis. Indeed, as such extension is ad-hoc, its use in a design can make the design be driven by the tools supporting these properties, and consequently that may influence the design of the system architecture.

The different components provided by AADL allows designers to get an architecture very close to the practical one. Moreover, designers using AADL through OSATE can benefit from the syntactical and architectural verification. Nonetheless, while the component-based modeling needs to have deep information about the designed system, its utilization is not very suitable for early design stages.

III.1.b. Modeling and Analysis of Real-Time and Embedded systems (MARTE)

Description In 2009, the OMG has adopted the Modeling and Analysis of Real-Time and Embedded systems language (MARTE) [OMG09], which is an UML profile providing support for specification, design, and validation stages. MARTE is a standard structured around two main concerns: (i) modeling the features of real-time and embedded systems and (ii) annotating application models in order to support analyses of system properties. Indeed, as an UML profile, MARTE annotates UML models (i.e. functional models) with the real-time properties. For this purpose, MARTE proposes 158 stereotypes to be used to model both hardware and software properties.

MARTE consists of four parts where elements composing each part are regrouped in a package as

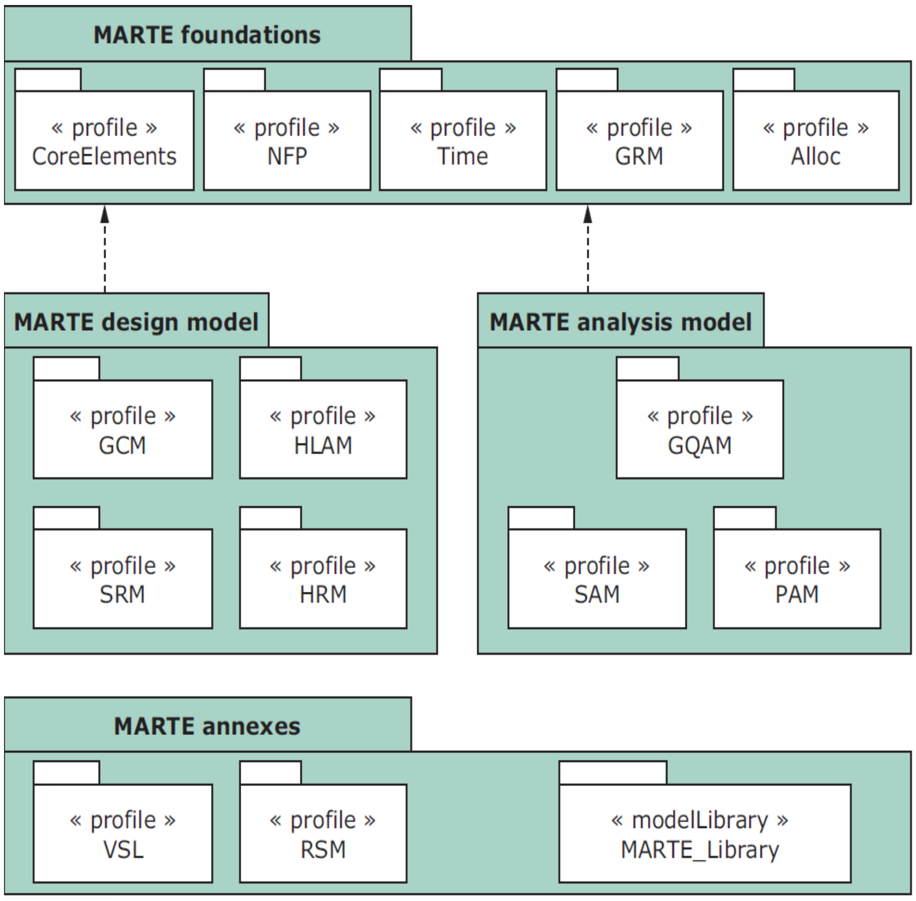


Figure 4.9: Architecture of the MARTE profile [OMG09]

depicted in Figure 4.9[2]:

- Foundations package: it contains the basic concepts required to support real-time and embedded domain.
- Design package: it refines the foundation concepts by supporting detailed design of real-time software and hardware characteristics.
- Analysis package: it provides elements supporting quantitative analysis of UML models. The package is specially aimed at offering information needed for schedulability and performance analyses.

[2]NFPs = Non-Functional Properties, GRM = Generic Resource Modeling, GCM = Generic Component Model, Alloc = Allocation modeling, RTEMoCC = RTE Model of Computation & Communication, SRM = Software Resource Modeling, HRM = Hardware Resource Modeling, GQAM = Generic Quantitative Analysis Modeling, SAM = Schedulability Analysis Modeling, PAM = Performance Analysis Modeling, VSL = Value Specification Language, RSM = Repetitive Structure Modeling

- Annexes package: it gathers all the MARTE annexes like the Value Specification Language (VSL) profile that defines a language for non-functional properties within UML models [Esp07].

Utilization No modeling process is defined to use the MARTE language, hence the utilization of several elements may be used in different contexts with different semantics. Figure 4.10 shows an example of MARTE utilization, where tasks are annotated by the *schedulableResource* stereotype, processors are annotated by the *saExecHost* stereotype, etc.

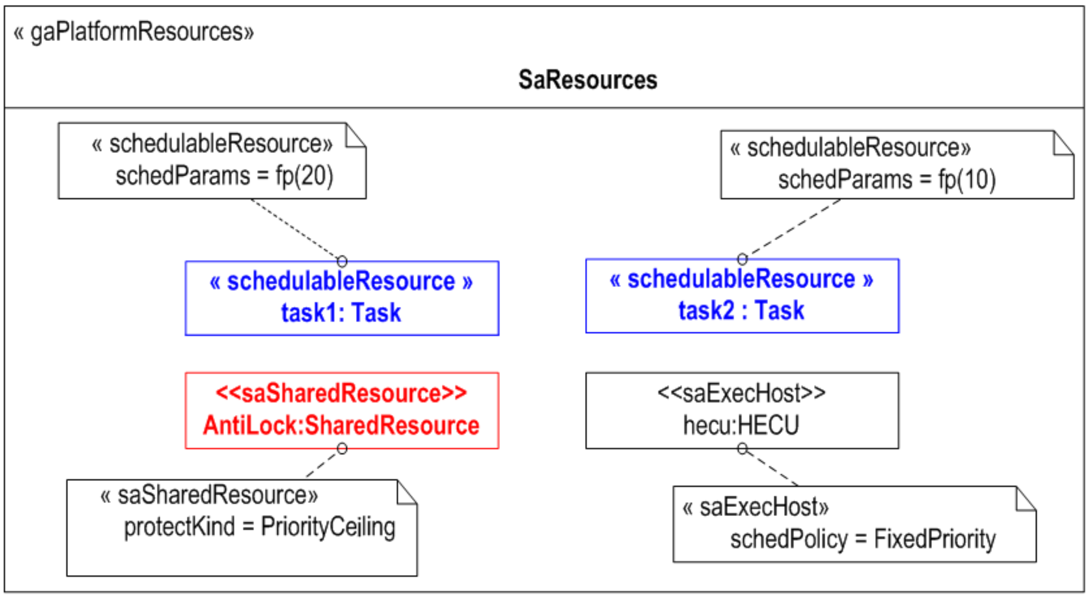

« gaPlatformResources»
SaResources
« schedulableResource» schedParams = fp(20)
« schedulableResource» schedParams = fp(10)
« schedulableResource » task1: Task
« schedulableResource » task2 : Task
<<saSharedResource>> AntiLock:SharedResource
<<saExecHost>> hecu:HECU
« saSharedResource» protectKind = PriorityCeiling
« saExecHost» schedPolicy = FixedPriority

Figure 4.10: Example of a MARTE model containing resources platform [MTPG11]

To fill the chasm due to the absence of the modeling process, different MARTE methodologies have been proposed [HG10, MC11, MTPG11]. Optimum process is one of the methodologies interested in scheduling analysis [MTPG11]. The underlying idea behind Optimum is to use MARTE as a modeling framework that provides a rich set of concepts for modeling (i) end-to-end flows, (ii) software and hardware resource platform, and (iii) the allocation of application modules to platform resources. Relying on Papyrus [LIS] which is a UML graphical editor, Optimum methodology suggests different kinds of models based only on a subset of MARTE stereotypes:

- Workload model: it represents the UML functional model (as an entry point) annotated by a set of stereotypes corresponding to the semantics defined by Optimum. The workload model illustrates the end-to-end flows and uses the UML activity diagram for this purpose. Once the workload model is specified, Optimum launches an internal process to produce a concurrency model and a deployment model also called architectural models.

- Concurrency model: it represents the architecture of the task-set and the mapping to the functional blocks.

- Deployment model: it represents the architecture of the software and hardware resources, and the tasks allocation.

Merging the three models leads to get the schedulability analysis model which may be analyzed. The process may be iterated many times until finding the architectural models that fit the workload model.

Advantages/Drawbacks As a UML profile, MARTE benefits from the facilities provided by UML, and also from the several mature UML editors. So, MARTE can be used in different design stages, and that leads to an incremental modeling. Yet, the traceability offered by the properties eases the modeling analysis in an incremental way also. Since MARTE focuses on giving concepts for modeling real-time systems without defining any methodology, that complicates its utilization, in particular in cases where designers have to choose suitable annotations for their schedulability analyses. Not only the utilization steps are not well defined but also no architecture verification is provided to check the reliability of the system architecture.
Following a modeling methodology as discussed earlier can make designers "hemmed" to design only architectures that are able to be analyzed by tools supporting this methodology. In our point of view, this manner of usage does not totally match the principle of the standard modeling language.

III.1.c. Comparison

Although UML-MARTE and AADL are standard design languages, each one has its own capabilities. Differences and similarities are summarized as a comparison table (see Table 4.1). The comparison metrics have been chosen regarding to the context of our work and the background we have acquired when handling those standard languages.

III.2. Analysis tools

When using design solutions (i.e. models), the design has to be analyzed to verify that the timing requirements of a real-time system are met. Performing the scheduling analyses manually is tedious and error-prone, hence several academic open-source and commercial tools have been developed. We present some of these tools.

III.2.a. Cheddar

Cheddar is a schedulability analyzer and simulation toolkit written in Ada for real-time systems [SLNM04]. Cheddar can manipulate AADL models under some conditions, such as the model has to not be complex, and it must be enriched by several ad-hoc properties only understood by Cheddar. Basic systems to analyze can be described by the Cheddar design language, because it cannot cover all kinds of the real-time systems. The Cheddar input and output formalism is based on XML. Furthermore, to ease the modeling, Cheddar provides a graphical user interface.
Although Cheddar supports a great number of schedulability tests, there are cases where existing tests do not match the characteristics of a given system. For those cases, Cheddar offers the possibility to define new analysis tests. This task is a very tedious because it requires to well understand the internal architecture of the Cheddar analyzer.

	AADL	**UML-MARTE**
Systems architecture	The component-based approach used in AADL enables to elaborate an architecture very close to the system in practical cases. Nevertheless, architectures modeled via AADL are frozen.	UML-MARTE focuses more on models which may be organized on separate diagrams for an incremental modeling than the architecture.
Utilization phase in the development life-cycle	The modeling with AADL requires to know several information which cannot be necessarily known at early design steps	UML-MARTE may be utilized in different steps composing the design phase. In addition, UML-MARTE models are grouped in two layers: the logical application layer which contains the functional UML model without annotations, and the non-functional layer which is composed from the other kinds of models the workload model, the platform model, the allocation model, etc.
Traceability: the ability to recuperate the analysis information to be added into input models	Currently, no standard property set enables to keep the traceability, but it is possible to add this feature in an ad-hoc way	the traceability is guaranteed by the non-functional properties of UML-MARTE, but it is still complex because it requires to be familiar with the VSL language
Timing information semantics	AADL components have a clear unchangeable semantic regardless on the designed architecture. Moreover, in case of using OSATE, the utilization of different elements is checked	Due to the absence of a standard semantic, some elements have different meaning according to the methodology followed, the type of the designed systems, and the analysis tools which would be used

Table 4.1: Design modeling languages capabilities

III.2.b. MAST

MAST (Modeling and Analysis Suite for Real-Time Applications) [dC] is a model-based schedulability analysis tool suite developed in Ada for real-time applications. MAST has its own metamodels based on XML, an input metamodel to create the models to analyze, and an output metamodel for analysis results. MAST provides a user environment facilitating the creation of models, a results viewer to view the analysis results, and also a graphical interface to choose an analysis test. However, the graphical interfaces are sprinkled which make their usage in a convenient way difficult. In addition of supporting basic real-time systems, MAST also supports distributed architectures. Moreover, the MAST design language enables to design the core of threads by proposing the *Operation* elements. The type of an operation can be simple, composite or enclosing. Elements composing the last version of MAST metamodels are very close to those of SAM which is a package of UML-MARTE profile. Even if MAST is open source, the extension of MAST for adding new tests requires to program new tests from scratch.

III.2.c. ASIIST

ASIIST (Application Specific Input/Output Integration Support Tool) [NPSB09] is a tool that supports system models specified in AADL to perform many types of real-time analyses. ASIIST has been developed as an Eclipse plug-in and works well with OSATE AADL editor. ASIIST helps to recognize Integrated Modular Avionics (IMA) architectures. Then, by performing the schedulability analysis for such applications, ASIIST can detect effects occurred due to I/O traffic existing in the systems. To solve the analysis equations, ASIIST is related to the kernel of Mathematica.

III.2.d. Rt-Druid

RT-Druid [GNS+07] is an analysis tool and a design environment implemented as an Eclipse plug-in with a graphical user interface. Although, RT-Druid is oriented towards automotive applications (It supports the OSEK/VDX standard), it is possible to use it in any other real-time systems domains. Systems designed via the RT-Druid tool are defined by an application part (pieces of application code), and a platform part (the platform resources, hardware and software). Next, RT-Druid captures the mapping of the functional components of the system to the concurrent threads, and provides schedulability and sensitivity analysis tests.

IV. Conclusion

As it is presented above, there are several standard and non-standard design languages and also a variety of analysis tools. However, each language/analysis tool has its advantages and its limitations depending on the nature of the system which needs modeling, and also depending on the kind of users and their experiences. Referring to the context of model driven engineering, the main idea behind the existence of design languages and analysis tools is the integration of the timing verification during a seamless model-based development process. Unfortunately, several issues are related to the lack of a guidance enabling a good integration of such scheduling analyses during the development process of real-time systems.

Part II

Contributions

Chapter 5

Synthesis and Work Orientation

Contents

Abstract

In this chapter, we synthesize the first part of the manuscript and we give an overview of our contributions presented in the second part. First, the work positioning is highlighted by presenting different problem statements and illustrating them via some motivating examples. The second section describes the objectives which we aim to achieve in this thesis. Finally, the third section gives a brief idea about our proposals.

I. Work Positioning

I.1. Introduction

During the model-driven development of real-time systems, the primary task of system designers is to create models with enough information to support timing requirements in the different design stages. The goal is to use *design models* to drive series of scheduling analysis that verify the timing requirements and help to predict the timing behavior of the target system. Thus, to perform these analyses, the design models must be first transformed into *analytical models* that admit mathematical evaluations. The modeling frameworks and design languages are used to get design models, and the analysis tools accept as input the analytical models and evaluate them mathematically to produce results used for successive refinement of design models. However, the transition from the design models to the analytical models is not an obvious task, and it may generate several problems as it is detailed in the next subsection.

I.2. Issues, Problem Statements and Motivating Examples

Different key issues are of concern in this work. Generally, they are related to a common problem, which is the design decision due to the complexity in terms of real-time systems analysis, and also due to the weak integration of this analysis into a design process.
In the following, we present details of every spotted issue.

I.2.a. Issue 1: usage of the analysis tools

Problem Statement The current utilization of model driven engineering through the design phase of real-time systems accelerates the development process. However, integrating modeling frameworks with analysis tools is still difficult. Indeed, due to the large semantic gap between design and analysis representations, some design information must undergo significant simplification or refinement before being fed into the analytical models used as input by analysis tools. So, the transformation process eases greatly the passage from the modeling frameworks to the analysis frameworks despite the different input analysis formalisms. Nonetheless, the example below gives an overview showing that yet the utilization of the actual solutions requires both modeling and analysis skills.

Example Figure 5.1 shows the two main scenarios illustrating the current situation related to design and analysis frameworks. The first scenario (see Scenario 1 of Figure 5.1) represents the successful classic flow of the tool utilization. In this case, the designer has obtained the analysis results after transforming a design model to an analysis tool. Nevertheless, the designer cannot know if the obtained result is correct and accurate, and if it corresponds well to the analyzed system.
To clarify Scenario 1, we consider an application composed of four periodic independent tasks where each task is defined by a set of properties (See Table 5.1). The preemptive task-set is executed on a uniprocessor architecture and follows a fixed priority scheduling policy. *Task1* is the highest priority task and *Task4* is the lowest priority task. To alleviate the example, we consider that the model is done via a design framework and the transformation is done correctly. Although, after launching the analysis process through two different analysis tools which are Rt-Druid [GNS$^+$07] and MAST

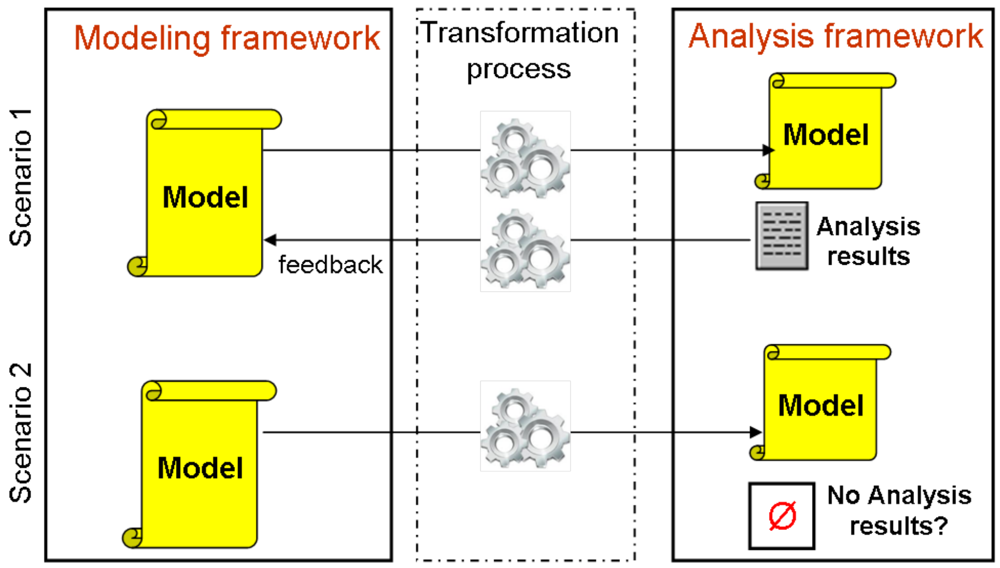


Figure 5.1: From the real-time modeling to the timing analysis: the current situation

Task	**Worst-case execution time**	**Deadline**	**Period**	**Release time**
Task1	3 ms	15 ms	20 ms	2 ms
Task2	4 ms	8 ms	23 ms	0 ms
Task3	5 ms	13 ms	23 ms	5 ms
Task4	9 ms	13 ms	23 ms	7 ms

Table 5.1: Values of task characteristics

[MPHD01], two different results have been produced for the same input model. Table 5.2 shows the response-times provided by Mast and Rt-Druid. While the Rt-Druid result shows that the system is not schedulable because the response-time of *Task4* exceeds the deadline, the Mast result shows that the system is schedulable and provides more accurate response times (i.e. the result provided by Rt-Druid is pessimistic).

Task	**Worst-case response-time** (Mast)	**Worst-case response-time** (Rt-Druid)
Task1	3 ms	3 ms
Task2	7 ms	7 ms
Task3	8 ms	12 ms
Task4	21 ms	33 ms

Table 5.2: Worst-case response times computed through different analysis tools

The difference between the two provided results is apparent. However, it may not be for a designer without deep analysis background, hence the result could be unexpected.

The difference is not related to a wrong implementation of the analysis methods, but to the input analytical model. The example shown in Table 5.1 is considered by Mast as a transaction model [PGH98]. Then, the mathematical formula calculating the response-times takes the release-times into consideration, whereas the analysis chosen via Rt-Druid masks the release-times and considers the model as a sporadic model [JP86]. So, the mathematical formula calculating the response-times considers that tasks are non-concrete (i.e. the release-times are unknown). The result provided by Rt-Druid does not mean that the tool does not support the transaction models, but the analysis functionality chosen via the tool does not support this kind of analysis. In some cases, choosing a very abstract model is not totally a wrong choice. For instance, the analysis functionality chosen via Rt-Druid (i.e. the analysis test) offers the possibility to use a sensitivity analysis (which is a dimensioning technique) technique allowing to tune the system design. This kind of analysis cannot yet be applied to a transaction model.

The second scenario shown on Figure 5.1 happens when the analysis process fails. Then, this failure can be due to the analytical model if it is not supported by the analysis tools (e.g. the Cheddar tool [SLNM04] does not support the transaction model).

The failure could be also related to the transformation of the design models from the modeling tools to the analysis tools, but the correctness of the transformation itself is out of the scope of our work.

I.2.b. Issue 2: design methodologies and semantics gap

Problem Statement In the current standard design languages, one impediment to integrate scheduling analysis is that the underlying design model is quite unlike the analytical models used in real-time analysis theory, which are assumed in the underlying tools. To overcome this semantic gap, several methodologies (like [MTPG11] or [HG10]) propose to use only some concepts focusing on the scheduling analysis to get analytical models, but those concepts do not have necessarily the same semantics inside different methodologies.

The model-driven engineering enables to automatize the transferring process of timing requirements from design languages to analysis tools by using transformation languages like ATL [rg13] or Acceleo [Com] (see Figure 5.2). Then, the current model-based utilization produces tool-driven analytical models. So, instead of allowing designers to have precise models and to help them to perform complex analysis techniques, this way of utilization directs designers to have abstract models dedicated to specific implemented analysis tools.

Example Focusing only on the modeling and analysis areas, we suppose that we have two models expressed in MARTE and representing the same real-time system. The first one respects *methodology 1* and the second one respects *methodology 2* (as it is shown in Figure 5.2). If we were to analyze the first model with the *Analysis Tool 1*, that could be possible after a transformation process. If we maintain the same model and we transform it to another input formalism tool (e.g. *Analysis Tool 2*), that cannot be analyzed, because *Analysis Tool 2* supports the same design language, but supports it according to another methodology. Therefore, we cannot compare analysis results because we did not analyze the same input models. The same scenario may be repeated with another design language/methodology and other analysis tools.

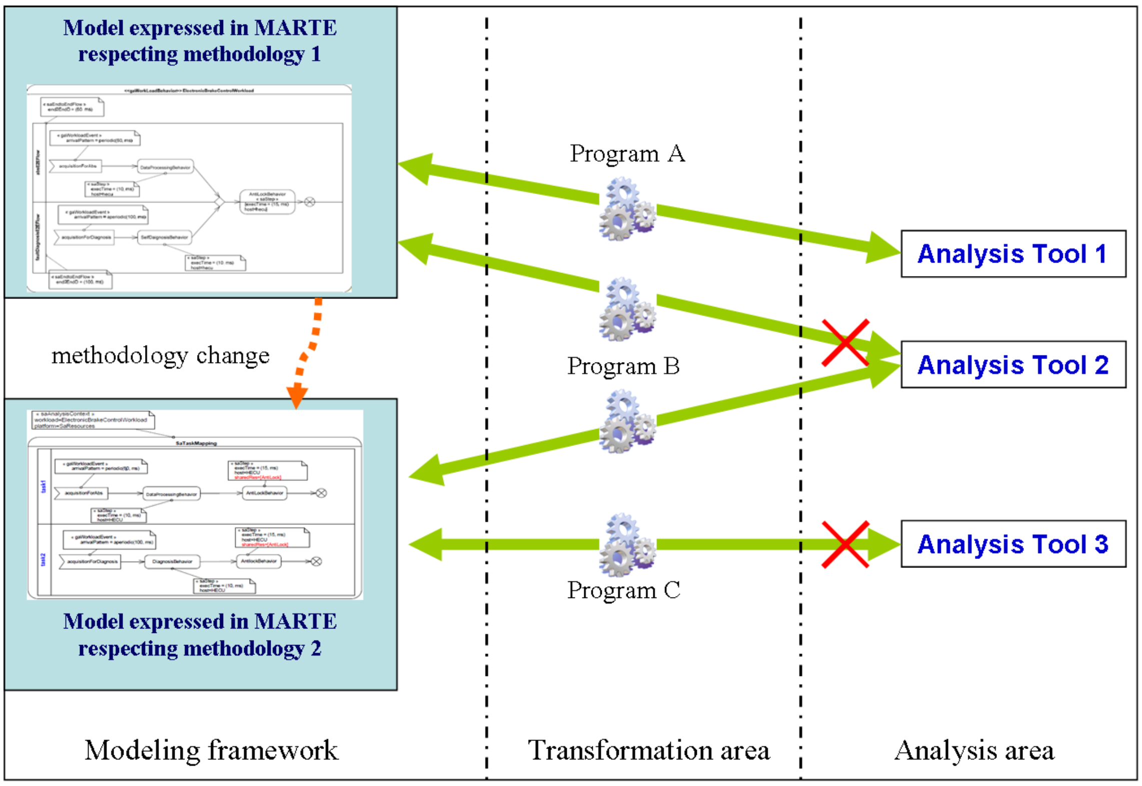


Figure 5.2: Example of methodologies usage with MARTE language

I.2.c. Issue 3: expressiveness and precision-level

Problem Statement During the modeling phase of a hard real-time system, the designers acquire information to specify a design with different entities and functionalities regardless of the resources utilization. Then, the specified design would be expressed in a mathematical model in order to be analyzed. As the mathematical model represents a hard real-time system, it can be characterized by the worst-case scenario of the resources utilization to check the resources availability and the satisfaction of the entities constraints. By following this classical conception progress of a real-time system from the modeling phase to the analysis phase, a pessimism and a resource over-dimensioning could be generated due to the estrangement between the abstract model and the practical application. So, the system modeling has to provide accurate modeling concepts, which are able to provide very clear abstract models without any misunderstanding concerning the system's timing behavior. However,the problem is that usually the more expressive the model is, the more complex is the schedulability test. Then, the time complexity of the analysis is a very important point that designers need to take into consideration.

Figure 5.3 shows an example of two models ($Model_1$ and $Model_2$) related to the same system. The analysis results of $Model_2$ are closer to the practical system than the analysis results of $Model_1$. This difference is related to the expressiveness of $Model_2$, which is more expressive than $Model_1$. However, to analyze a model such as $Model_2$ requires the choice of suitable tests. This task is not trivial for

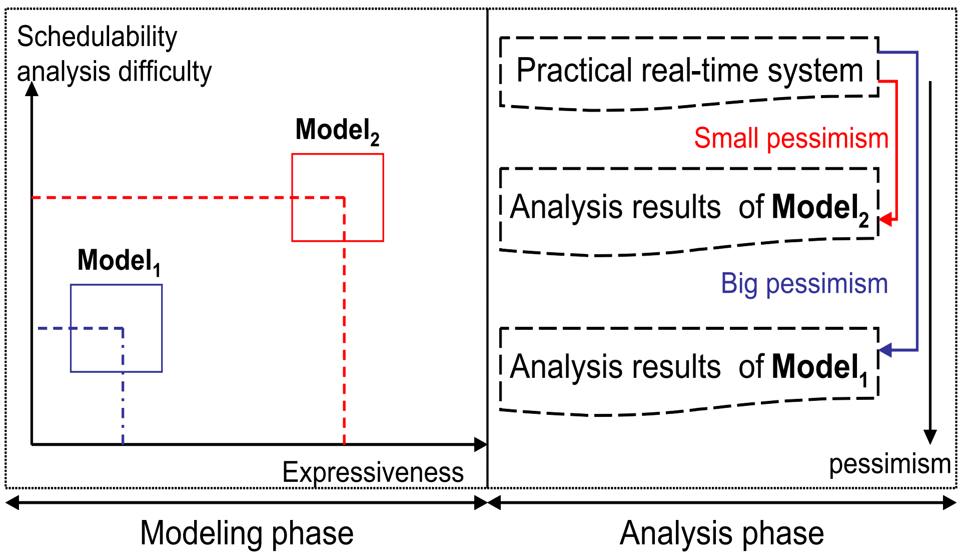

Schedulability analysis difficulty
Model$_2$
Model$_1$
Expressiveness
Practical real-time system
Small pessimism
Analysis results of Model$_2$
Big pessimism
Analysis results of Model$_1$
pessimism
Modeling phase
Analysis phase

Figure 5.3: Design part of real-time system's life-cycle

designers.

Example 1 Testing the schedulability of periodic independent tasks, with implicit deadlines and using a deadline-driven scheduler on a single processor is a very easy problem, and requires a linear time in the number of tasks. A small change in the operational context has a big impact on the analysis. For example, with the same hypothesis, if we consider a fixed-priority scheduling policy, in the case where the periodic tasks are either non strictly periodic, or strictly periodic and released simultaneously, then, the feasibility problem is NP-hard in the weak sense (pseudo-polynomial feasibility test). However, if the tasks are strictly periodic and not released simultaneously, hence the feasibility problem is Co-NP-hard in the strong sense, while one could, for test efficiency reasons, choose to use a pseudo-polynomial feasibility test as a sufficient (but non-necessary) test for this problem.

Example 2 In the following, we illustrate a simple example highlighting that even for expert designers/analysts, it is not always trivial to know the appropriate scheduling context of the analytical model, therefore, to choose the appropriate schedulability analysis tests. Figure 5.4 shows three instances of analytical models (Parts 1, 2 and 3) which seem very similar. Nonetheless, these representations lead to different analytical models in the real-time scheduling theory. Moreover, their schedulability analysis tests are also different.

Part 1 of Figure 5.4 represents an instance of the transaction model (model with offsets) [TC94] where Δt and Δt' are the time offset values of task τ_j and task τ_k respectively in the transaction. The offset values are referring to the external event triggering the treatments. Therefore the three tasks (τ_i, τ_j and τ_k) define a transaction T which is triggered by an external event and Δt" is its period. While each task is characterized by a worst-case execution time and an offset time, tests like the demand bound function [RGRR12] or Turja-Nolin approximate response time analysis [MTN04] could be applied for

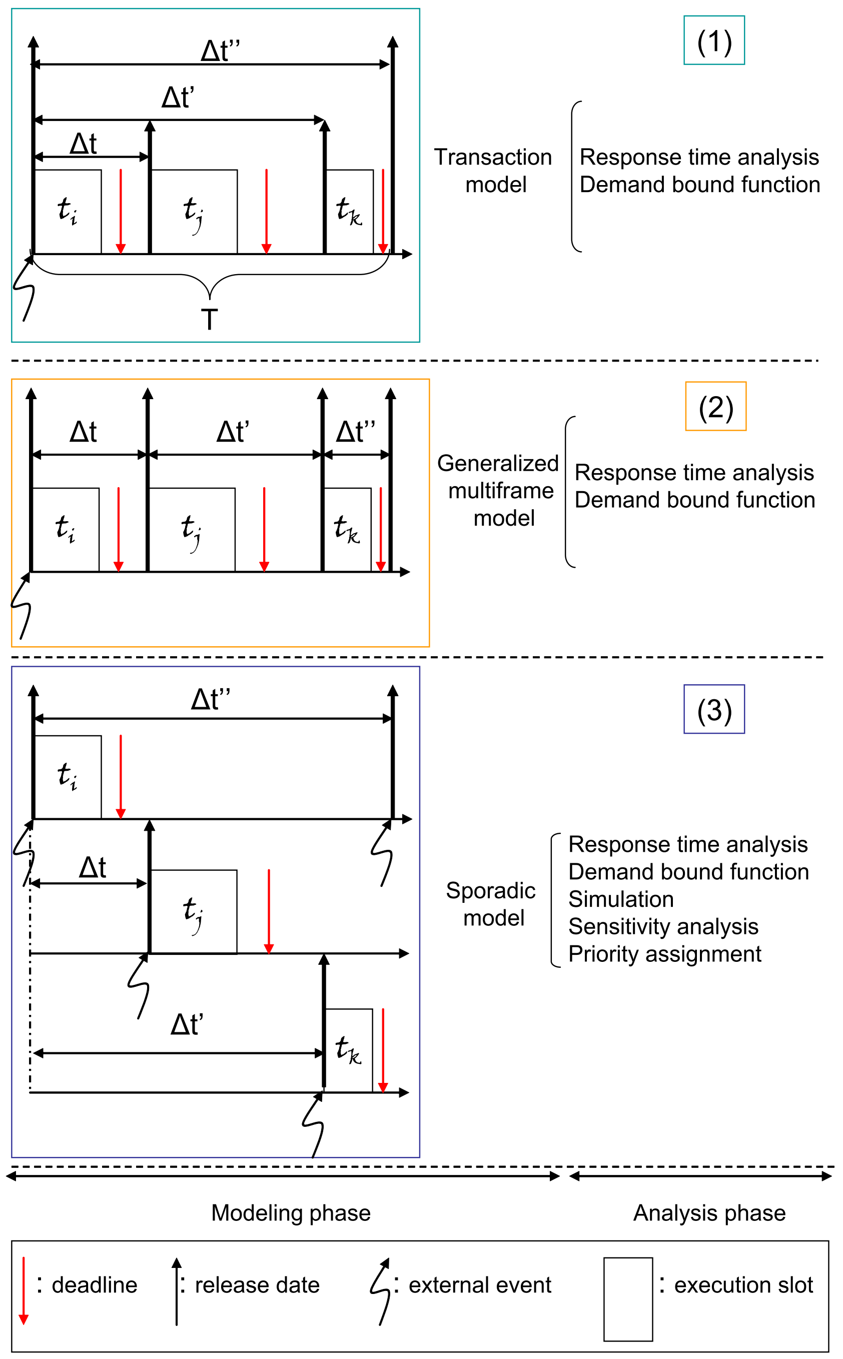


Figure 5.4: Design of a real-time system represented by different kinds of analytical models

the schedulability analysis.

Part 2 of Figure 5.4 represents an instance of the GMF model (generalized multiframe model) [BCGM99].

Therefore, τ_i, τ_j and τ_k represent different frames of the same task instance (job). Δt, Δt' and Δt" represent the minimal durations between every two consecutive frames release times. In case when Δt, Δt' and Δt" are identical, the release time of each frame would represent the deadline of its predecessor, and the priorities of these treatments would be the same, then, the GMF model could be viewed as an instance of the multiframe model [MC96]. For both kinds of model (the GMF and the multiframe models), the response time analysis and the demand bound function could be chosen for analysis.

On the one hand, a non-expert designer may choose to implement the system (previously presented as instances of GMF and transaction models) as an instance of sporadic model (Part 3 of Figure 5.4) due to a lack of scheduling analysis background. Then, the system may contain three sporadic tasks τ_i, τ_j and τ_k which are triggered by external events. Δt" is the minimum inter-arrival time of τ_i. Δt and Δ't are the release times of the events triggering τ_j and τ_k respectively and referring to the release time of τ_i. By considering sporadic tasks, the release times are ignored, and an unrealistic worst-case behavior (critical instant [LL73]) is considered in the schedulability analysis. Hence, these associated tests can provide pessimistic results referring to the real practical system. On the other hand, one can be attracted by the sporadic model in order to benefit from various analysis tests like the sensitivity analysis and the task allocation [ZZZ$^+$12, ZN13].

I.2.d. Issue 4: academia/industry gap

Since the 1970s, the real-time scheduling theory has been devoted to propose different models providing several levels of expressiveness, and different analytical methods with different levels of accuracy and determinism. The utilization of the real-time scheduling theory in practical cases could be profitable. Unfortunately, it is not sufficiently applied and the research results have been exploited in the industry only to a modest extent to date. The chasm between the scheduling theory and the industrial practices may be due to several reasons:

- The scheduling techniques mastery requires a colossal work related to the real-time theory knowledge (i.e. the state of art). However, many systems designers may not be well versed in the real-time scheduling theory. Therefore, the steep learning curve behind many of the current analysis methods has been one of the major impediments to their adoption and their exploitation in the industry.
- Industrial designers are too busy to perform accurate analyses due to cultural and economical reasons, and they prefer to use model-based engineering methods to design and implement their systems. Different working environment reasons have to be taken into consideration (e.g. concurrency/competitiveness, time-to-market, etc.).
- Transferring the knowledge from the research area to an industrial area can be expensive. On the one hand, even if many analysis tools exist, a tool cannot offer all the scheduling models and techniques. As a result, scheduling models and methods can take years before being included in a schedulability analysis tool. This integration requires a lot of efforts, especially if it is implying some changes in the analytical model, and in the way the analytical model can be built from a source language. On the other hand, whereas an academic researcher develops a prototype easing the exploitation of a special research study, adding this prototype in different analysis

environments may require to modify their internal structures. That needs a high development effort for a research group other than the original tool makers.

I.2.e. Issue 5: actors collaboration

The current use of modeling and analysis tools via the model based development process reveals two collaboration types between actors (modelers and analysts). Those types are somehow related to the tools, the utilization domains (automotive domain, avionic domain, etc.), and also to some issues discussed earlier.

- The first type considers that the model designers (also known as modelers) have to be able to analyze the applications by giving them analysis tools to predict the schedulability. This type can be interesting especially in some industrial contexts, where the system designers outnumber the real-time analysts [NPSB09]. However, as experts in both design and analysis of real-time systems are uncommon, the scheduling analysis is driven by the tools capabilities and the designers experience. Furthermore, the analysis tools used have to be completely compatible with a design language (used for modeling), such as using ASIIST analysis tool to analyze models expressed in AADL.

- The second collaboration type consists of the separation of the modelers tasks from analysts tasks. It concerns the case when most designers are under-trained in analysis field and too busy to perform the analysis. Then, each actor focuses on the favorite area. This collaboration type may be important referring to the criticality of hard real-time system, where the lack of the analysis background can lead to imprecise analysis results due to a wrong choice of the analysis tests. Nevertheless, collecting the relevant information from design models to define the analytical models may be laborious. So, this type requires that analysts should have a good modeling skills, or to appeal to the model transformers.

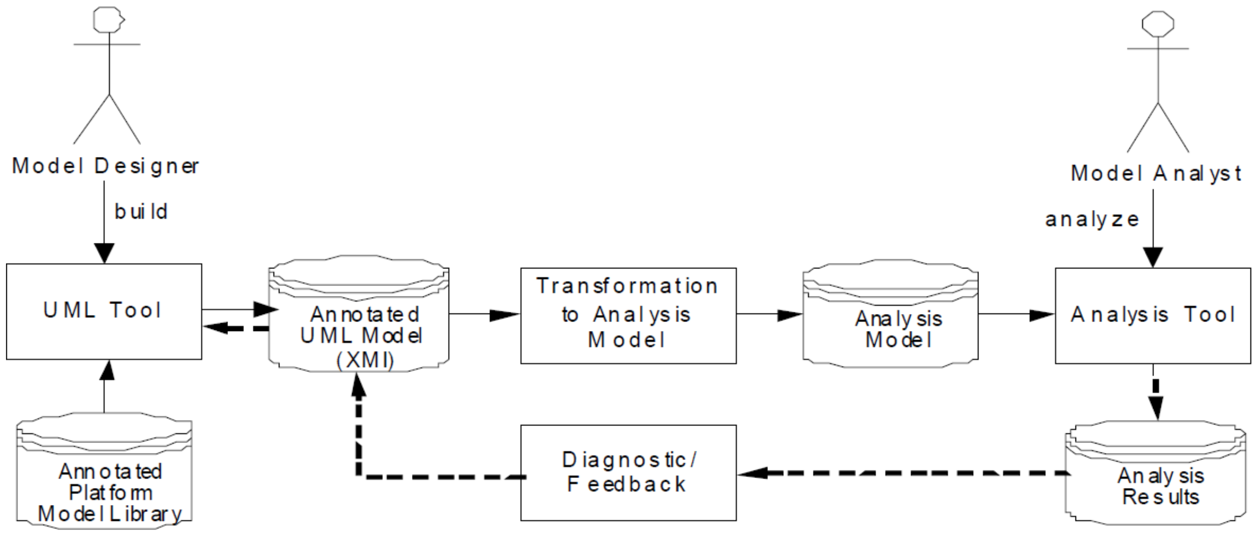


Figure 5.5: A tool chain for carrying out the analysis of a model [OMG09]

Figure 5.5 is extracted from the MARTE specification and depicts an example of the second collaboration type, where the model designer builds models and the analyst is responsible for annotating models and for their analysis.

II. Objectives

The context of this thesis is mainly based on the design phase of real-time systems with its both sides, the modeling and the analysis. Thus, the main objective of our work is to assist real-time designers during the design phase for a seamless development process. Indeed, to achieve this ultimate objective, many points have to be improved and to be taken into account as follows.

II.1. Reducing modeling difficulties

Objective In order to ensure a semantic clarity of the design, the system model should be sufficiently expressive. Thus, the model must be as clear as possible, especially concerning the temporal behavior, and it should offer facilities to use it without any difficulties or misunderstanding for both industrial designers and real-time scheduling analysis researchers.
Then, we aim to assist designers during the modeling by:

- Handling evolutionary models based on clear semantics by using accurate real-time concepts enabling to provide very clear abstract models.
- Detecting if the system is analyzable. In other words, verifying the soundness of the model and checking if it contains sufficient information enabling its analysis, hence the analysis step can start.
- Maintaining the traceability: models must keep the archive of their evolutions oriented by the scheduling analysis. The traceability not only helps during the design phase to know which mistakes have to be fixed, but also during the analysis phase. So, detecting if the system is analyzable may be partially based on the traceability.
- Modeling the operational side (system architecture and timing behavior) of the a real-time system independently of its functional side (also known as applicative or logical part). Then, it is preferable to design the functional part separately, and link it to the operational part to obtain complete models for deep analysis.

II.2. Reducing the analysis difficulties

Objective To avoid design mistakes and to reduce the pessimism when performing the scheduling analysis, the designer should be assisted during analysis phases. Once the system is analyzable, the designers should be oriented to choose the appropriate analysis tests. Furthermore, analysis tests are scattered in different analysis tools. So the tools do not provide every known analysis test, and they do not support every analytical model. Thus, it is also preferable to analyze the system with different tools in order to compare the analysis results. To help the designers to choose the scheduling tests matching their real-time system models, the following questions must be answered:

- What are the possible task allocations?
- What is the worst-case task model?
- What are the efficient tests to validate it?
- Are these solutions exact or pessimistic?
- If the system is not schedulable, what needs to be changed to make it schedulable?

Another way to reduce the analysis difficulties is to increase the applicability of the real-time scheduling theory. Then, it is very needful to have a way to transfer the scheduling theory results from academia to industrial practice, in order to be accessible by users regardless of their experiences. Then, we aim to have a storage to collect real-time research studies. This storage has to offer the possibility to easily integrate scheduling analysis advances.

II.3. Gathering modeling and analysis efforts

Objective Novel methods for system-level analysis are needed not only for computing or estimating timing values (the actual tools are widely efficient), but for providing guidance and support to the designer.
Our idea is to propose a solution playing the role of a communication medium between designers and analysts. Such that:

- The implementation of the first objective (Section II.1) should be independent of design methodologies and generic in order to support every analysis situation.
- The implementation of the second objective (Section II.2) targets to encourage real-time researchers to share their research works (models, tests, tools, etc.), hence:
  - That enables to compare the work with existing ones in terms of the test efficiency, time complexity and the determinism.
  - That eases the integration by different research groups of the prototypes implementing the research works.
  - That facilitates the use of the prototypes (or analysis tools) by designers when their designs correspond to the real-time situations of the proposed works.

III. Overview of our Proposals

As the real-time modeling and real-time analysis are both in constant evolution and improvement in distinct scientific communities, the design methodologies, design languages and analysis tests are constantly improved. Indeed, the methodologies are impacted by the hardware equipments, the software operating systems, and the programing languages. While the model driven engineering offers a relative independence regarding technological changes, and provides a re-usability of the design elements, which are measurable, predictable, and manageable, hence we apply the model driven engineering:

- to unify modeling and analysis efforts;
- to achieve a friendly utilization taking benefits from this variety of standard design languages and timing analysis tools; and
- to increase the applicability of the real-time scheduling theory.

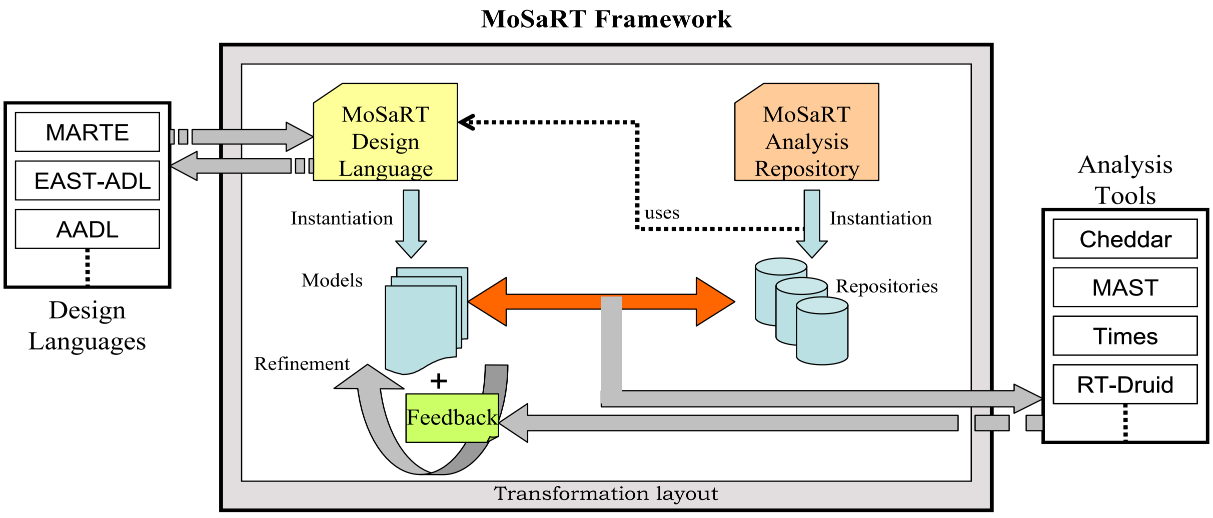


Figure 5.6: MoSaRT Framework

Consequently, we propose an intermediate framework named *MoSaRT* (Modeling-oriented Scheduling analysis of Real-Time systems). To partition the efforts, the intermediate framework plays the role of a bridge between the real-time design languages and the analysis tools (see Figure 5.6). It is based on two metamodels interacting with each other:

- MoSaRT Design Language, which is a domain specific language offering enough concepts and semantics to obtain models independent from any methodology, and to cover with few modifications, most exiting analytical models and easily extended to include also the future ones.
- MoSaRT Analysis Repository metamodel allows analysts to plug different theoretical studies and prototypes and ensures the interoperability with different analysis tools in order to compare their output results.

Figure 5.6 gives only an overview of our contributions. It shows a generic scenario due to the usage of the MoSaRT framework. This latter helps designers to analyze step by step their system designs during the design phase, which can be increasingly improved by applying the three following processes:

- Using the MoSaRT design language for system modeling, or for refining imported models.
- Selection of the analytical models and the relevant tests, by helping the real-time designers to extract the relevant elements from the models.

- Scheduling analysis, by offering to the real-time designers the equipped analysis tests which correspond to their models via the proposition of one or several analysis repositories.

The next chapters discuss the details of each contribution.

Chapter 6

MoSaRT Design Language

Contents

Abstract

This chapter is devoted to present the MoSaRT design language. It introduces our contribution referring to the research foundations and describe both functional and operational elements, and their semantics. Rules enabling a good usage of the MoSaRT design language are also presented.

I. Introduction

In this chapter we look for a real-time systems design as a result of a modeling driven by the scheduling analysis background as it is presented in Chapter 3.
The word "designer/designers" used in this chapter means modelers, architects, analysts, real-time researchers, etc. We precise the designer's type when it is necessary for a good understandability.
The next sections present the different capabilities of our proposed modeling language called "MoSaRT (Modeling Oriented Scheduling Analysis of Real-Time systems) design language". Section II introduces our contribution referring to the research foundations discussed earlier. Section III and Section IV describe both functional and operational elements. Section V presents the rules enabling a good usage of the MoSaRT design language. Section VI summarizes and concludes this chapter.

II. General description

When it comes to design a real-time system, the design process should provide a correct functional and operational design. By operational design, we mean the hardware architecture platform and the software platform (the software architecture and the timing behavior). To obtain a complete real-time system design, the software platform can also include the functional design (a software application design expressed with SysML [OMG07], UML [OMG04], EAST-ADL [DSLT05], etc.).
Several works (for instance [BBAL06, MTPG11, MKH03, BLN05]) are devoted to the specification of real-time systems at the functional level (also known as logical level). They go from a functional model (e.g. it can be a data flow model expressed in UML) in order to map the functional blocks to a set of threads automatically generated assuming execution on top of a specific execution platform (that is what we call the operational design). The mapping is generated via specific algorithms which differ depending on the followed approach. Moreover, some design languages such as UML-MARTE provide a set of concepts to embody those approaches. We recall that UML-MARTE requires the existence of a functional model as en entry point in order to be annotated by several stereotypes corresponding to non-functional requirements.
As our objective is to be the independent from any method, the MoSaRT design language does not require the existence of functional models to build a coherent design. Indeed, in complex industrial systems, many software modules are re-used in different system versions. Therefore, at the design stage, designers have already a number of information allowing to study partially the system. In addition, in case of time-partitioned systems that have appeared recently in avionics fields, the architect should resize its partitions at a very early design stage.
Furthermore, scheduling analysis models and tests evolve theoretically and much less industrially. The real-time research community may not be totally satisfied of the use of its theoretical studies in the industry. This dissatisfaction is related to many factors, among them, we can find the lack of expressiveness of the design languages used for the system modeling.
Focusing on the real-time scheduling theory, the main problem faced by designers during the analysis phase is how to get a valid design. However, it would be good to ensure first that model needing analysis is conceptually correct regardless of its schedulability correctness. That is what we call an "analyzable model". Thus, through MoSaRT design language we also aim to obtain analyzable models, hence facilitating the design tasks for analysts/real-time researchers who can be not well versed in the

modeling in particular via a model-driven development process.
The goal of the MoSaRT language is to provide models that are very close to the scheduling analysis compared to the design development or the code generation. Thus, we aim to provide generic real-time concepts allowing:

- modelers/architects (i) to construct real-time architectures to be analyzed, and (ii) to consolidate the logical models by the real-time properties.
- analysts/real-time researchers to build the timing analytical models supported by every analysis tool and presented in every research study.

This manner of use corresponds to the content presented in Section III.2.d of Chapter 2.

II.1. Overview

MoSaRT design language is conceived as a domain specific modeling language for real-time systems. Indeed, it is based on four concepts:

1. The platform is the required concept to model the two sides of the system architecture: hardware and software. This part allows to have an architectural model which is a part of the operational model.
2. The second one is the behavioral concept that expresses the dynamism and the vivacity of the architecture over a behavioral model.
3. The third concept is the analysis which enriches the earlier mentioned models by using real-time capabilities in order to have an analyzable design.
4. The last one is the functional concept that represents the functional part of the system. It does not affect directly the schedulability of the system, but taking this part into consideration may permit deep analyses (e.g. conditional model with if-then-else).

A design built via MoSaRT design language is based on two layers: the operational layer and the application layer. The operational layer is composed of the hardware model, the software architecture model and the behavioral model of this software architecture. The application layer corresponds to the functional model. Figure 6.1 shows the different kinds of models and their complementarity offering a set of viewpoints. Each viewpoint represents a combination of different kinds of models focusing on specific sides of the global system.
In the sequel, we use Ecore formalism [SBPM08] to describe elements of the MoSaRT design language. Ecore is the implementation of the Meta Object Facility (MOF) provided by Eclipse [Ecl]. The Ecore formalism is like the UML class diagram formalism, and is dedicated to meta-modeling. Figure 6.2 shows that the design-level of MoSaRT language corresponds to the metamodel level. Thus, every instance of the MoSaRT language conforms to the MoSaRT metamodel.

II.2. General Structure

Elements composing the MoSaRT design language are organized in a set of packages. Every package regroups a set of elements referring to their common semantics. Figure 6.3 shows various packages of the general structure of MoSaRT language and their interactions.

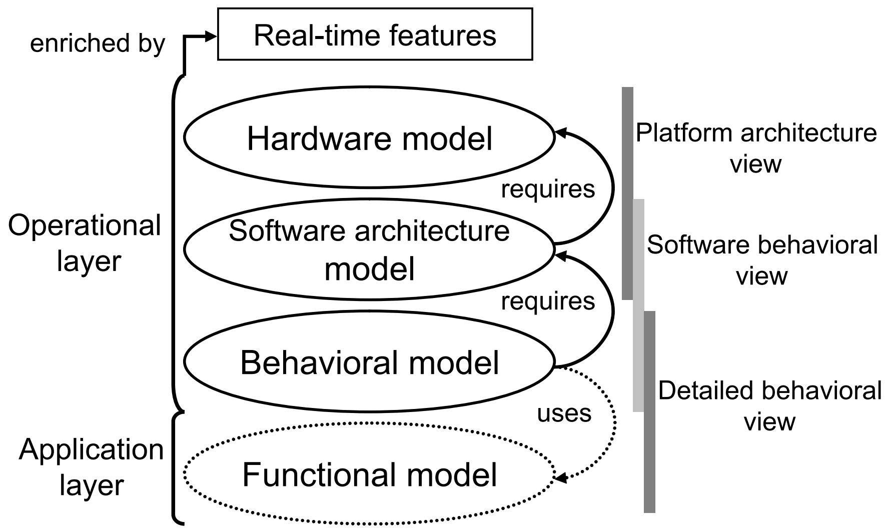


Figure 6.1: Different views of a real-time system designed through MoSaRT

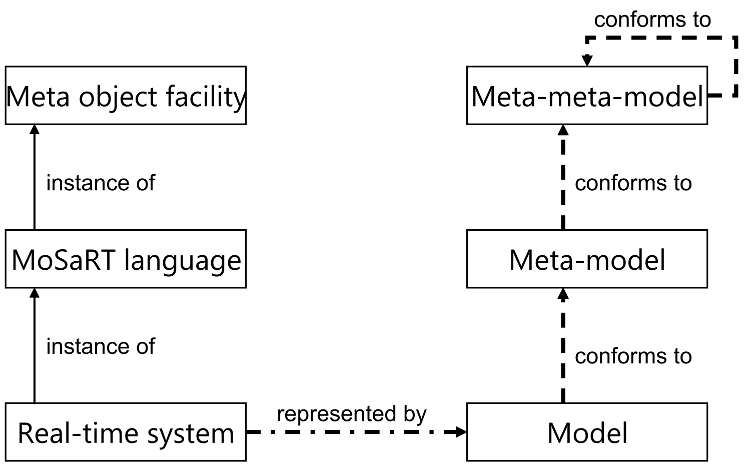


Figure 6.2: MoSaRT design language corresponds to the meta-model level

Formalization and semantics Figure 6.4 depicts the root elements used by the MoSaRT language to build the real-time systems. Let `GlobalSystem` be a set of the systems constructed via MoSaRT language.
For every global system $Gs \in$ `GlobalSystem`, $Gs = <Hs, Ss, Fs>$, such that:
Hs is the hardware side of the global system, and $Hs \in$ `SystemHardwareSide` which is the set of hardware sides;
Ss is the software side of the global system, and $Ss \in$ `SystemSoftwareSide` which is the set of the software sides; and
Fs is the functional side of the global system, and $Fs \in$ `SystemFunctionalSide` which is the set of the functional side.

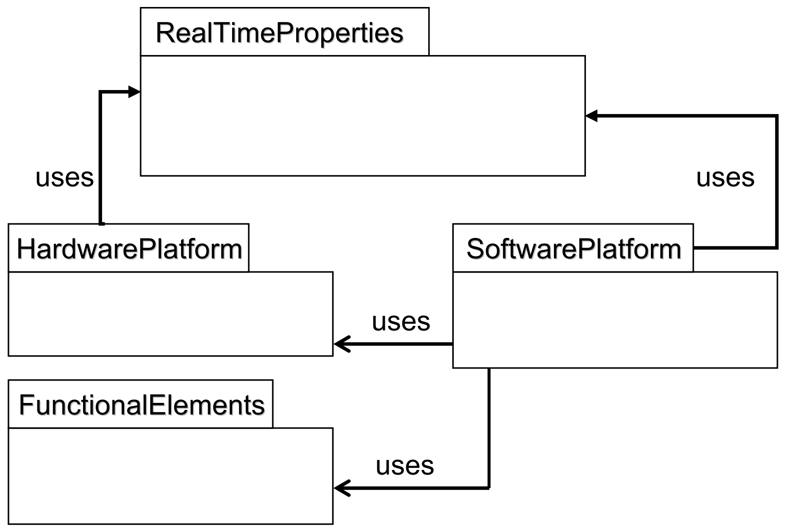


Figure 6.3: General structure of MoSaRT language

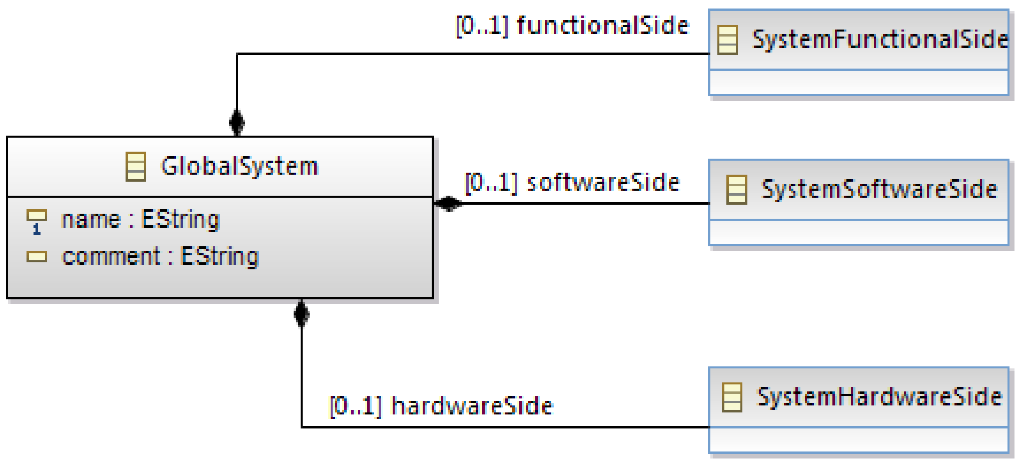


Figure 6.4: The root elements forming the different sides of real-time systems

Every instance of the `GlobalSystem` class is characterized by a unique `name` and a `comment`. While the name is mandatory, because it serves to distinguish models, the `comment` may be used for details and explanation.

III. Functional Side Description

III.1. Overview

The functional side *Fs* represents the content of the logical elements allowing designers to have a functional model. In our case, we have chosen the Unified Modeling Language (UML) [OMG04] to design functional elements. We have chosen the UML due to its popularity in both academic and industrial sectors. Then, a functional model can be illustrated by different diagrams, like a class diagram.

III.2. Formalization and semantics

Fs is the root element of the `FunctionalElements` package. *Fs* can be composed with different kinds of functional models such as UML or SysML. However, currently only UML is supported. Indeed, UML has been injected as a sub-package in the `FunctionalElements` package as shown in Figure 6.5.

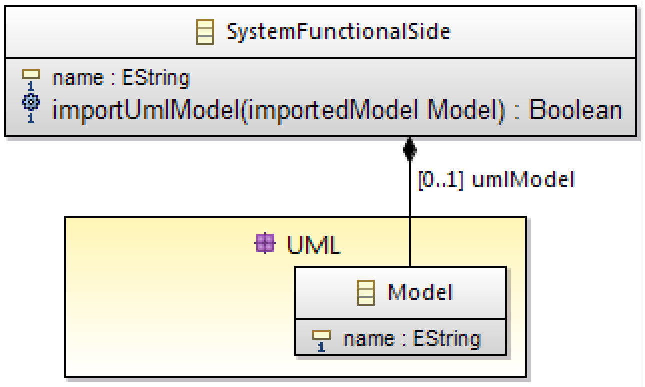


Figure 6.5: Concepts used for modeling functional sides

Every instance of the `SystemFunctionalSide` class represents the logical part of the designed real-time system. The `SystemFunctionalSide` class contains the root element of the UML metamodel which is the `Model` class, hence every model instance conforms to the UML metamodel can be either built through MoSaRT language or imported (if it is built outside MoSaRT language) thanks to the operation `importUmlModel`. This latter returns `true` in case of a successful importation.

IV. Operational Side Description

Referring to the context of our work, the operational layer is the most important side in the real-time systems. That is why we give a special importance to this section. As the operational layer is applied to a set of quantitative scheduling analyses, we prefer to first introduce the real-time properties before the other concepts related to the platform architecture, the hardware execution and the software behavior.

IV.1. Real-Time Properties

Real-time properties represent the main design service offered by the MoSaRT language. They are the analytical timing elements for adding the timing characteristics into the designed system.

IV.1.a. Overview

The `RealTimeProperties` package contains many sub-packages (see Figure 6.6), which their names are prefixed by *Rtp* (initials of Real-time properties).

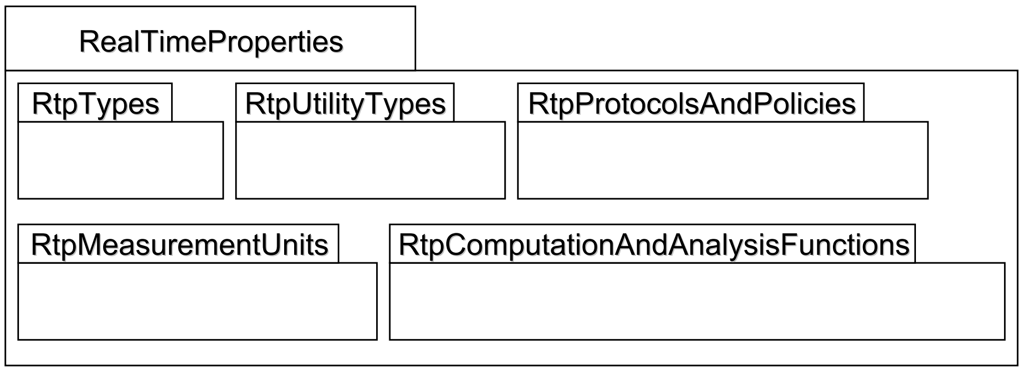


Figure 6.6: Internal structure of the `RealTimeProperties` package

1. The `RtpMeasurementUnits` package proposes the various units used in the real-time system design, like time units, size units, frequency units, etc.
2. The `RtpUtilityTypes` package offers some utility elements like lists, matrix, intervals, durations, etc.
3. The package called `RtpTypes` defines the classic real-time notions like the periodicity, the deadline, the execution time, etc.
4. There is also another package called `RtpProtocolsAndPolicies` which proposes classic scheduling policies, resource access mechanisms, network communication protocols, etc.
5. The `RtpComputationAndAnalysisFunctions` package assembles some functions enabling the computation of some properties like the Weibull distribution law which is a function that may be used to generate the execution times.

There are two sets of real-time properties: input and output property sets. The input property is a property entered by the designer, such as the scheduling policy, the period, etc. The output property represents a storage of the results found after the analysis process, such as the response time, the blocking time, the utilization factor, etc. Yet, some properties can be simultaneously considered as input/output properties in some cases, like the execution time.

IV.1.b. Definitions and semantics

As we cannot exhaustively present the underlying concepts, we have only chosen to present the definitions of the deadline property called `RtpDeadlineType` and the execution time property called `RtpExecutionTimeType`. These definitions will be done step by step in order to give an idea about the contents of the sub-packages presented above. Moreover, we focus on some details to show the **modularity**, the **genericity** and the **extensibility** of the real-time properties.

- Modularity: every real-time property is composed from a set of modules which enable to ensure the re-usability and facilitate modifications without having unwanted impacts on other elements.
- Genericity: the creation of every real-time property as a generic concept helps to cover a large number of task models and their specificities.
- Extensibility: to enrich the semantics of the existing elements, they should be easily extensible. The modularity and the genericity ease the extensibility.

Definition and semantics of the deadline property. The way the deadline property is conceived highlights the modularity and generalization capabilities of the MoSaRT design language.

1. *RealVariable* property: element of the `RtpUtilityTypes` package

   Let Rv be a real variable, where $Rv \in$ `RealVariable` which is a set of real numbers with variable values, and `RealVariable`=`SimpleReal` ∪ `UnknownReal` ∪ `MeasuredReal` (see Figure 6.7), where:

   - $Rv \in$ `SimpleReal`, means Rv represents a classic real value. For example, $Rv=$ <`value`=6.5>.
   - $Rv \in$ `UnknownReal`, means Rv is a real variable characterized by the name of the variable as a value. For example, Rv=<`value`='ABC'>. That looks like the declaration of variables in programming languages (e.g. "*RealVariable ABC = null ;*").
   - $Rv \in$ `MeasuredReal`, means Rv was an element of `UnknownReal`, but after an analysis or a computation process, the value becomes known and may be provided by one or several tools (if the computation process is applied many times). For instance, Rv=<`value`=6.5, `unknownValueName`='ABC', `toolUsedDescriptions`='the value is calculated by my tool', allOldValues=∅>

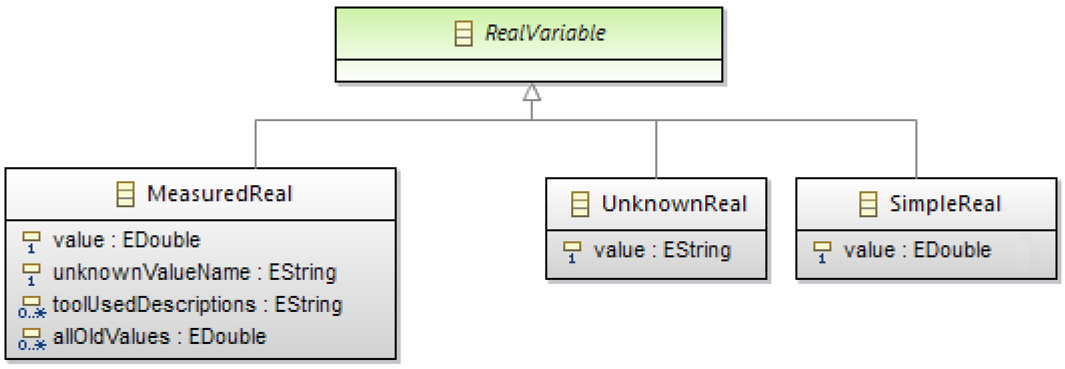


Figure 6.7: *RealVaribale* class belongs to the `RtpUtilityTypes` package

2. *TimeUnits* property: element of the `RtpMeasurementUnits` package
   `TimeUnits` is the enumeration of units used to express the time, then `TimeUnits` = {us, ms, s, min, hr, day} (see Figure 6.8).

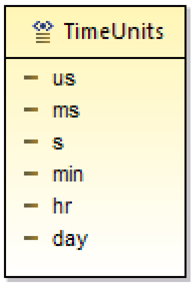


Figure 6.8: *TimeUnits* enumeration

3. *RealDurationType* property: element of the `RtpUtilityTypes` package
   `RtpDurationType` is a set of durations, where every duration Rd is expressed by a time unit *unit* and a real variable *value*:
   $Rd = <unit, value>$, where $unit \in$ `TimeUnits` and $value \in$ `RealVariable`.

   Figure 6.9 shows the formalization of `RtpDurationType` as an Ecore class with *unit* and *value* attributes.

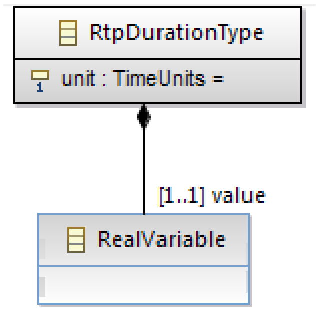


Figure 6.9: *RtpDurationType* class

4. *RtpDeadlineType* property: element of the `RtpTypes` package
   `RtpDeadlineType` is a set of the deadline properties, where every deadline Dp is characterized by a duration *deadline*, and $deadline \in$ `RtpDurationType`.

   By using Ecore, we have conceived the set of the deadline properties `RtpDeadlineType` as an Ecore class (see Figure 6.10). Furthermore, while the response time property, the blocking type property, the period property, etc. are related to the duration values, they are somehow conceived like the `RtpDeadlineType` property.

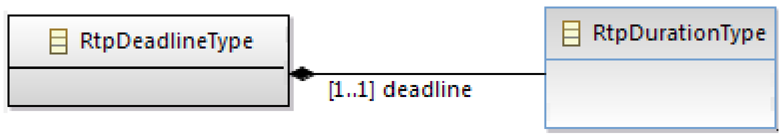


Figure 6.10: *RtpDeadlineType* class

Definition and semantics of the execution time property As it is shown in the conception process of the deadline property, the specificity of the real-time properties via MoSaRT language is the *generalization* capability. Every timing property represents a generic concept permitting to cover different existing task models and real-time situations. Indeed, future analytical models with specific concept values can also be covered by injecting the new property kind as a specific type inheriting from the generic concept. This way ensures an easy extensibility.

1. Definition of the `RtpExecutionTimeType` property as a generic concept
   Let *ETprop* be an execution-time property, i.e. $ETprop \in$ `RtpExecutionTimeType`, and `RtpExecutionTimeType` is a set of execution-time properties.
   $ETprop = <Val, Tunit>$ where $Tunit \in$ `TimeUnits` and *Val* represents an instance of one of different possible formats for expressing the execution time value. Figure 6.11 illustrates several formats of an execution time value.

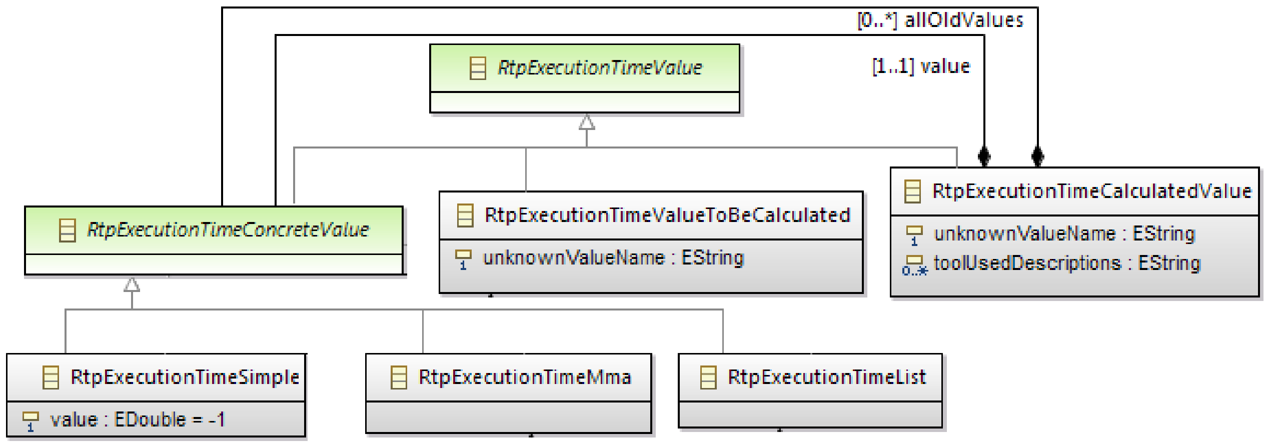

[0..*] allOldValues
[1..1] value
RtpExecutionTimeValue
RtpExecutionTimeConcreteValue
RtpExecutionTimeValueToBeCalculated
unknownValueName : EString
RtpExecutionTimeCalculatedValue
unknownValueName : EString
toolUsedDescriptions : EString
RtpExecutionTimeSimple
value : EDouble = -1
RtpExecutionTimeMma
RtpExecutionTimeList

Figure 6.11: Some types of the execution time value

   Thus, *Val* can be unknown and then calculated by calling a sensitivity analysis test or generated by a probabilistic distribution law. In case where *Val* is known, it can be a simple value (i.e. a real number), or a list of values, etc.

2. Extending the *RtpExecutionTimeType* property
   In order to extend the execution time value *Val* by adding new formats, one can easily extend the abstract class called `RtpExecutionTimeConcreteValue` as shown in Figure 6.12. So, to add a format allowing to support the execution time value as an interval, the `RtpExecutionTimeInterval` class can be added as a sub-class inheriting from `RtpExecutionTimeConcreteValue`.

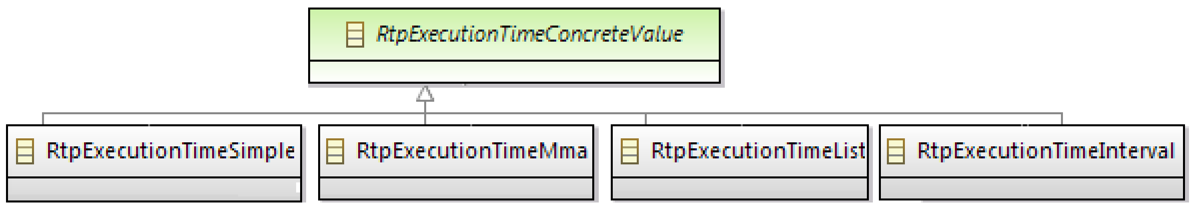


Figure 6.12: Extending the value of the RtpExecutionTime property

IV.1.c. Example

Part A of Listing 6.1 shows an example, where the execution time value is undefined and needs to be calculated regarding the real-time situation of the system. So, the execution time value is considered as a variable called "valueC3".
Once the content of the "valueC3" becomes known (e.g. after a specific calculation), the value of the execution time is presented as it is shown in Part B of Listing 6.1. It means that the calculated value is provided by a third-party tool implementing the appropriate calculating function.
Thus, we can say that `RtpExecutionTime` is an input/output property. Furthermore, The process of calling different analysis tools to calculate the value can be repeated many times enabling designers to compare the provided results.

Listing 6.1: Example showing the utilization of the `RtpExecutionTime` property that the formalism conforms to its concrete syntax

```
(A): Execution Time (value = ToBeCalculated(?valueC3?), unit = ms)


(B): Execution Time (value = CalculatedValue(value = 13.75,
                                             unknownValueName = valueC3,
                                             usedTools = [LiasLabRT],
                                             oldValues = []),
                     unit = ms)
```

IV.1.d. Operational semantics of real-time properties

Most real-time properties are characterized by operational semantics. The `RtpExecutionTimeType` class (see Figure 6.13) is characterized by a set of operations. The `greater`, `lower` and `equals` operations are dedicated for comparison. The `convertsTo` operation enables to convert the value of the execution time from a time unit to another one. Finally, the `adoptsCalculatedValue` operation performs an internal transformation in order to adopt the value provided after an analysis or a calculation process instead of the current value.

IV.1.e. Discussion

Even if the real-time properties are scattered in different packages, those used by the elements of the operational side are such that their names start by "Rtp" and finish by "Type". Values of several

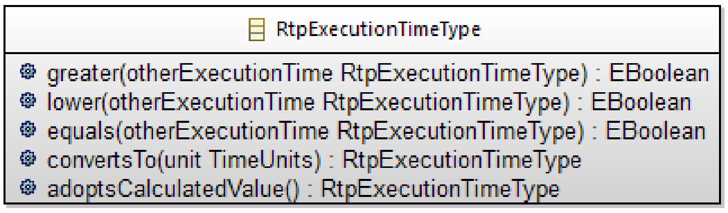

RtpExecutionTimeType
greater(otherExecutionTime RtpExecutionTimeType) : EBoolean
lower(otherExecutionTime RtpExecutionTimeType) : EBoolean
equals(otherExecutionTime RtpExecutionTimeType) : EBoolean
convertsTo(unit TimeUnits) : RtpExecutionTimeType
adoptsCalculatedValue() : RtpExecutionTimeType

Figure 6.13: Operations of the `RtpExecutionTimeType` class

real-time properties are traditionally expressed as numbers. To ease the extensibility of the MoSaRT design language, those values are expressed as derivable and refinable concepts.

IV.2. System's Hardware Side

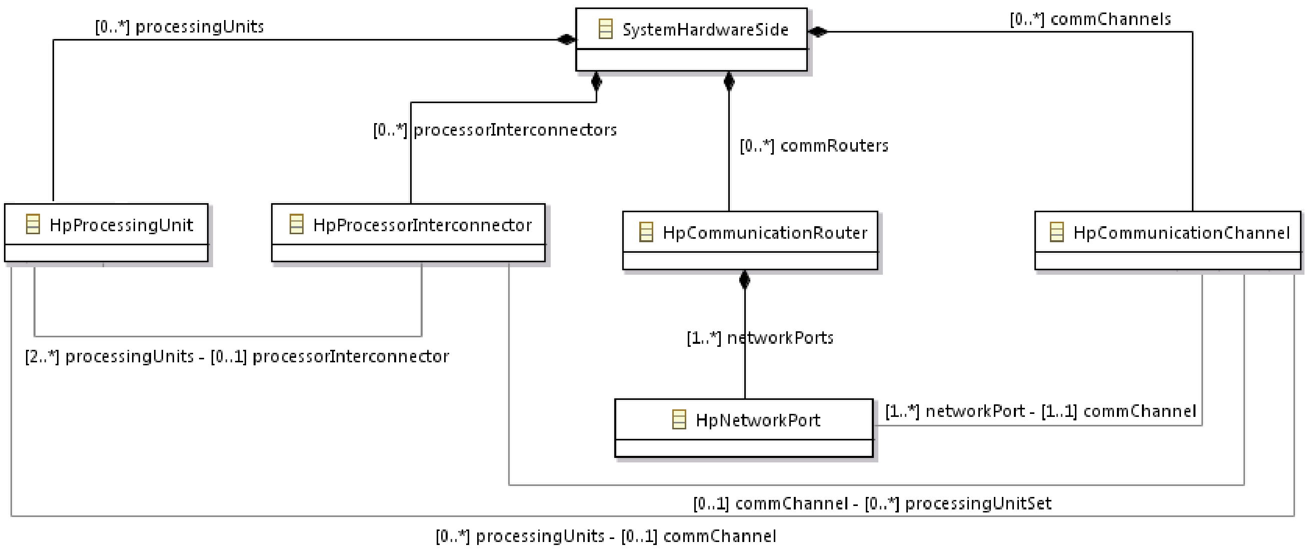

SystemHardwareSide
[0..*] processingUnits
[0..*] commChannels
[0..*] processorInterconnectors
[0..*] commRouters
HpProcessingUnit
HpProcessorInterconnector
HpCommunicationRouter
HpCommunicationChannel
[2..*] processingUnits - [0..1] processorInterconnector
[1..*] networkPorts
HpNetworkPort
[1..*] networkPort - [1..1] commChannel
[0..1] commChannel - [0..*] processingUnitSet
[0..*] processingUnits - [0..1] commChannel

Figure 6.14: Part of the HardwarePlatform package of the MoSaRT design language

The hardware side *Hs* contains different hardware elements (like processors, routers, channels, etc.) to construct hardware models. The hardware model represents the hardware architecture of the designed real-time system. Figure 6.14 represents a part of the `HardwarePlatform` package where the name of each element is prefixed by "Hp" (initials of Hardware Platform).

IV.2.a. Definitions and semantics

In the following, we present the real-time properties, the syntax and the semantics of the hardware elements.
Table 6.1 summarizes the hardware elements and their real-time properties.

Elements	**Real-Time properties**
`HpProcessingUnit`	`rtpUtilizationFactor`, `rtpCommunicationSpeed`
`HpProcessorInterConnector`	`rtpMultiprocessorCategory`
`HpCommunicationRouter`	
`HpNetworkPort`	`rtpFlowRate`, `rtpNetworkProtocol`, `rtpTransmissionMode`
`HpCommunicationChannel`	

Table 6.1: Summary of the elements of the `HardwarePlatform` package

▷ `HpProcessingUnit` instances can be used to design CPUs. When more than one processor exist in the hardware model, the processors have to be related via the `HpProcessorInterConnector` instances or via the network.

▷ `HpProcessorInterConnector` instances are used in case of multicore systems. They connect the cores of the same node in order to form multicore processors. The `rtpMultiprocessorCategory` property is mandatory to mention the kind of cores if they are homogeneous, uniform or heterogeneous.

▷ The `HpCommunicationRouter` instances are used in the design of real-time distributed systems. Every instance of `HpCommunicationRouter` has to contain at least one instance of the `HpNetworkPort`. Every `HpNetworkPort` instance represents a network, hence it is characterized by a flow, network protocol (e.g. Ethernet, CAN, FDDI, etc.), and a transmission mode (it may be simple, half duplex or full duplex). When an instance of `HpCommunicationRouter` contains many `HpNetworkPort` instances, that means that every port is a separate network, and the transition from a network to another one has to be via the `HpCommunicationRouter` instances. The `HpCommunicationChannel` plays the role of mediums linking distributed sites. Then, it can link `HpProcessingUnit` instances, `HpProcessorInterConnector` instances and `HpNetworkPorts` instances. When an instance of HpCommunicationChannel links a set of `HpNetworkPort` instances, those ports do not have to uappertain to the same `HpCommunicationRouter` instance.

▷ Graphical concrete syntax: we have chosen a graphical concrete syntax for the hardware elements to express the hardware model (see Figure 6.15).

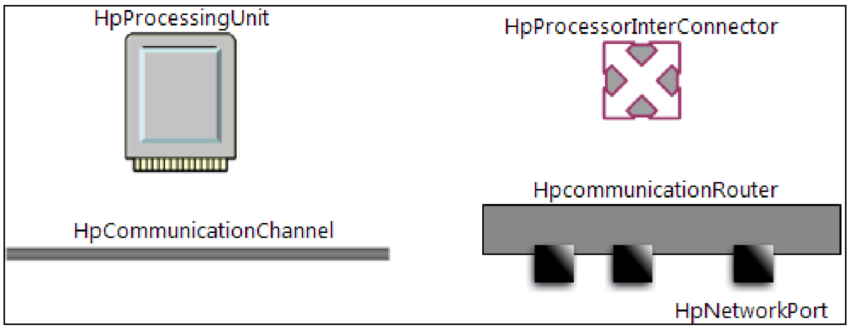


Figure 6.15: Concrete syntax corresponding to the hardware elements of MoSaRT design language

IV.2.b. Example

Figure 6.16 shows a hardware architecture of a real-time system. The system is composed of two nodes communicating through a CAN (Controller Area Network) network. The first node is uniprocessor, and the second node is a multi-core processor containing four cores.

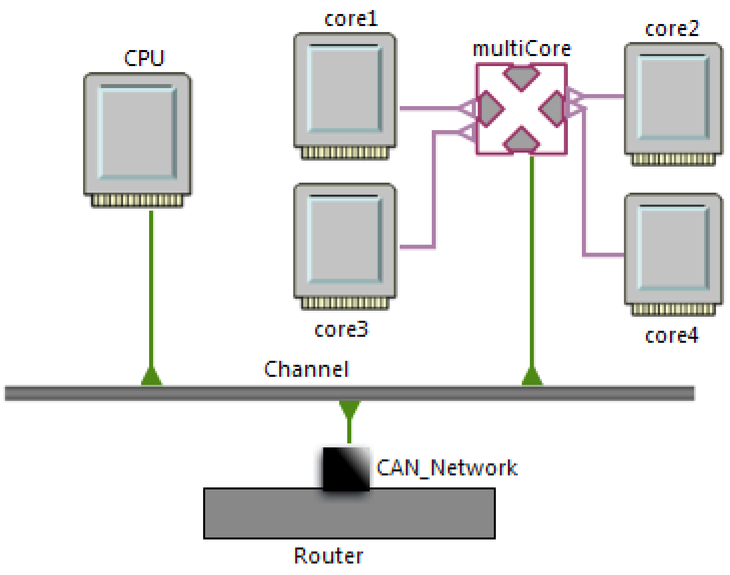


Figure 6.16: Example of a hardware architecture in MoSaRT language

IV.3. System's Software Side

The software side *Ss* is composed of two kinds of models representing the software platform: the software architecture model and the software behavior model.

1. The `SoftwareOperators` package is a sub-package of `SoftwarePlatform` package allowing to design the software architecture of the real-time system by using elements like tasks, interaction resources, processes, etc. Figure 6.17 shows a part of elements called software operators. The names of these elements are prefixed by "So" (initials of Software Operators). The mapping of the hardware model to the software architecture model gives a global view of the platform architecture.

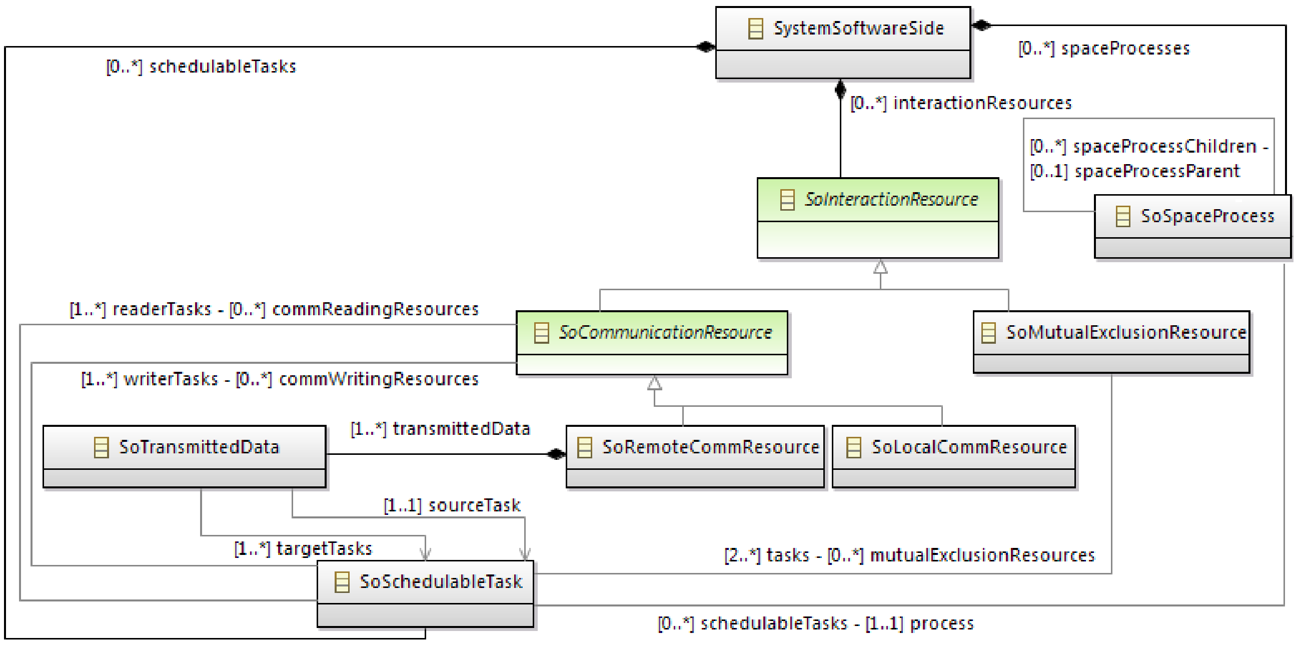

SystemSoftwareSide
[0..*] schedulableTasks
[0..*] spaceProcesses
[0..*] interactionResources
[0..*] spaceProcessChildren -
[0..1] spaceProcessParent
SoInteractionResource
SoSpaceProcess
[1..*] readerTasks - [0..*] commReadingResources
SoCommunicationResource
SoMutualExclusionResource
[1..*] writerTasks - [0..*] commWritingResources
[1..*] transmittedData
SoTransmittedData
SoRemoteCommResource
SoLocalCommResource
[1..1] sourceTask
[1..*] targetTasks
[2..*] tasks - [0..*] mutualExclusionResources
SoSchedulableTask
[0..*] schedulableTasks - [1..1] process

Figure 6.17: Part of SoftwareOperators package of MoSaRT design language

2. The second model of the software side *Ss* is the behavioral model. It is constructed by using the elements of the `SoftwareBehavior` package. The behavioral model allows to express the behavior of the software architecture, such as the triggers, the precedence relationships, the task activity which represents how each task reacts, and the step elements which are very important to have a detailed description of the task activities. Figure 6.18 shows a part of the `SoftwareBehavior` package. The names of these elements are prefixed by "Sb" (initials of Software Behavior). The mapping of the software architecture model to the behavioral model leads to a global software behavioral view.

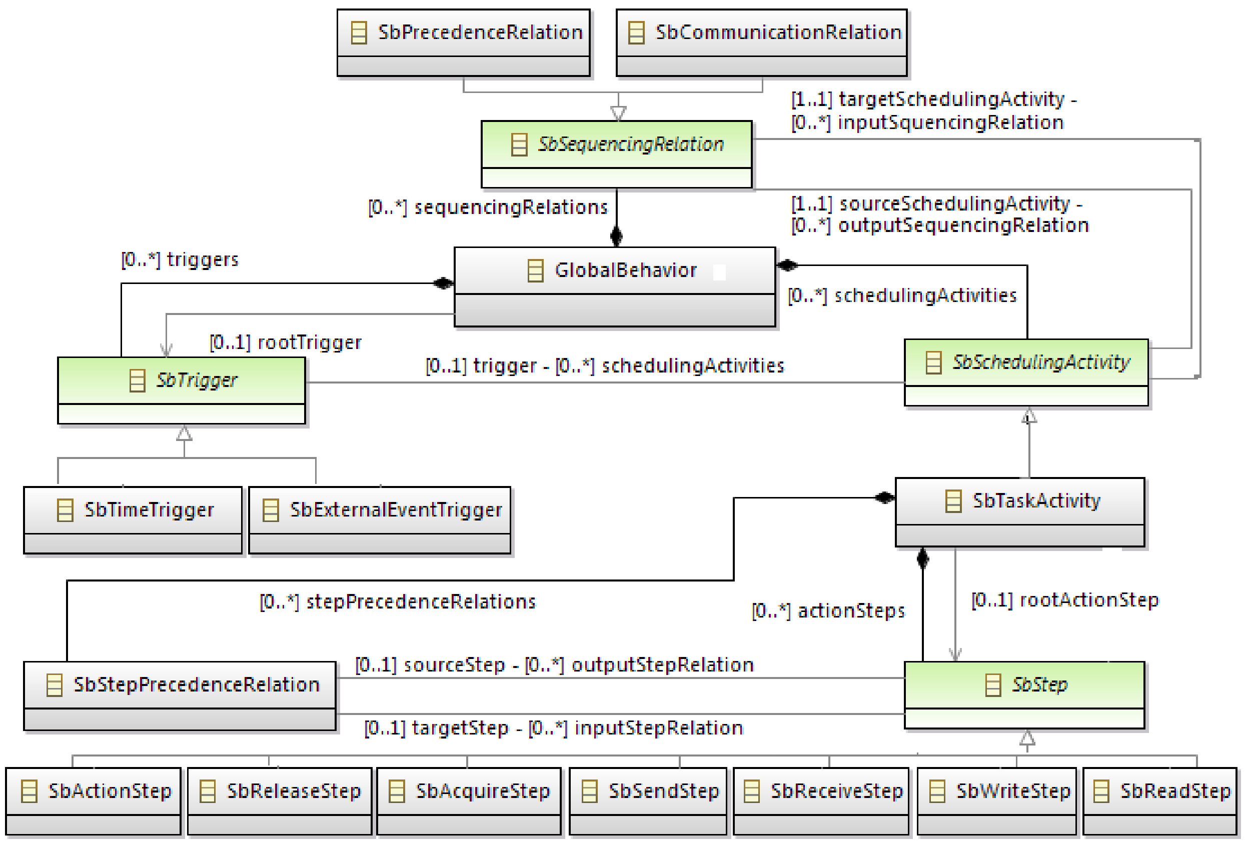

SbPrecedenceRelation
SbCommunicationRelation
SbSequencingRelation
[1..1] targetSchedulingActivity - [0..*] inputSquencingRelation
[0..*] sequencingRelations
[1..1] sourceSchedulingActivity - [0..*] outputSequencingRelation
[0..*] triggers
GlobalBehavior
[0..*] schedulingActivities
[0..1] rootTrigger
SbTrigger
[0..1] trigger - [0..*] schedulingActivities
SbSchedulingActivity
SbTimeTrigger
SbExternalEventTrigger
SbTaskActivity
[0..*] stepPrecedenceRelations
[0..*] actionSteps
[0..1] rootActionStep
SbStepPrecedenceRelation
[0..1] sourceStep - [0..*] outputStepRelation
SbStep
[0..1] targetStep - [0..*] inputStepRelation
SbActionStep
SbReleaseStep
SbAcquireStep
SbSendStep
SbReceiveStep
SbWriteStep
SbReadStep

Figure 6.18: Part of the SoftwareBehavior package of MoSaRT design language

IV.3.a. Definition and semantics of the `SoftwareOperators` package elements

Thereafter, we present the real-time properties, the syntax and the semantics of the `SoftwareOperators` package elements.

Elements	**Real-Time properties**
`SoSpaceProcess`	`rtpSchedulingPolicy` `rtpTaskAllocation`
`SoSchedulableTask`	`rtpMutualExclusionResourceUtilization`

`SoLocalCommResource`	`rtpCommunicationMechanism`
	`rtpWaitingQueuePolicy`
	`rtpProtectProtocol`

`SoRemoteCommResource`	
`SoTransmittedData`	`rtpDataSize`

Table 6.2: Summary of the elements of the `SoftwareOperators` package

Table 6.2 summarizes the software architectural elements and their real-time properties.
▷ `SoSpaceProcess` instances are used to express the schedulers/dispatchers and to define the scheduling policy via the `rtpSchedulingPolicy` property. They also represent the processes and the space memories shared by the tasks. An instance of the `SoSpaceProcess` class can inherit from another instance in order to design the hierarchical schedulers. A `SoSpaceProcess` instance can be matched with one processor or with one or several cores in case of a multicore system. When, a `SoSpaceProcess` instance is related to several cores, the kind of the task allocation (i.e. partitioned, restricted migration, full migration) must be specified via the `rtpTaskAllocation` property.

▷ Every `SoSchedulableTask` instance illustrates a task/thread of a real-time system. Each `SoSchedulableTask` instance belongs to a `SoSpaceProcess` instance. It can be allocated to one or several processors of the same node, where those processors are related to the same `SoSpaceProcess` instance than the task. When a `SoSchedulableTask` instance shares critical resources, the `rtpMutualExclusionResourceUtilization` property should be specified to indicate the time required for each resource access.

▷ A `SoLocalCommResource` instance can be shared by a set of tasks related to the same process. Otherwise, the `SoRemoteCommResource` instances can be used. Every `SoLocalCommResource` instance represents shared data, and it can be characterized by the `rtpProtectProtocol` property (e.g. PCP, PIP [SRL90]) if it is protected. Moreover, the `rtpCommunicationMechanism` property defines the type of the mechanism used for communication, such as a mailbox mechanism or a data queue mechanism, etc.

▷ Every `SoRemoteCommResource` instance is shared by tasks which are not related to the same process. The `SoRemoteCommResource` instances are composed from a set of `SoTransmittedData` instances, where every `SoTransmittedData` instance illustrates a data sent by only one task and received by several other tasks. Every `SoTransmittedData` instance is characterized by the `rtpDataSize` property.

▷ Graphical concrete syntax: Figure 6.19 shows the graphical syntax of elements used to represent the software architecture model. The concrete syntax of the `SoLocalCommResource` element varies according to its real-time properties.

IV.3.b. Example of software architecture model

As an example of utilization of the `SoftwareOperators` package, Figure 6.20 represents a software architecture model of a real-time system. The model contains seven tasks managed by three schedulers, and two of them are hierarchical. It also contains two interaction resources shared by three tasks (mutual exclusion resource and box communication resource), and a remote communication resource allowing to transmit data between two tasks.

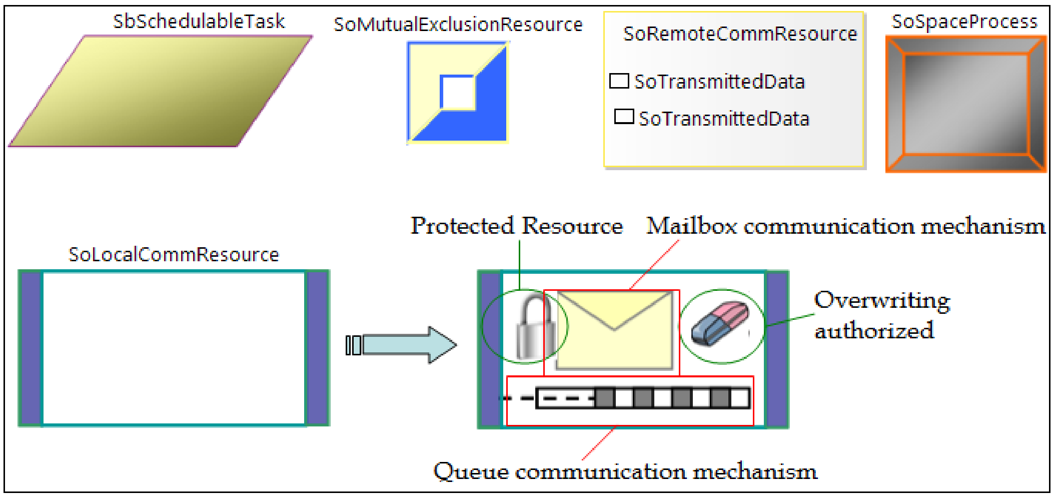


Figure 6.19: Concrete syntax corresponding to the elements enabling to get the software architecture model. The concrete syntax of the SoLocalCommResource is decorated by some features which vary according to the value of some real-time properties.

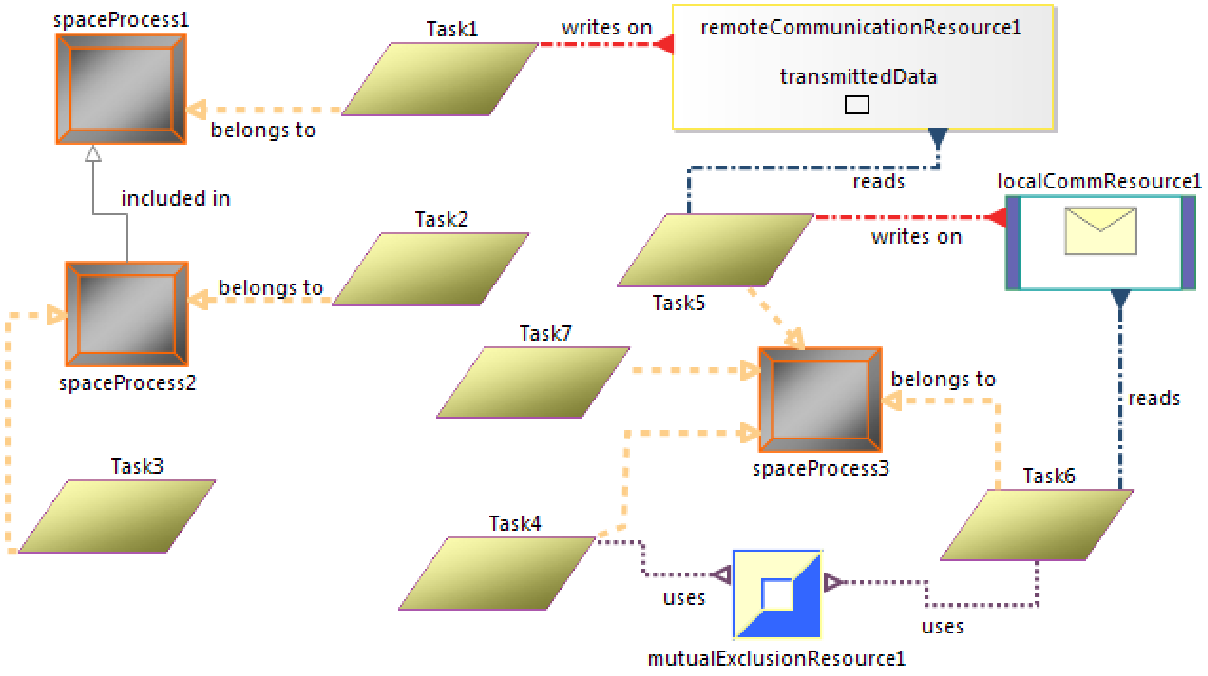


Figure 6.20: Example of a software architecture through MoSaRT language

IV.3.c. Definitions and semantics of the `SoftwareBehavior` package elements

Below, we present the real-time properties, the syntax and the semantics of some elements existing in the `SoftwareBehavior` package.

Elements	Real-Time properties
SbTimeTrigger	`rtpReleaseTime` `rtpPeriodicity`
`SbExternalEventTrigger`	

SbTaskActivity	`rtpDeadline`
	`rtpExecutionTime`
	`rtpOffset`
	`rtpBlockingTime`
	`rtpRepetition`
	`rtpResponseTime`
	`rtpPriority`
	`rtpCriticality`
	`rtpPreemptibility`
	`rtpSelfSuspension`

`SbCommunicationRelation`	
`SbPrecedenceRelation`	

`SbActionStep`	`rtpDeadline`
	`rtpExecutionTime`
	`rtpBlockingTime`
	`rtpRepetition`

`SbStepPrecedenceRelation`	

Table 6.3: Summary of the elements of the `SoftwareBehavior` package

Table 6.3 summarizes the behavioral elements and their real-time properties.
▷ We have shown in Example 2 of I.2.c of Chapter 5, the accuracy of expressiveness has an impact on the behavioral conception of the system, then on its schedulability analysis. Unlike other design languages (such as AADL) which attach the activation patterns to the real-time tasks, via the MoSaRT design language we have chosen to integrate the concept of triggers. Therefore, the activation patterns are related to triggers instead of tasks. So, a `SbTimeTrigger` instance corresponds to a periodic or sporadic trigger which triggers one or several tasks (instances of `SbTaskActivity` presented hereafter) or parts of tasks. Every `SbTimeTrigger` instance has a `rtpReleaseTime` property defining its first activation and a `rtpPeriodicity` property to specify its activation pattern. The `SbExternalEventTrigger` instances represents triggers by external events, where the events may occur aperiodically. For a consistency reason, the behavioral model requires a root trigger. This latter represents by default the time reference for the other triggers release times. Yet, the release times can also be expressed according to the execution of the `SbTaskActivity`. Indeed, the execution behavior of a task can play the role of the time reference. For example, let *Tr* be a `SbTimeTrigger` instance and triggers *TA1* which is a `SbTaskActivity` instance. The release time of *Tr* can be expressed according to the execution of

another `SbTaskActivity` instance called *TA2* as follows:
"Release Time (value = 2.5, unit = ms, TimeReference (after_the_end_of 3 execution(s) of TA2))".
It means that the first activation of the trigger *Tr* will be happen 2.5 ms after the end of the third job of the task represented by the `SbTaskActivity` instance *TA2*.

▷ Every instance of `SbTaskActivity` represents the behavior of a task, that was already created in the software architecture model (i.e. an instance of the `SoSchedulableTask` class). Then, there are so many `SbTaskActivity` instances as the `SoSchedulableTask` instances. The `SbTaskActivity` instances are defined by a set of real-time characteristics as those presented in Chapter 3. The `rtpRepetition` property represents the number of jobs. If it is specified that means that the `SbTaskActivity` will execute only N times, where N is the value of the `rtpRepetition` property. A `SbTaskActivity` instance can be triggered by its own trigger or by another `SbTaskActivity` instance when a communication or a precedence relationship exists.

▷ The precedence relationship (instance of `SbPrecedenceRelation`) represents a synchronization between the `SbTaskActivity` instances. Moreover, the existence of a communication relationship between two `SbTaskActivity` instances implies the existence of a shared communication resource in the software architecture model. In the case when more than one input/output relation exists for the same behavioral element, the 'AND' semantics are assumed. For instance, if a `SbTaskActivity` instance 'A' is preceded by other instances, that means 'A' will be triggered only after completion of every previous instance. If a `SbTaskActivity` instance is preceded by another instance and related to a `SbTimeTrigger` instance, this latter is considered as a watch-dog timer with a timeout equals the activation pattern.

▷ Through the behavioral model, the content of every `SbTaskActivity` instance can be defined thanks to the step elements. `SbActionStep` instances describe the elementary actions of the `SbTaskActivity` instances like the read action, the send action, the release action, etc. Moreover, every `SbTaskActivity` instance may be characterized by a set of real-time properties. If more than one input/output relation exist for the same step element, the 'OR' semantics are assumed. For instance, if a step element precedes more than one step element, this means that the next step element which has to be executed depends on the evaluation of a conditional statement. Then, this conditional statement is represented by an operation of the functional model. As an example, the use of 'OR' semantics allows to express a conditional task model (with "if-then-else" or "switch case").

▷ Graphical concrete syntax: Figure 6.21 shows the graphical syntax of elements used to get the behavioral model.

IV.3.d. Example of behavioral model

Figure 6.22 shows a behavioral model of a real-time system. It contains several `SbTaskActivity` instances. Via the `SbTaskActivity` instances we can also express transactions (or tasks with offsets) such as *taskActivity5*, *taskActivity6* and *taskActivity7* shown in Figure 6.22. The existence of a communication relationship between two task activities implies the existence of a shared communication resource in the software architecture model.

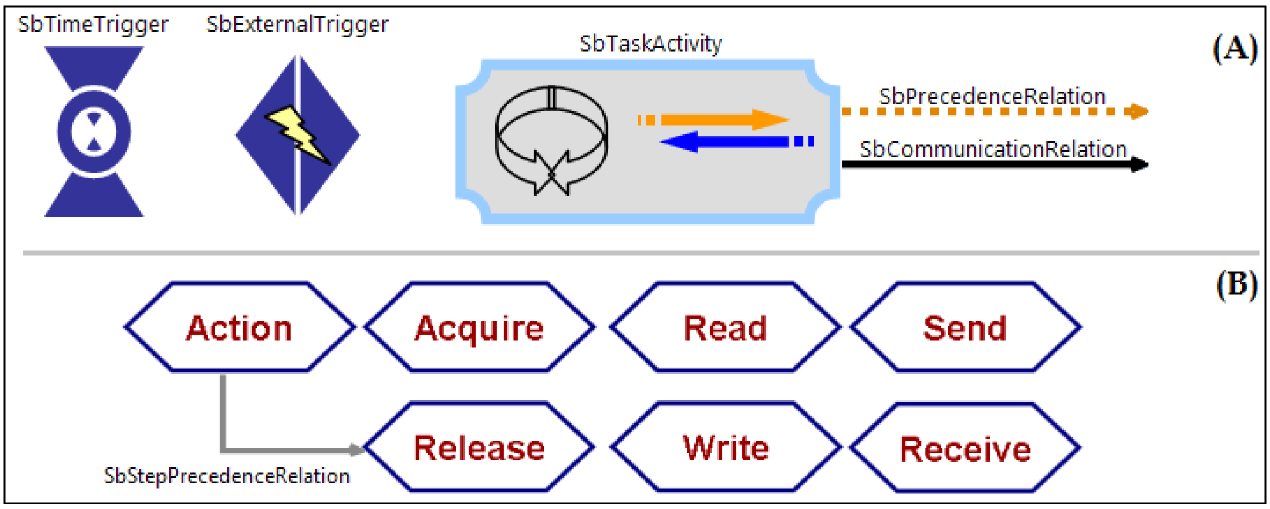


Figure 6.21: Concrete syntax corresponding to the elements enabling to get behavioral models. (A): elements used to design the global behavior of a real-time system. (B): elements used to design the core of each `SbTaskActivity` element.

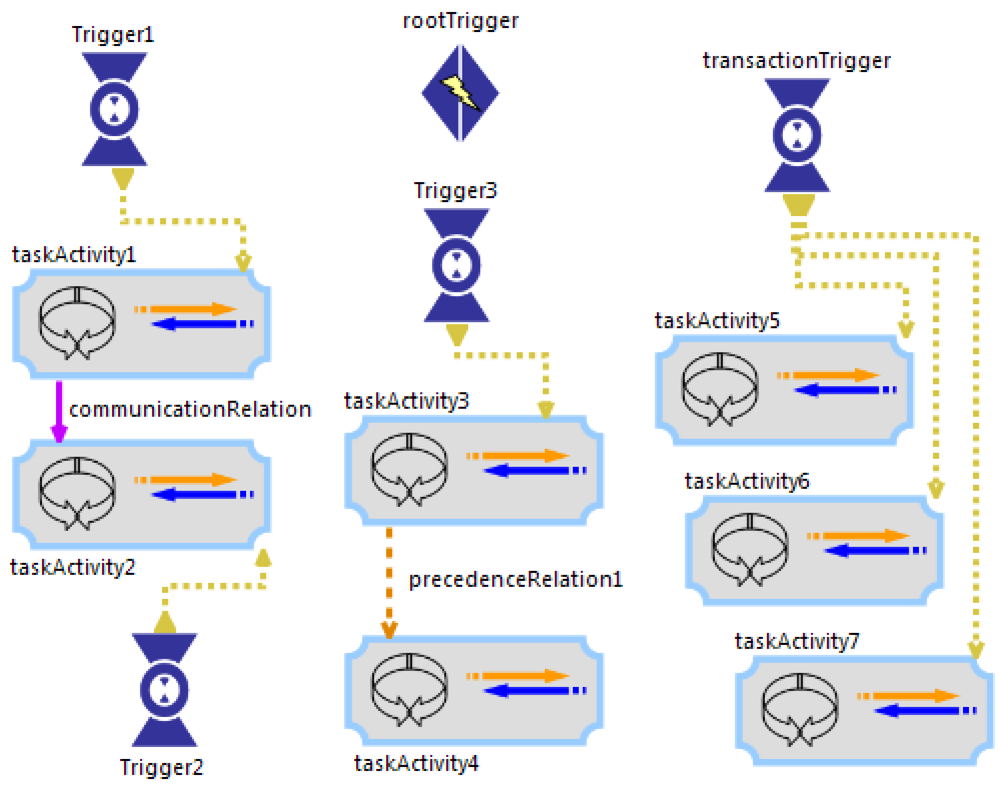


Figure 6.22: Example of a behavioral model in MoSaRT language

Figure 6.23 shows an example of the content of a task activity existing in the behavioral model. The content is modeled as a set of different kinds of step linked to each other via the step precedence relationships. The link between step elements and the functional model is not mandatory. However, that can help designers to have an accurate model for an advanced analysis. Thus, the importance of step elements is also in their capabilities to allocate the operational layer (the hardware, software architecture and behavioral models) to the applicative layer (the functional model).

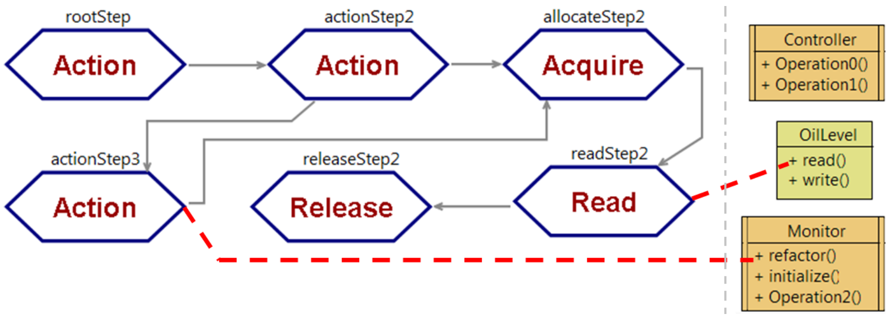


Figure 6.23: Part of a detailed behavioral model showing the content of a task activity and its mapping to a part of the functional model represented by the UML class diagram

Appendix A contains the behavioral parts of some famous task models expressed in the MoSaRT design language.

V. MoSaRT structural rules

V.1. Objectives

The MoSaRT language is dedicated to be used by different actors (designers and analysts) who are not necessarily proficient in different fields. Therefore, MoSaRT language is enriched by several structural rules with different severity types. MoSaRT design language encapsulates these rules and generates just the error or information message which can be understood easily. For the formalization of the structural rules, we have opted for OCL (Object Constraint Language) [OMG06] for two reasons. The first one is the clarity of OCL, which is a formal language conceived to be read and written easily. The second reason is related to the meta-metamodel Ecore, used for the abstract syntax of the MoSaRT design language (i.e. the description of MoSaRT language's metamodel). Ecore language operates very well with OCL and allows to define the constraints, the operations and the derived properties (see the first section of Appendix B). Thus, to get a coherent system design, the metamodel of the MoSaRT language has been enriched by three kinds of structural rules: **architectural**, **safety** and **vivacity** rules[1], that are described hereafter.
Every rule is an invariant related to a specific concept (i.e. class), and it must be respected by the instances of that concept throughout the modeling phase.

[1]The safety and vivacity terms are not the ones we find in model checking

V.2. Architectural rules

The architectural rules are constraints ensuring a good utilization of all the elements used to obtain the platform architecture. For example, to design a hierarchical scheduler, the following rule must be respected: *a scheduler cannot inherit from a scheduler B if B is inheriting from A or if B inherits from a scheduler which inherits from A*. Figure 6.24 shows a violated architectural rule. This violation is due to the inheritance relationship between *ProcessA* and *ProcessD*. The interpretation of the hierarchic scheduling rule corresponds to Listing 6.2.

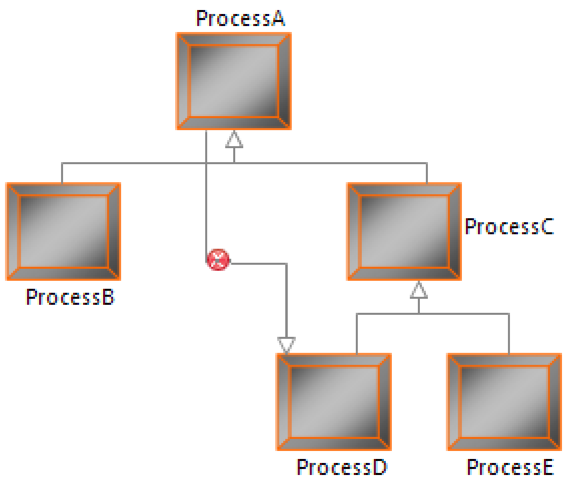


Figure 6.24: A part of the software model: an architectural rule is violated

Listing 6.2: An architectural rule expressed in OCL

```
invariant SoSpaceProcessRule2:
SoSpaceProcess.allInstances()
->forAll(sp1,sp2 | sp1.spaceProcessChildren->includes(sp2)
implies sp1.allSpaceProcessParents->excludes(sp2));
```

V.3. Safety rules

The safety rules guarantee that no erroneous behavior will happen. Figure 6.25 shows a case where a safety rule is violated. In this case, a “release step” is preceded by itself. A release step means that a task releases a semaphore after the end of the access to a critical shared resource. The Listing 6.3 shows the corresponding OCL rule.

Listing 6.3: A MoSaRT safety rule expressed in OCL

```
invariant SbStepPrecedenceRelationRule2:
self.sourceStep.oclIsTypeOf(SbReleaseStep)
implies
not self.targetStep.oclIsTypeOf(SbReleaseStep);
```

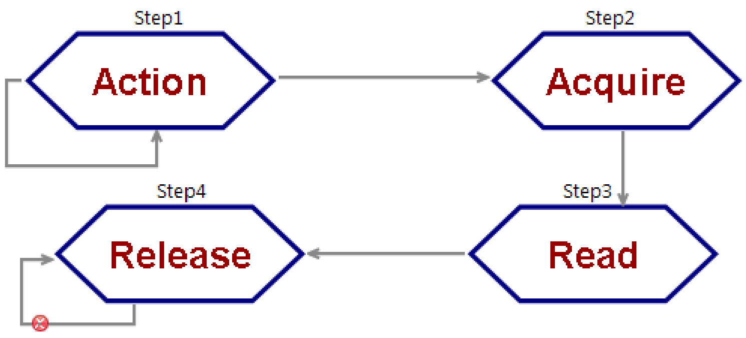


Figure 6.25: A part of the behavioral model [step view]: a safety rule is violated

V.4. Vivacity rules

The vivacity rules ensure the correctness and the completeness of the global behavior. This kind of rules is very important especially for behavioral model. For example (see Figure 6.26), in the behavioral model of a real-time system designed with the MoSaRT language, a task activity must be triggered by a trigger, or be preceded by another task activity. This precedence relationship can be designed as a precedence synchronization or as communication relationship. The Listing 6.4 shows the OCL rule corresponding to our example.

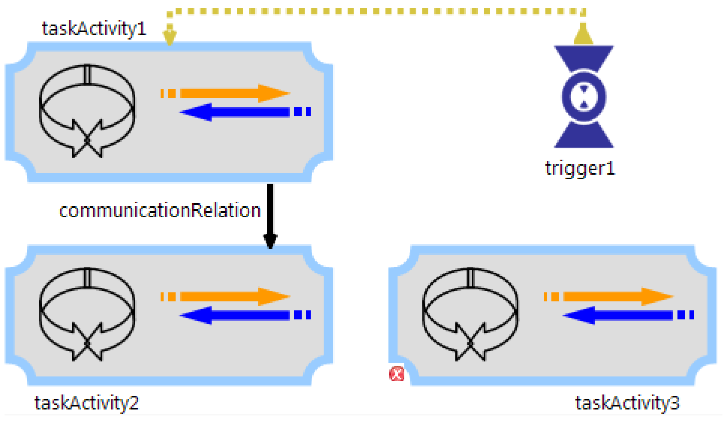


Figure 6.26: A part of the behavioral model: a vivacity rule is violated

Listing 6.4: A vivacity rule expressed in OCL

```
invariant SbTaskActivityRule1:
self.oclAsType(SbSchedulingActivity).trigger->isEmpty()
implies
self.oclAsType(SbSchedulingActivity).inputSquencingRelation->notEmpty();
```

V.5. Discussion

Structural rules must be respected during the modeling phase. By using the MoSaRT design language, the structure verification process of a designed system is incremental. This approach has two advantages. The first one is to cope with scaling problems. So, even with very complex systems the structure verification can succeed, thanks to the process which stops at the first violated rule. The second advantage is to maintain the traceability through the structure verification process, then the designer can be informed about the model element that causes the invalidation of the system. While the MoSaRT design language contains a large number of real-time properties, the structural rules allow also designers to know the necessity or the optionality of a property depending on the other elements existing in the design.

VI. Conclusion

To summarize the main features of the MoSaRT design language, Table6.4 clarifies the relations between various packages, the interaction of models and the views offered by combining them.

Global System		Packages					Views		
Layers	Models	Real-Time Properties	Hardware Platform	Software Platform		Functional Elements			
				S. Operators	S. Behavior				
Operational	Hardware	X	X				Platform Architecture		
	Soft. Architect.	X		X				Software Behavioral	
	Behavioral	X		X	X	#			Detailed Behavioral
Applicative	Functional					X			

Table 6.4: Different models, packages, and views of MoSaRT design language. “X” means mandatory utilization and “#” means optional utilization.

Figure 6.27 summarizes the global view of packages composing MoSaRT design language, their interdependence relationships, the different kinds of models and the structural rules.
Technical solutions which are used to implement the MoSaRT design language are presented in Appendix B.

Towards an orientating advisor for scheduling analysis Once all structural rules are met, the system designed through MoSaRT language is called structurally valid. Hence, the model forming the global system is analyzable (i.e. the model is ready for applying scheduling analysis tests), but it is not yet valid in a timing point of view. Thus, the utilization of MoSaRT language only is insufficient. It requires an orientating advisor to choose the right analysis tests. So, the next chapter provides our solution which consists on helping designers for choosing the suitable analyses (validation or dimensioning) depending on the model completion phase.

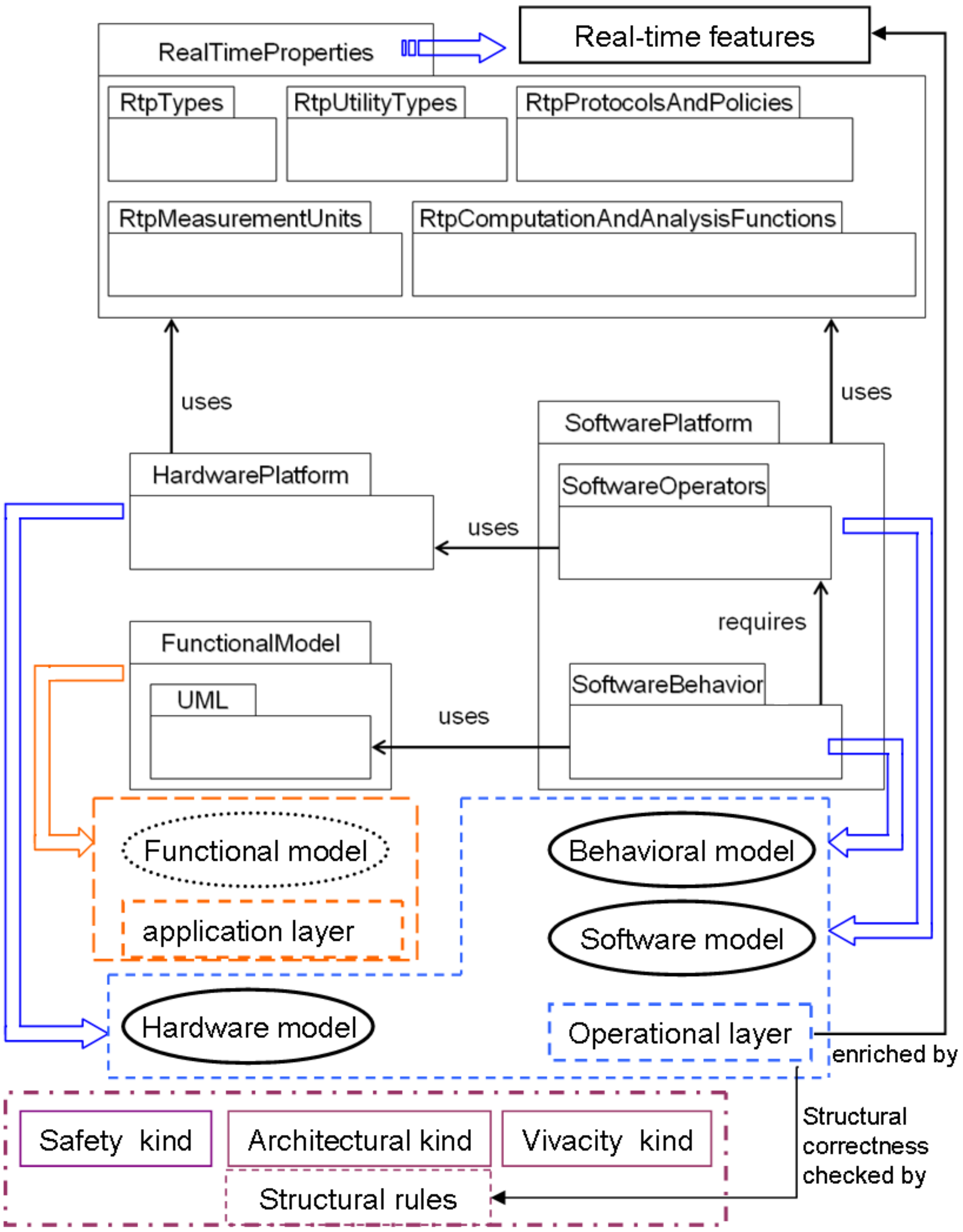


Figure 6.27: A global view of the MoSaRT language structure

Chapter 7

MoSaRT Analysis Repository

Contents

Abstract

The scheduling analysis is one of the main analyses required to ensure the timing correctness of a real-time system. Due to the criticality of real-time systems, the choice of the appropriate validation tests and/or dimensioning techniques has a relevant impact on the development life-cycle. Indeed, we highlight different scheduling characteristics helping designers and easing the use of the real-time scheduling theory. Then, we propose an approach based on the analysis repository to improve the way designers check their system designs.

This chapter and Chapter 3 are strongly coupled. So, we invite readers to glance at Chapter 3 in order to well understand the objective of the contribution discussed in this chapter.

I. Introduction

This chapter highlights different scheduling characteristics helping designers and easing the use of real-time scheduling theory. It presents an approach based on the analysis repository to improve the way designers check their system designs, and to increase the applicability of real-time scheduling theory.

The remainder of the chapter is organized as follows. Section II is devoted to present elements on which our contribution is based. Section III describes elements composing the metamodel called “MoSaRT analysis repository”. Section IV shows the instantiation of the MoSaRT analysis repository and discusses its capabilities and drawbacks. Section V summaries and concludes this chapter.

II. Taxonomy and some notions of the scheduling theory

In Chapter 3 we have presented several notions related to the real-time scheduling theory. Several task models, hardware resources, scheduling algorithms have been presented and every combination of those elements leads to a particular real-time situation. This latter is called “a real-time context” in our contribution. Moreover, in this section we present some notions we deem very important in order to increase the applicability of the scheduling theory. We introduce the real-time contexts which represent the cornerstone of the scheduling theory. Then, we expose an overview of context relationships. Accordingly, we present the analysis tests and their relevant characteristics.

II.1. Real-time contexts

Several research efforts have proposed analytical models with more expressiveness in terms of task activations, synchronization, communication and hardware resources, and many of them lead to efficient analyses. To choose one or many analytical models corresponding to a design model for analysis purposes, the real-time context of the design model must be identified. The real-time context represents the timing information extracted from a design model. This information is related to the execution requirements, the tasks behavior, the communication protocols, etc.

Real-time context. Every possible combination of a software model and a hardware architecture corresponds to a real-time context χ. The real-time context χ is a set of specific assumptions, where each assumption ε may be related to the software architecture, the timing behavior or the hardware architecture.

Let $\mathcal{X} = \{\chi_1, \chi_2, ..,\chi_n\}$ be a set of real-time contexts. Then, every real-time context $\chi_i = \{\varepsilon_1, \varepsilon_2, ..., \varepsilon_n\}$ is a set of assumptions. Each ε_i can be characterized by properties related to an analysis performance. So, an assumption may complicate worsening the analysis results, improve it or be indifferent. For instance, in a uniprocessor architecture, while the precedence relationships, the existence of shared resources, and the self-suspension are among factors worsening the timing behavior of the system, the existence of the release times improves the timing behavior (i.e. the critical instant can be avoided).

Example. Every abstract model respecting the following assumptions corresponds to the real-time context of Liu and Layland periodic model:

- ε_1 = "tasks are periodic";
- ε_2 = "deadlines are implicit";
- ε_3 = "simultaneous task set";
- ε_4 = "tasks are independent";
- ε_5 = "for each task, the worst-case execution time is less than or equal than the period value";
- ε_6 = "there is no self-suspension";
- ε_7 = "each task is preemptible";
- ε_8 = "priorities are fixed";
- ε_9 = "uniprocessor architecture".

II.2. Generalization relationships

Over the years, substantial generalizations of basic periodic task-set model have been considered. We have surveyed some of these generalization relationships in Section II.2 of Chapter 3. However, the generalization notion presented below is extended to support the generalization relationships among real-time contexts.

Generalization relationship. The real-time context χ_A is a generalization of the real-time context χ_B, if the behavior of χ_A includes all possible behaviors of χ_B.

Worst-case generalization relationship. The real-time context χ_A is a worst-case generalization of the real-time context χ_B, if the behavior of χ_A can capture the worst-case behavior of χ_B.

Best-case generalization relationship. The real-time context χ_A is a best-case generalization of the real-time context χ_B, if the behavior of χ_A can capture the best-case behavior of χ_B.

Let $\mathcal{G}$ = {g_1, g_2, ..,g_n} be a set of generalization relationships between some real-time contexts. We note for every g_z = <χ_i, χ_j, B_k> where the real-time context χ_i is a generalization of χ_j according to the generalization behavior kind B_k, and $B_k \subseteq$ {worst-case, best-case}.

Example. In the motivating example presented in Chapter 5 and illustrated by Figure 5.4, the transaction model (Part 1 of Figure 5.4) represents a generalization of the sporadic model (Part 3 of Figure 5.4). So, in this case B_k = {worst-case, best-case}.
The transaction model [RGRR12] is also a worst-case generalization of the generalized multiframe model [BCGM99]. In this case, B_k = {worst-case}.

II.3. Scheduling Analysis Tests

Two main purposes are sought through the analysis. The first one is to evaluate the scheduling correctness of an abstract model. In this case, the analysis test is applied to validate a complete model. The second purpose of the analysis is to help designers to dimension their models at an early phase. In this case, models are not completely defined, and designers need to be helped, for instance to know the feasibility domain for some parameters, the task allocation, the processor partition, etc. Thus, referring to the goal desired by applying an analysis, two categories of tests are proposed: validation category and dimensioning category. Therefore, the choice of the tests category depends on the model completion.

Correctness of the analysis tests. Every analysis test t_i is based on a real-time context. When applied to a system, the result provided by the test t_i is correct if the system respects all the context assumptions.

Validation tests category. The category of validation tests denoted C_v can be solicited when the whole model parameters are provided. The validation tests category contains analysis tests checking that a given application can satisfy its specified deadlines when scheduled according to a specific scheduling algorithm.

Dimensioning tests category. The dimensioning tests category C_d provides a set of techniques in order to help designers to estimate some feasible regions of the timing characteristics, or to rectify some properties in order to have a feasible system.

The set of schedulability analysis tests which are compatible with a designed system will be denoted $\mathcal{T} = \{t_1, t_2, .., t_n\} = C_v \cup C_d$ such that:
if t_i is a validation test, then $t_i \in C_v$.
if t_i is a dimensioning test, then $t_i \in C_d$.

II.3.a. Test Characteristics

A validation test may be considered by a set of characteristics related to the feasibility and the sustainability. While a feasibility characteristic permits to conclude about the accuracy of the test referring to the applicant system, the sustainability characteristic enables to assess the reliability of the feasibility although some timing properties vary.

Let χ_i be a real-time context compatible with a validation test t_j. Hence, referring to the applicant real-time context χ_i, the schedulability test t_j is characterized by the feasibility characteristic Ch_f. Where $Ch_f \subseteq$ {sufficient condition, necessary condition}.

Example. χ_i is a real-time context of a set of n independent, preemptible and sporadic tasks on uniprocessor, such that their relative deadlines are constrained and the deadline monotonic is the scheduling policy. t_k is a validation test representing the response time analysis of χ_i's task set. In this context the test t_k is an exact (sufficient and necessary) condition. However, the same test t_k is a sufficient condition for the context χ_j, such that, in addition of χ_i's assumptions the real-time context χ_j is characterized by an asynchronous task set (it means that tasks do not have the same release times). Hence the critical instant may be not observed during the system life.

A schedulability test is sustainable if the applicant system is still deemed schedulable when decreasing the execution-time, delaying arrival times, reducing jitter, or increasing relative deadlines [BB08] (see Section III.2 of Chapter 3). When change is only related to:

- the decrease of the execution-times, then the test ensures the C-sustainability (also known as predictability).
- the decrease of the jitters, then the test ensures the J-sustainability.
- the increase of the periods, then the test ensures the T-sustainability.
- the increase of the deadlines, then the test ensures the D-sustainability.

Let χ_i be a real-time context applying a validation test t_j. Hence, referring to the applicant real-time context χ_i, the schedulability test t_j is characterized by the sustainability characteristic Ch_s, where $Ch_s \subseteq$ {C-sustainability, J-sustainability, T-sustainability, D-sustainability}.

Example. Let t_z be a processor utilization test of RMS (Rate Monotonic System). Thus, t_z represented by $U = \sum_{i=1}^{n} \frac{Ci}{Pi} \leq n(2^{1/n} - 1)$ (where n is number of tasks, Ci is the worst-case execution time, and Pi is the period) is sustainable with respect to execution requirements, deadlines and periods for periodic fixed-priority tasks with implicit deadlines. Then, the sustainability characteristic of t_z is Ch_s = {C-sustainability, T-sustainability}.

II.3.b. Test Family

Test family. A test family f_i is a set of scheduling analysis tests dedicated to the same objective. It means that the test family is an aggregate of an analysis test and its different alternatives and derivations according to different real-time contexts.

Let $\mathcal{F} = \{f_1, f_2, .., f_n\}$ be a set of test families, such that, every $f_i = \{t_1, t_2, .., t_n\}$ is a set of analysis tests. For every t_i and t_j which are analysis tests of real-time contexts χ_i and χ_j respectively, t_i is a derivation of t_j and $\chi_i \neq \chi_j$.

Example. The processor utilization-based analysis is a test family f_a containing several tests related to the processor factor utilization U. Then, $f_a = \{t_1, t_2\}$ means :
• Test t_1 depicts $U = \sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$, such that n is the number of tasks, Ci is the worst-case execution time of each task τ_i, and Pi is the period of each task τ_i.
• Test t_2 is an alternative of t_1, where t_2 represents $U = \sum_{i=1}^{n} \frac{Ci}{Pi} \leq n(2^{1/n} - 1)$.

II.3.c. Analysis Tools

Originally, the scheduling analysis has been done manually which is error-prone and tedious. Recently, scheduling analysis tools using systematic approaches to generate equations have been developed. Analysis tests are scattered in different analysis tools. Moreover, each tool is based on its specific model. As our objective is not to propose a new analysis tool, we aim to take advantages of the existing ones. Indeed, we intend to help analysts to integrate their prototypes/tools. We give a special attention to classify analysis tools capabilities. That reduces the risk of getting incoherent abstract model due to an incorrect translation form modeling framework to input analysis tool formalisms. If such incoherence exists, it may lead to wrong results or at least to a pessimistic result due to several issues (already presented in Chapter 5). Hence, every analysis tool functionality should be affected to a specific schedulability test related to a specific real-time context. In [AGAT10], authors present a comparative study highlighting differences and similarities between MAST [MPHD01] and Cheddar [SLNM04] analysis frameworks. The meaning of the analysis tool in our context is not the whole tool but each one of its capabilities.

Analysis tool. The analysis tool e_i is an engine proposing an independent functionality inside an analysis framework. The same analysis functionality inside another analysis framework is considered as another engine e_j.

Let $E = \{\mathrm{e}_1, \mathrm{e}_2, .., \mathrm{e}_n\}$ be a set of engines implementing various analysis tests, such that the equation performing a schedulability test t_j can be provided by several tools E_{t_j}, such that
$E_{t_j} \subseteq E$. The designer can compare the analysis results obtained with different implementations of a test.

Example. In case of a real-time context corresponding to the time-offsets based system [TC94], the response time analysis test in one of the tests that can analyze this kind of system. Hence, designers should be oriented to choose the appropriate response time analysis provided by the offset analysis tool [oY94].

II.4. Discussion: towards an analysis-aided support for design decisions

Nowadays, we have noticed the absence of an instrumented method guiding the designers to the best model and tests for their systems. Moreover, the passage from the system modeling to the system analysis requires dual skills, in order:

- to identify the appropriate real-time context from the system design;
- to find the analysis tests that are suitable to give a reasonable result referring to the real-time context of the system; and
- to transform the design model to another formalism that is understandable by the analysis tools supporting the analysis tests.

The next section presents our approach called "MoSaRT Analysis Repository", and exposes the conception of the contribution and some relevant capabilities. The MoSaRT Analysis Repository is dedicated to be instantiated by schedulability-aware theorists/analysts in order to obtain analysis decision supports. **The designer's orientation is based on the richness and the correctness of the analysis decision support.** This latter may be fed and enriched by theorists/analysts (i) to compare their results with existing ones (already stored in the decision support), (ii) and to facilitate the use of their works (and their prototypes) by designers.

III. Model-driven analysis through the MoSaRT Analysis Repository

As we had overcome in Section II.4, the existence of a decision support playing the role of an advisor is very needful. We propose an analysis repository, which is a reorganization of concepts tackled in Section II. The analysis repository Ar consists of a set of real-time contexts $\mathcal{X}$, a set of generalization relationships $\mathcal{G}$, a set of analysis tests $\mathcal{T}$, a set of test families $\mathcal{F}$ and a set of analysis tools E.
Nonetheless, it is common to find several real-time context characterized by the same subset of assumptions, or the opposite subset of assumptions. So, for factoring the number of assumptions and to guarantee a good scalability of the analysis repository, we suggest that the analysis repository contains also a set of identification rules $\mathcal{R}$. They will help for identifying correctly the closest real-time context matching the design model which requests analysis. Then we note, $Ar = <\mathcal{R}, \mathcal{X}, \mathcal{G}, \mathcal{T}, \mathcal{F}, E>$.

III.1. Meta-model of the MoSaRT Analysis Repository: Conception and Semantics

To ease the use of the analysis techniques and to guarantee the success of the analysis process, we take advantage of model driven engineering by presenting an open analysis repository model dedicated to real-time analysts.

III.1.a. Identification rules: $\mathcal{R}$

Let $\mathcal{R} = \{r_1, r_2, .., r_n\}$ be a set of identification rules, where the evaluation result of every rule r_i represents an assumption $\epsilon_i \in \{\chi_1 \cup \chi_2 \cup ... \cup \chi_n\}$.
For instance, r_z is a rule which checks if "no task may voluntarily suspend itself?". So, the evaluation of r_z corresponds to the assumption ϵ_x when the evaluation result is false, and it corresponds to the assumption ϵ_y when the evaluation result is true. Knowing that, $\epsilon_x \in \chi_x$ and $\epsilon_y \in \chi_y$, χ_x is a real-time context corresponding to a model with self-suspension, and χ_y is a real-time context corresponding to a model without any self-suspension.
We note: $\chi_x \models \overline{r_z}$ (i.e. r_z is not satisfied by χ_x) and $\chi_y \models r_z$ (r_z is satisfied by χ_y).

Let $\mathcal{S}_r$ be a function for specifying real-time contexts in the analysis repository.
$\mathcal{S}_r$: $\mathcal{X} \rightarrow \mathcal{R}^{card(\mathcal{R})}$ where, $\mathcal{S}_r(\chi_i) = \mathcal{R}_i^{true} \cup \mathcal{R}_i^{false} \cup \mathcal{R}_i^{undef} = \mathcal{R}$, such that:
$\mathcal{R}_i^{true}$ is a set of rules which have to be satisfied, for every $r_a \in \mathcal{R}_i^{true}$, $\chi_i \models r_a$;
$\mathcal{R}_i^{false}$ is a set of rules whose complements have to be satisfied, for every $r_b \in \mathcal{R}_i^{false}$, $\chi_i \models \overline{r_b}$;
$\mathcal{R}_i^{undef} = \mathcal{R} \setminus (\mathcal{R}_i^{true} \cup \mathcal{R}_i^{false})$.

Figure 7.1 shows the `IdentificationRule` class (expressed in Ecore [SBPM08]). Every `IdentificationRule` instance is characterized by an `id` and a `timingKind` as an element of the `TimingKindType` enumeration. The `timingKind` property indicates (when the evaluation result is true) if the rule complicates or not the scheduling. For instance, "the existence of concrete tasks (i.e. release-times are known)" reduces the scheduling difficulty, hence in this case `timingKind` = "*improving*". By default, the `timingKind` equals to "*indifferent*". Every `IdentificationRule` instance is also characterized by the `componentKind` property, that corresponds to the nature of the checked element. This latter can be hardware (e.g. checking if the architecture is uniprocessor) or software (e.g. checking if tasks are preemptible). The `description` property describes (using a native language) the meaning and goal of the rule evaluation.

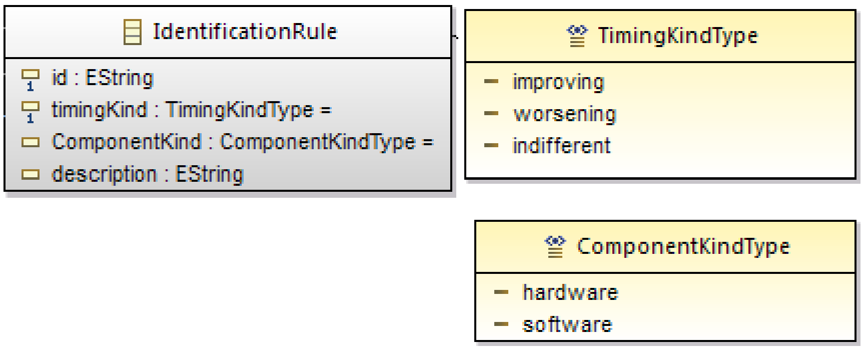


Figure 7.1: `IdentificationRule` class and its properties

III.1.b. Real-time contexts: $\mathcal{X}$

As it is shown in Figure 7.2 every real-time context is an instance of the `ContextModel` class. It is specified by a set of identification rules (instances of `IdentificationRule` class) following the specification relation $\mathcal{S}_r$ already described. Each `ContextModel` instance is characterized by a unique `name`, an optional `description` explaining the real-time context in native language, and `references` property showing details of research papers which had introduced an analytical model according to the real-time context.

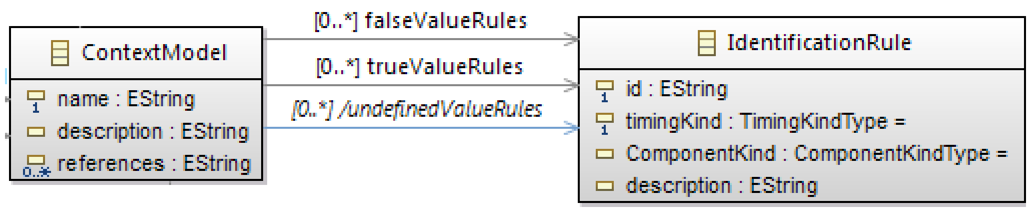


Figure 7.2: ContextModel class, its interactions and its properties

III.1.c. Generalization relationships: $\mathcal{G}$

Every generalization relationship between two real-time contexts can be instantiated by using the `Generalizationrelationship` class (see Figure 7.3). While every generalization-relation is identified by a `name` and a `description`, it is also possible to precise the `behaviorKind` property. This later is an enumeration related to the `BehaviorKindType`.
The `Generalizationrelationship` concept also permits during its instantiation to precise how one can transform a specific context to a generic one and vice-versa. This information can be stored in an external file which its location can be mentioned through the `genericToSpecificPath` and the `specificToGenericPath` properties.

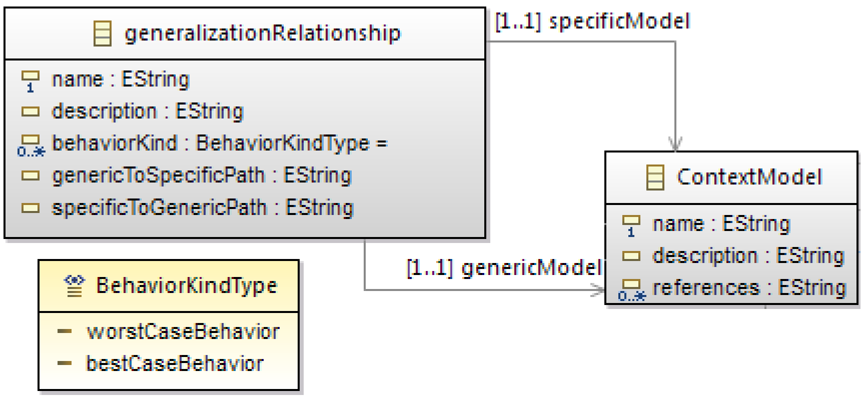


Figure 7.3: GeneralizationRelationship class, its interactions and its properties

III.1.d. Analysis tests: $\mathcal{T}$

A real-time context is interesting only if it can be analyzed. So, every instance of `ContextModel` can be analyzed by a set of analysis tests (see Figure 7.4). Every test is an instance of `Test` class, and it has a set of characteristics: `FeasibiltyType` and `SustainabilityType` characteristics. Indeed, each test characteristic (instance of `TestCharacteristicType` abstract class) is related to a real-time context. Then, a test can have different characteristic depending on the applicant context. As other concepts, the properties of each `Test` instance enable to distinguish the test (via `id`, `name` and `description`) and to mention the research studies, where the test was proposed.

III.1.e. Analysis test families: $\mathcal{F}$

Figure 7.5 illustrates that every `Test` instance can be linked to a `TestFamily` instance. This latter contains several `Test` instances sharing the same specificities.

III.1.f. Analysis tools: E

It will be very interesting to perform an analysis test implemented by different tools (see Figure 7.6). That enables to compare different results related to the same test after being applied to a specific

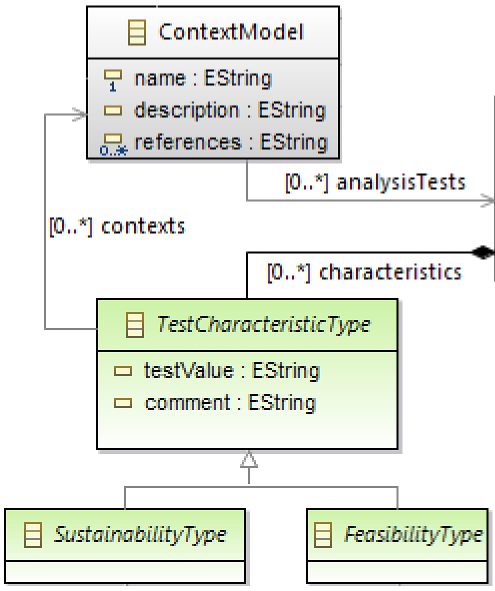


Figure 7.4: Test class, its interactions and its properties

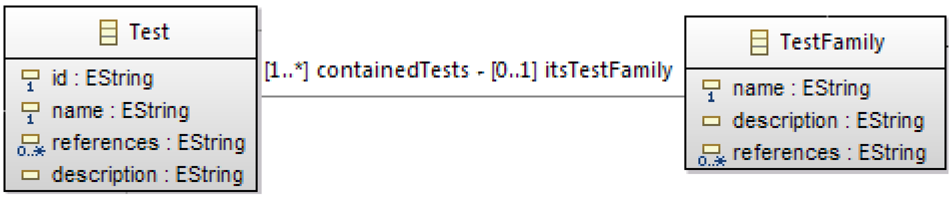


Figure 7.5: TestFamily class, its interactions and its properties

context. For this purpose, besides `id`, `name` and `description` properties, the `Tool` class provides the `MosartToToolPath` property to mention the location of external files (like an executable program, a web service, etc.) enabling the transformation of the system design (corresponding to a real-time context on which the test application is correct) to the input formalism of the analysis tool. The `ToolToMosartPath` property may contains the location of a program ensuring the restitution after analysis. By the restitution, we mean the transformation from the output tool formalism to the design model formalism.

III.1.g. Analysis Repositories: a set of *Ar*

Every *Ar* can be represented by an instance of the `AnalysisRepository` class (see Figure 7.7). Every instance of the `AnalysisRepository` class is the root element allowing to specify other analysis repository elements (contexts, tools, tests, etc.). Then, every organism/company/laboratory can have its own analysis repository instances to orient their designers referring to the information proposed by every used analysis repository. The `name` and the `description` properties permit to give detailed information about each established repository.

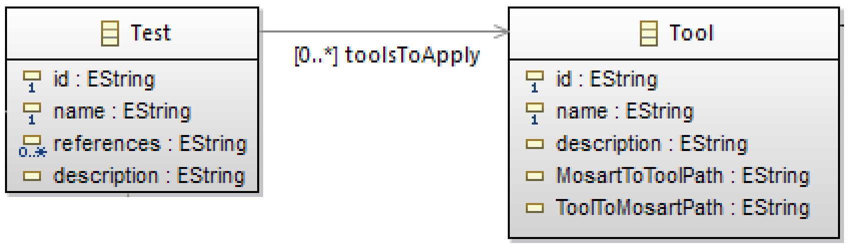


Figure 7.6: Tool class, its interactions and its properties

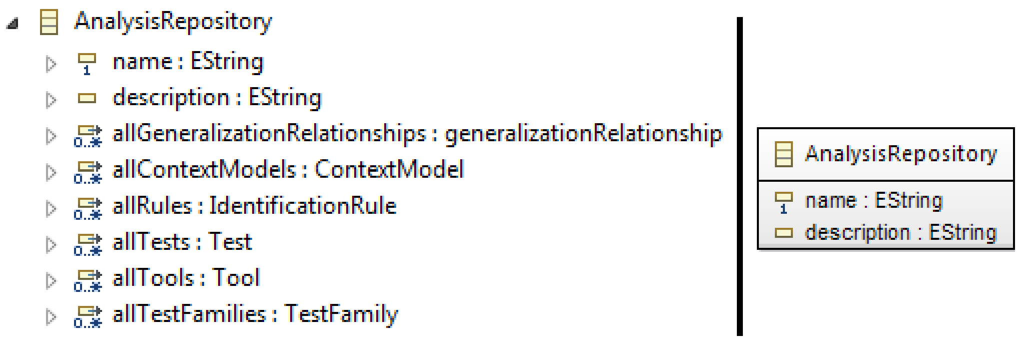

AnalysisRepository
name : EString
description : EString
allGeneralizationRelationships : generalizationRelationship
allContextModels : ContextModel
allRules : IdentificationRule
allTests : Test
allTools : Tool
allTestFamilies : TestFamily
AnalysisRepository
name : EString
description : EString

Figure 7.7: Two representations of the AnalysisRepository class and its properties. The left representation shows the composition relationship between the analysis repository class (root container) and the other remaining classes of the analysis repository meta-model (containing)

III.2. Identification process

III.2.a. Analysis Repository - Design languages Interdependency

To assist designers to conclude about the schedulability of their designs, an analysis repository has to be invoked by a valid system design. Consequently, the meta-model of the analysis repository has to be linked to a specific design language. This relation enables the analysis repositories (instances of the MoSaRT Analysis Repository) to be able to detect the appropriate contexts starting from any system modeled in accordance with the specific design language.

III.2.b. The identification process of the real-time contexts

Figure 7.8 gives an overview of the identification process of the real-time contexts. The identification process starts when a valid model calls the identification rules in order to be checked (step (1) of Figure 7.8). During the second step (see step(2) of Figure 7.8), every identification rule is checked in order to get the evaluation result. For instance and as it is shown in Figure 7.8, $\surd$ means that the evaluation result of the rule is true (the rule is satisfied by the model), ⊠ symbol means false (the rule is not satisfied by the model). Consequently, the evaluation result of the rule-set corresponds to a certain real-time context. We note that for every instance of the MoSaRT analysis repository, we consider that the number of rules characterizing each context equals to the number of identification rules (cardinal of $\mathcal{R}$). It happens that a context model stored in the analysis repository does not use

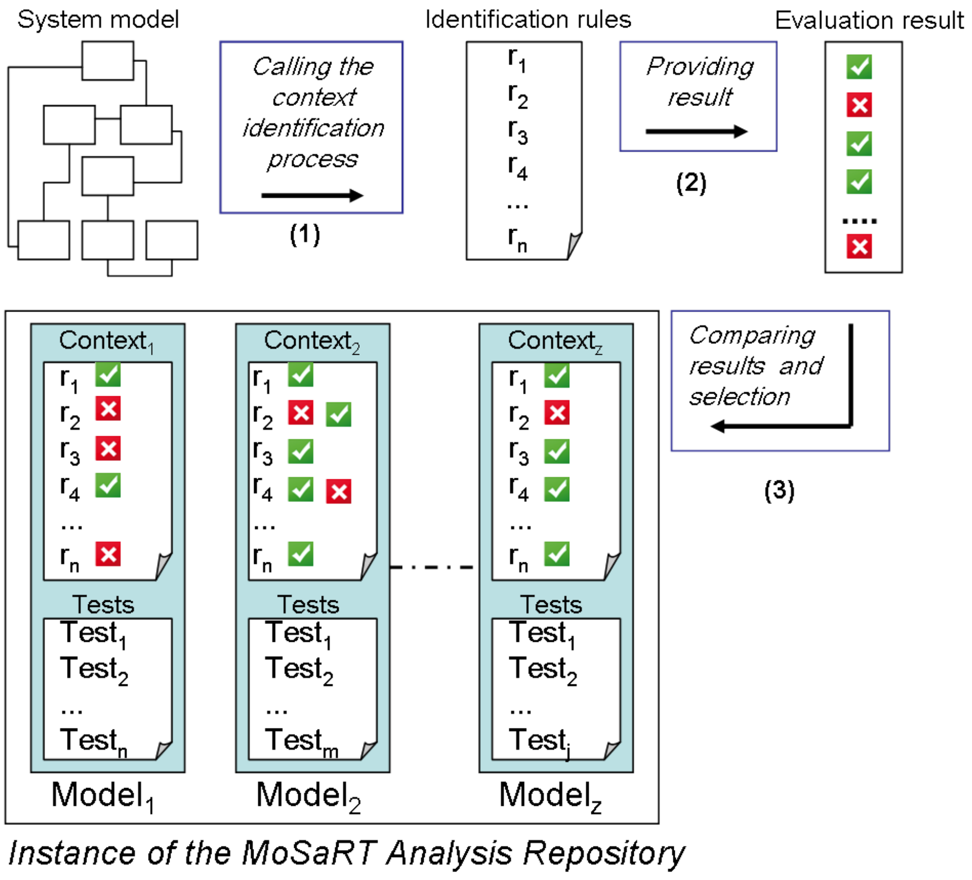


Figure 7.8: Different steps of the identification process

some of the evaluation results. For example, if the real-time context takes into consideration a rule about "arbitrary deadline", then this context does not need the evaluation of "implicit deadline " rule. For instance, the real-time context of "Model2" (see Figure 7.8) does not need r_2 (thence the use of $\surd$ ⊠ together as a symbol of an undefined rule evaluation). The third step consists on comparing the evaluation result to different real-time contexts existing in the analysis repository.
The relevant content of the different steps are presented through Algorithm 1. At the end of the identification process, a designer obtains a real-time context matching the system design, as well as the analysis tests, the scientific papers concerning the tests and also the analysis tools providing those tests. The input files for several analysis tools would be generated if any transformation program had been already integrated.

IV. Instantiation of the MoSaRT Analysis Repository

In this section, we surround some capabilities of the MoSaRT Analysis Repository by instantiating its elements and illustrate its flexibility as well as the simplicity of a modification or extension. The

Algorithm 1: Identification process

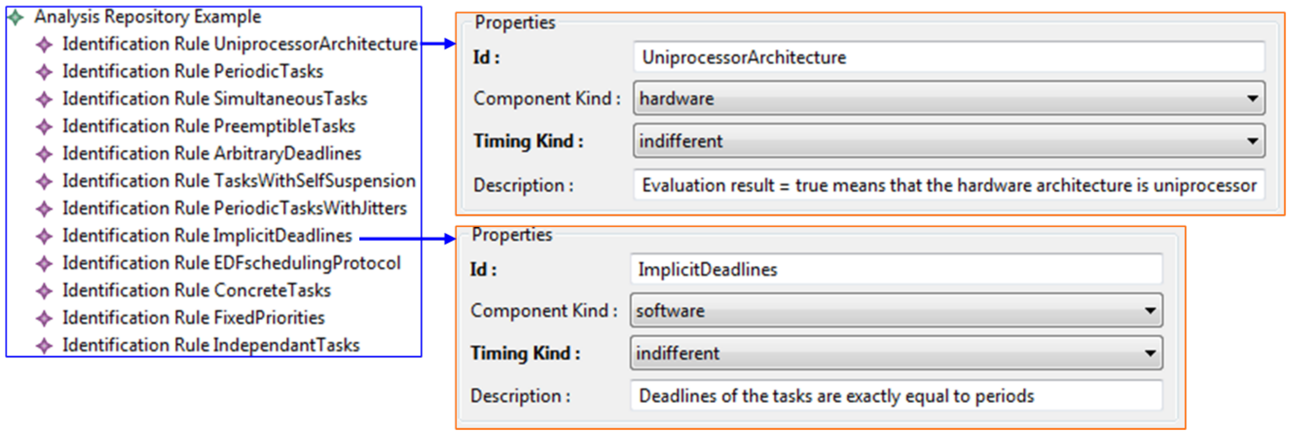
```
Inputs   : M: A valid analytical model of a real-time system;
           $Ar_k$: A specific analysis repository chosen by the designer;
Outputs: $\chi_M$: The appropriate real-time context;
Loading and evaluation of all identification rules $\mathcal{R}$ existed in $Ar_k$
Let $\mathcal{R}_M^{true} = \mathcal{R}_M^{false} = \mathcal{R}_M^{undef} = \{\}$;
for each $r_i \in \mathcal{R}$ do
    if $M \models r_i$ then
        $\mathcal{R}_M^{true} \leftarrow \mathcal{R}_M^{true} \cup \{r_i\}$ ;
    else
        if $M \models \overline{r_i}$ then
            $\mathcal{R}_M^{false} \leftarrow \mathcal{R}_M^{false} \cup \{r_i\}$ ;
        else
            $\mathcal{R}_M^{undef} \leftarrow \mathcal{R}_M^{undef} \cup \{r_i\}$ ;
Loading and comparing $\mathcal{R}_M^{true}$, $\mathcal{R}_M^{false}$ and $\mathcal{R}_M^{undef}$ to $\mathcal{X}$
for each $\chi_j \in \mathcal{X}$ do
    for each $r_i \in \mathcal{R}_M^{true}$ do
        if $\chi_j \models \overline{r_i}$ then
            break;
        if $i = cardinal(\mathcal{R}_M^{true})$ then
            $\chi_M \leftarrow \chi_j$ ;
    if $\chi_M \in \mathcal{X}$ then
        for each $r_k \in \mathcal{R}_M^{false}$ do
            if $\chi_M \models r_k$ then
                $\chi_M \leftarrow$ Null ;
                break;
    if $\chi_M \in \mathcal{X}$ then
        break;
return $\chi_M$ ;
```

Instantiation below is independent from any design language.

IV.1. Example of creation of a repository instance

In order to create a new repository instance from scratch, we will start by instantiating the `IdentificationRule` to get a set of identification rules. Next, we will instantiate the `ContextModel` to create a real-time context based on the existing identification rules. Furthermore, we will add to the repository instance some tests and analysis tools corresponding to the created real-time context.

IV.1.a. Instantiation of the identification rules



Analysis Repository Example
- Identification Rule UniprocessorArchitecture
- Identification Rule PeriodicTasks
- Identification Rule SimultaneousTasks
- Identification Rule PreemptibleTasks
- Identification Rule ArbitraryDeadlines
- Identification Rule TasksWithSelfSuspension
- Identification Rule PeriodicTasksWithJitters
- Identification Rule ImplicitDeadlines
- Identification Rule EDFschedulingProtocol
- Identification Rule ConcreteTasks
- Identification Rule FixedPriorities
- Identification Rule IndependantTasks

Properties

Id :	UniprocessorArchitecture
Component Kind :	hardware
Timing Kind :	indifferent
Description :	Evaluation result = true means that the hardware architecture is uniprocessor

Properties

Id :	ImplicitDeadlines
Component Kind :	software
Timing Kind :	indifferent
Description :	Deadlines of the tasks are exactly equal to periods

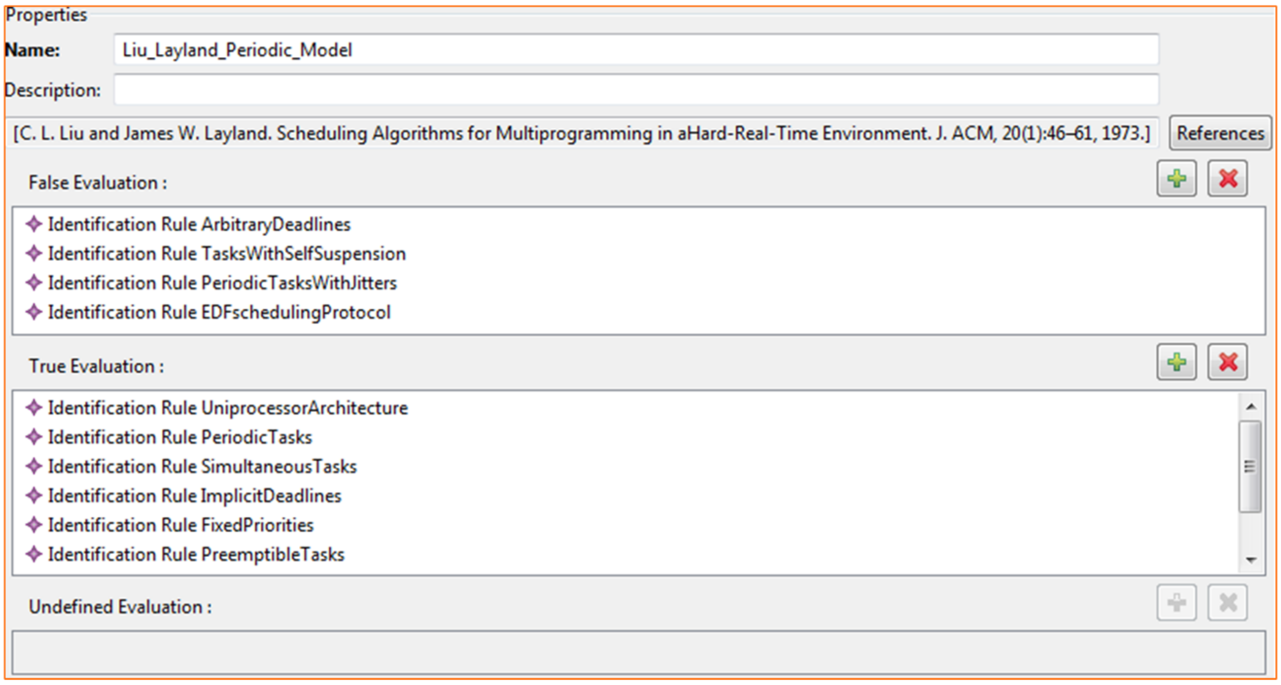
Figure 7.9: Identification rules and their properties

Figure 7.9 shows the identification rules that we have created in the repository. They corresponds to the following propositions:

- The hardware architecture is uniprocessor.
- The tasks are periodic.
- The release times of tasks are simultaneous.
- The tasks are preemptible.
- The deadlines are arbitrary.
- The tasks are self suspended.
- The periodic activations of tasks are subject to the jitters.
- The deadlines are implicit.
- The scheduling policy is EDF.
- The release times of tasks are concrete.

- The tasks priorities are fixed.
- The tasks are independent.

Every identification rule is specified through its properties in particular the description property. For instance the proposition “the hardware architecture is uniprocessor” is represented by the identification rule *UniprocessorArchitecture*. The description of this latter means that if the evaluation result is true, hence the hardware architecture is uniprocessor. Otherwise, the hardware architecture is not uniprocessor.


Properties

Name: Liu_Layland_Periodic_Model

Description:

[C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in aHard-Real-Time Environment. J. ACM, 20(1):46–61, 1973.] References

False Evaluation :

- Identification Rule ArbitraryDeadlines
- Identification Rule TasksWithSelfSuspension
- Identification Rule PeriodicTasksWithJitters
- Identification Rule EDFschedulingProtocol

True Evaluation :

- Identification Rule UniprocessorArchitecture
- Identification Rule PeriodicTasks
- Identification Rule SimultaneousTasks
- Identification Rule ImplicitDeadlines
- Identification Rule FixedPriorities
- Identification Rule PreemptibleTasks

Undefined Evaluation :

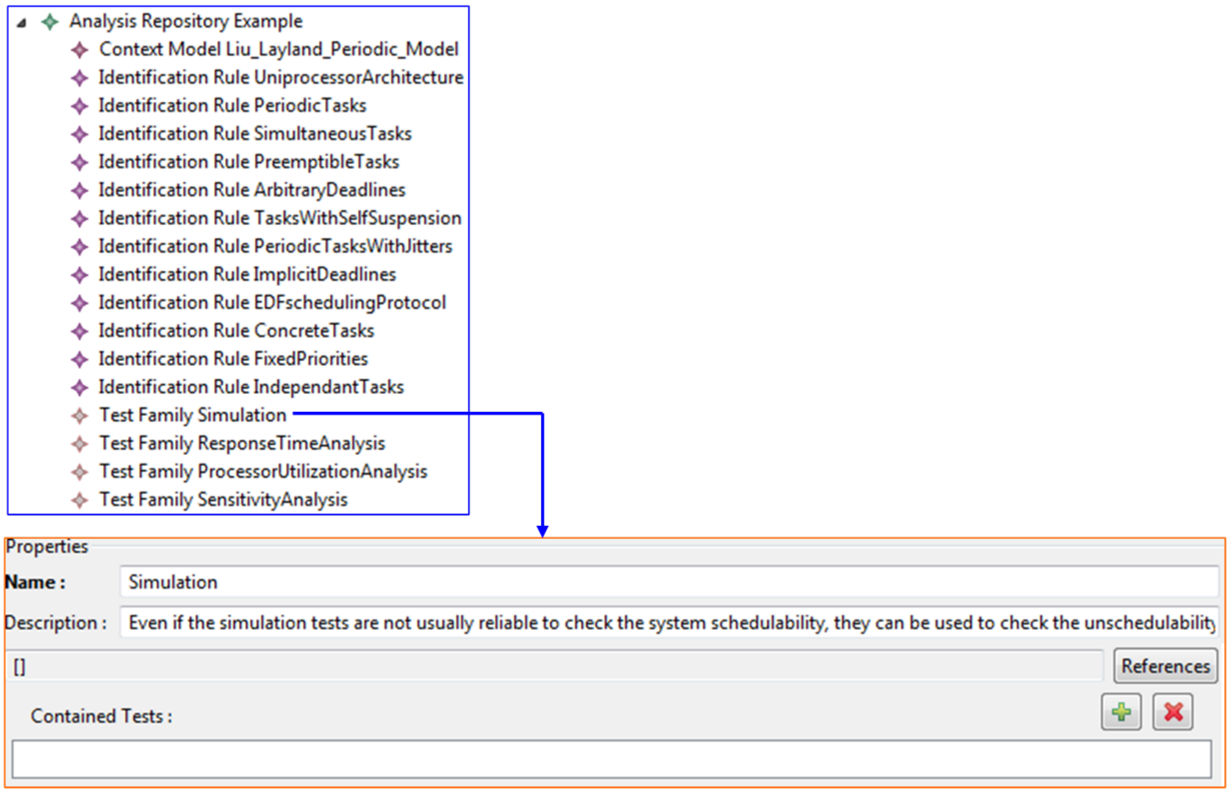
Figure 7.10: Liu & Layland real-time context and its properties

IV.1.b. Creation of the Liu & Layland real-time context

The creation of a real-time context corresponding to the Liu & Layland model [LL73] will be based on the identification rules already existing in the repository. Figure 7.10 illustrates the Liu & Layland real-time context. The “False Evaluation” part contains rules which must not be satisfied by a design model corresponding to the Liu & Layland context. The “True Evaluation” part contains rules which have to be satisfied by a design model corresponding to the Liu & Layland context. The “Undefined Evaluation” part contains rules which does not have any impact on the identification process. In other words, the evaluation result of every rule in the “Undefined Evaluation” part does not boost nor reduce the chance of choosing the real-time context as a context corresponding to the design model subjected to the identification process. “References” property contains details about research papers, which have discussed and presented the real-time context.

IV.1.c. Creation of the analysis tests

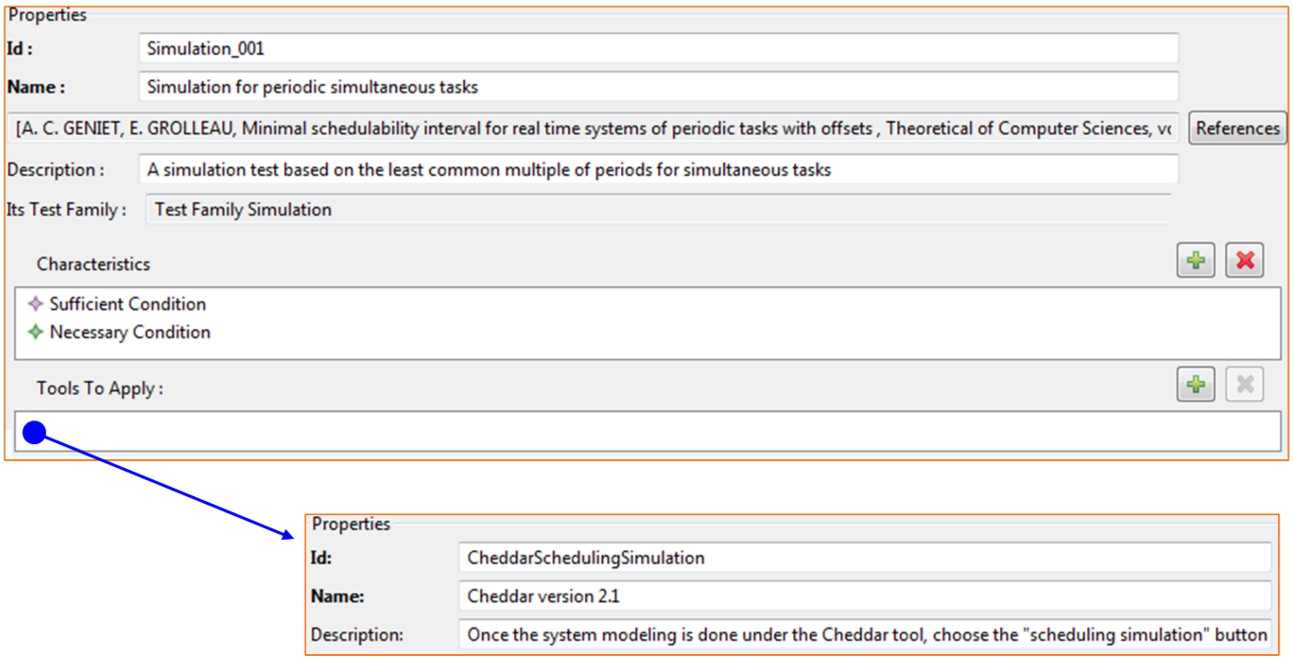
Once the Liu & Layland context is created, we create its corresponding analysis tests.
Test families We start by creating four test families: the simulation, the response time analysis, the processor utilization, and the sensitivity analysis. Every test family may be characterized by "Description" , "References" and "Contained Tests" properties.


Analysis Repository Example
Context Model Liu_Layland_Periodic_Model
Identification Rule UniprocessorArchitecture
Identification Rule PeriodicTasks
Identification Rule SimultaneousTasks
Identification Rule PreemptibleTasks
Identification Rule ArbitraryDeadlines
Identification Rule TasksWithSelfSuspension
Identification Rule PeriodicTasksWithJitters
Identification Rule ImplicitDeadlines
Identification Rule EDFschedulingProtocol
Identification Rule ConcreteTasks
Identification Rule FixedPriorities
Identification Rule IndependantTasks
Test Family Simulation
Test Family ResponseTimeAnalysis
Test Family ProcessorUtilizationAnalysis
Test Family SensitivityAnalysis

Properties
Name : Simulation
Description : Even if the simulation tests are not usually reliable to check the system schedulability, they can be used to check the unschedulability
[]

Figure 7.11: Simulation test family and its properties

Figure 7.11 illustrates the creation of four test families and shows properties of the *Simulation* test family.
Analysis test 1 Figure 7.12 depicts the creation of the analysis test *Simulation_ 001* as an exact test for the Liu & Layland context (see "Characteristics" property). This test belongs to the *Simulation* test family that is already created. Detailed information related to this test is presented via "Name", "References" and "Description" properties. Figure 7.12 shows also the creation of the analysis tool *CheddarSchedulingSimulation*. This latter represents a functionality provided by the Cheddar tool, which implements the analysis test *Simulation_ 001*.
Analysis test 2 Figure 7.13 evinces the instantiation of `Test` and `Tool` elements in order to create the analysis test *RTA_ 001* and its implementation *RtDruid_ analysis_ with_ Offset_ FP*. This latter is provided by the Rt-Druid tool. Referring to the Liu & Layland context existing in the repository under the name of *Liu_ Layland_ Periodic_ Model*, the analysis test *RTA_ 001* is exact and sustainable. The analysis test *RTA_ 001* belongs to the *ResponseTimeAnalysis* test family.


Properties

Id : Simulation_001

Name : Simulation for periodic simultaneous tasks

[A. C. GENIET, E. GROLLEAU, Minimal schedulability interval for real time systems of periodic tasks with offsets , Theoretical of Computer Sciences, vo

Description : A simulation test based on the least common multiple of periods for simultaneous tasks

Its Test Family : Test Family Simulation

Characteristics
- Sufficient Condition
- Necessary Condition

Tools To Apply :

Properties

Id: CheddarSchedulingSimulation

Name: Cheddar version 2.1

Description: Once the system modeling is done under the Cheddar tool, choose the "scheduling simulation" button

Figure 7.12: The *Simulation_001* analysis test and its properties

IV.2. Extension of the repository instance

Figure 7.14 depicts the current content of the analysis repository instance. This content is a result of the instantiation done in Section IV.1. In the following, we show the extension of the analysis repository instance, by adding two real-time contexts and their analysis tests. Therefore, we will face two different cases.

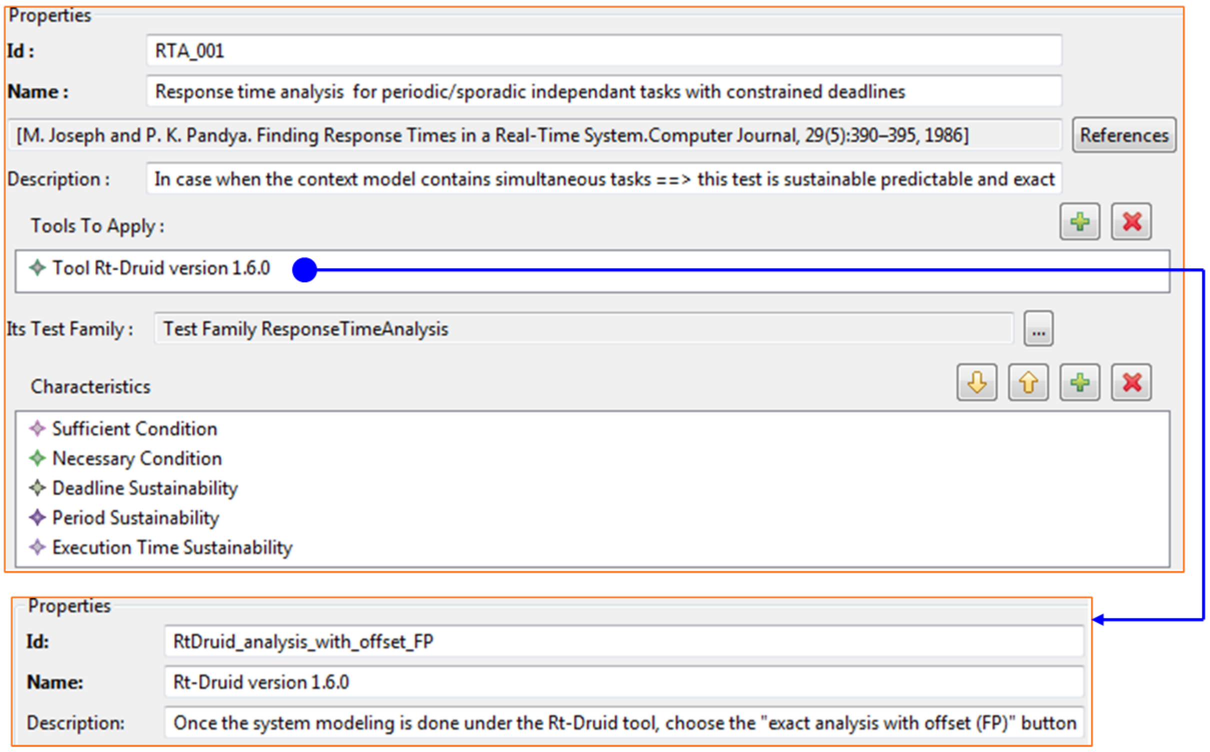
IV.2.a. Case 1: adding new context based on existing identification rules

In this case, we will add a new real-time context which looks like the Liu and Layland model, but the deadlines are arbitrary. Since the identification rules existing in the analysis repository instance are sufficient to define the new context, we will not add any new identification rule.

Adding a new real-time context Figure 7.15 shows the creation of the new real-time context that is called *Basic_Periodic_Model_with_Arbitrary_Deadlines*. After the creation of a new real-time context, all identification rules are considered by default as rules with undefined evaluation result. Next, users have to specify on which category each identification rule belongs to.

As the *Basic_Periodic_Model_with_Arbitrary_Deadlines* context covers models with both concrete and non concrete tasks, the *ConcreTasks* and *SimultaneousTasks* rules have been classified as rules with undefined evaluation result. The *ImplicitDeadlines* rule has also been classified in “Undefined Evaluation” part, because the context covers tasks with arbitrary deadlines which is a general case of tasks with implicit deadlines.

Adding the corresponding analysis test Figure 7.16 illustrates the creation of the response time analysis test called *RTA_002*. This test corresponds to the



Id :	RTA_001
Name :	Response time analysis for periodic/sporadic independant tasks with constrained deadlines
[M. Joseph and P. K. Pandya. Finding Response Times in a Real-Time System.Computer Journal, 29(5):390–395, 1986]	References
Description :	In case when the context model contains simultaneous tasks ==> this test is sustainable predictable and exact
Tools To Apply :	Tool Rt-Druid version 1.6.0
Its Test Family :	Test Family ResponseTimeAnalysis
Characteristics	Sufficient Condition, Necessary Condition, Deadline Sustainability, Period Sustainability, Execution Time Sustainability

Id:	RtDruid_analysis_with_offset_FP
Name:	Rt-Druid version 1.6.0
Description:	Once the system modeling is done under the Rt-Druid tool, choose the "exact analysis with offset (FP)" button

Figure 7.13: The *RTA_001* response time analysis test, its properties and its tool provider

Basic_Periodic_Model_with_Arbitrary_Deadlines context. The test is presented by Lehoczky in [Leh90] (see “References” property), and it is implemented by the Mast tool especially via the functionality said “periodic task editor”.

IV.2.b. Case 2: adding a new context requiring the addition of new identification rules

This case is devoted to show the instantiation of a real-time context representing an uncompleted model which needs to be dimensioned.

Adding the real-time context analysis test and its appropriate analysis test The new instance of the `ContextModel` element is dedicated to construct a real-time context for simultaneous periodic tasks with arbitrary deadlines, where the scheduling protocol is EDF (Earliest Deadline First) and the hardware architecture is uniprocessor. The incompleteness of this model is due to the absence of the worst-case execution time of a task. So, it is necessary to add one or several identification rules. Figure 7.17 shows the new real-time context called *Uncompleted_Periodic_Model_with_Arbitrary-_Deadlines*. The “True Evaluation” part contains the identification rule that we have added and which is called *WCET_of_a_task_is_unknown*. In case when the evaluation of the *WCET_of_a_task_is-_unknown* rule is true, that means there exists a task which its worst-execution execution time is unknown.

As the *Uncompleted_Periodic_Model_with_Arbitrary_Deadlines* context represents a model which needs to be dimensioned, then the sensitivity analysis test is one of the appropriate tests. As it is de-

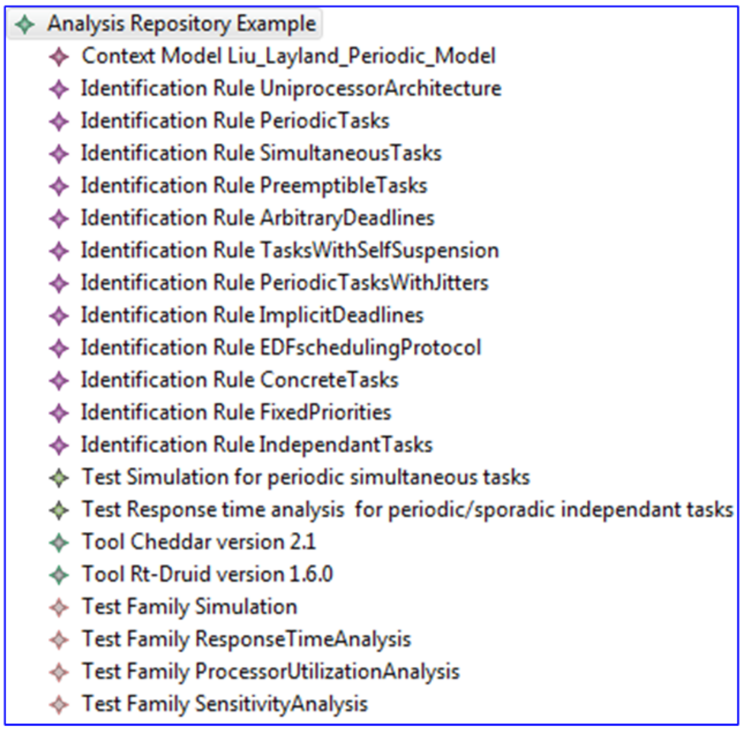
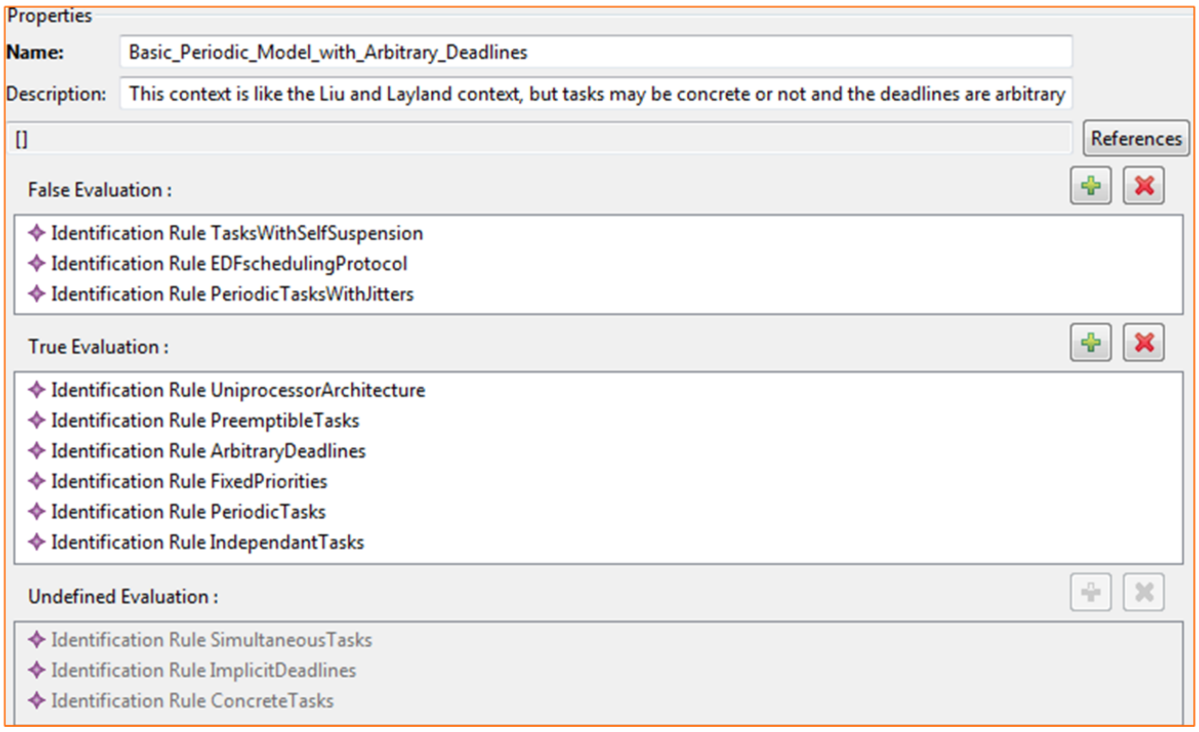

Analysis Repository Example
- Context Model Liu_Layland_Periodic_Model
- Identification Rule UniprocessorArchitecture
- Identification Rule PeriodicTasks
- Identification Rule SimultaneousTasks
- Identification Rule PreemptibleTasks
- Identification Rule ArbitraryDeadlines
- Identification Rule TasksWithSelfSuspension
- Identification Rule PeriodicTasksWithJitters
- Identification Rule ImplicitDeadlines
- Identification Rule EDFschedulingProtocol
- Identification Rule ConcreteTasks
- Identification Rule FixedPriorities
- Identification Rule IndependantTasks
- Test Simulation for periodic simultaneous tasks
- Test Response time analysis for periodic/sporadic independant tasks
- Tool Cheddar version 2.1
- Tool Rt-Druid version 1.6.0
- Test Family Simulation
- Test Family ResponseTimeAnalysis
- Test Family ProcessorUtilizationAnalysis
- Test Family SensitivityAnalysis

Figure 7.14: Current content of the analysis repository instance

pcted in Figure 7.17, we instantiate a new sensitivity analysis test (*Sensitivity_ Execution_ Time_ 001*). It is proposed by Zhang et al. in [ZBB11]. However, in our best knowledge we do not know any implementation of this analysis test via an analysis tool. Indeed, interested users can use manually (or program) the mathematical formulas presented in [ZBB11].

Impact of adding new identification rules While two real-time contexts have been created before, the addition of the *WCET_ of_ a_ task_ is_ unknown* identification rule in the analysis repository instance has an impact on the existing contexts. The identification rule has been automatically added as a rule with undefined evaluation. Therefore, every context has to be modified in order to precise which part must contain the identification rule. Figure 7.18 shows the Liu & Layland context after adding the identification rule *WCET_ of_ a_ task_ is_ unknown*. This latter is classified in the “False Evaluation” part.


Properties

Name: Basic_Periodic_Model_with_Arbitrary_Deadlines

Description: This context is like the Liu and Layland context, but tasks may be concrete or not and the deadlines are arbitrary

[]

References

False Evaluation :

- Identification Rule TasksWithSelfSuspension
- Identification Rule EDFschedulingProtocol
- Identification Rule PeriodicTasksWithJitters

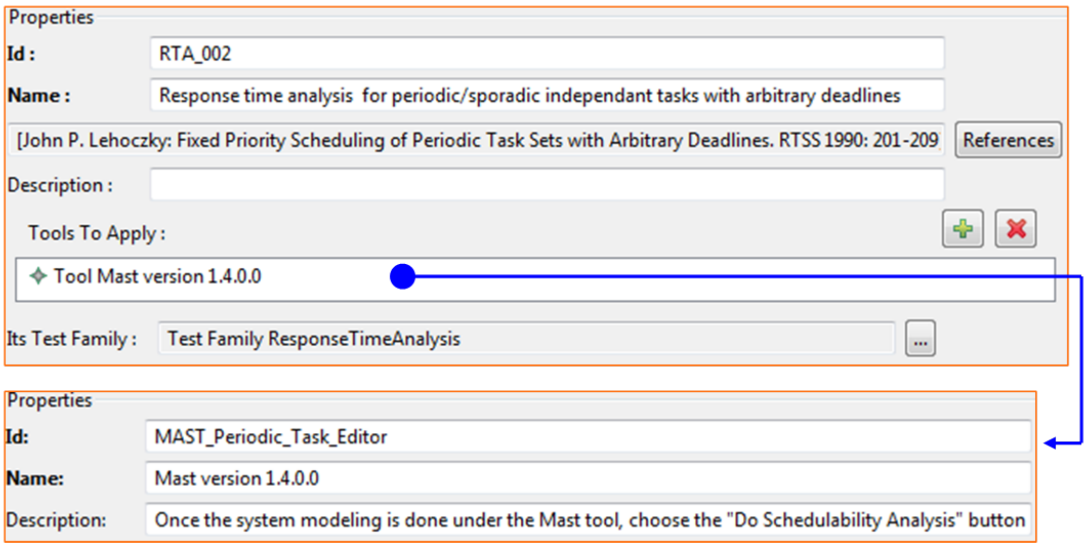
True Evaluation :

- Identification Rule UniprocessorArchitecture
- Identification Rule PreemptibleTasks
- Identification Rule ArbitraryDeadlines
- Identification Rule FixedPriorities
- Identification Rule PeriodicTasks
- Identification Rule IndependantTasks

Undefined Evaluation :

- Identification Rule SimultaneousTasks
- Identification Rule ImplicitDeadlines
- Identification Rule ConcreteTasks

Figure 7.15: *Basic_ Periodic_ Model_ with_ Arbitrary_ Deadlines* real-time context and its properties


Properties

Id : RTA_002

Name : Response time analysis for periodic/sporadic independant tasks with arbitrary deadlines

[John P. Lehoczky: Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. RTSS 1990: 201-209

References

Description :

Tools To Apply :

- Tool Mast version 1.4.0.0

Its Test Family : Test Family ResponseTimeAnalysis

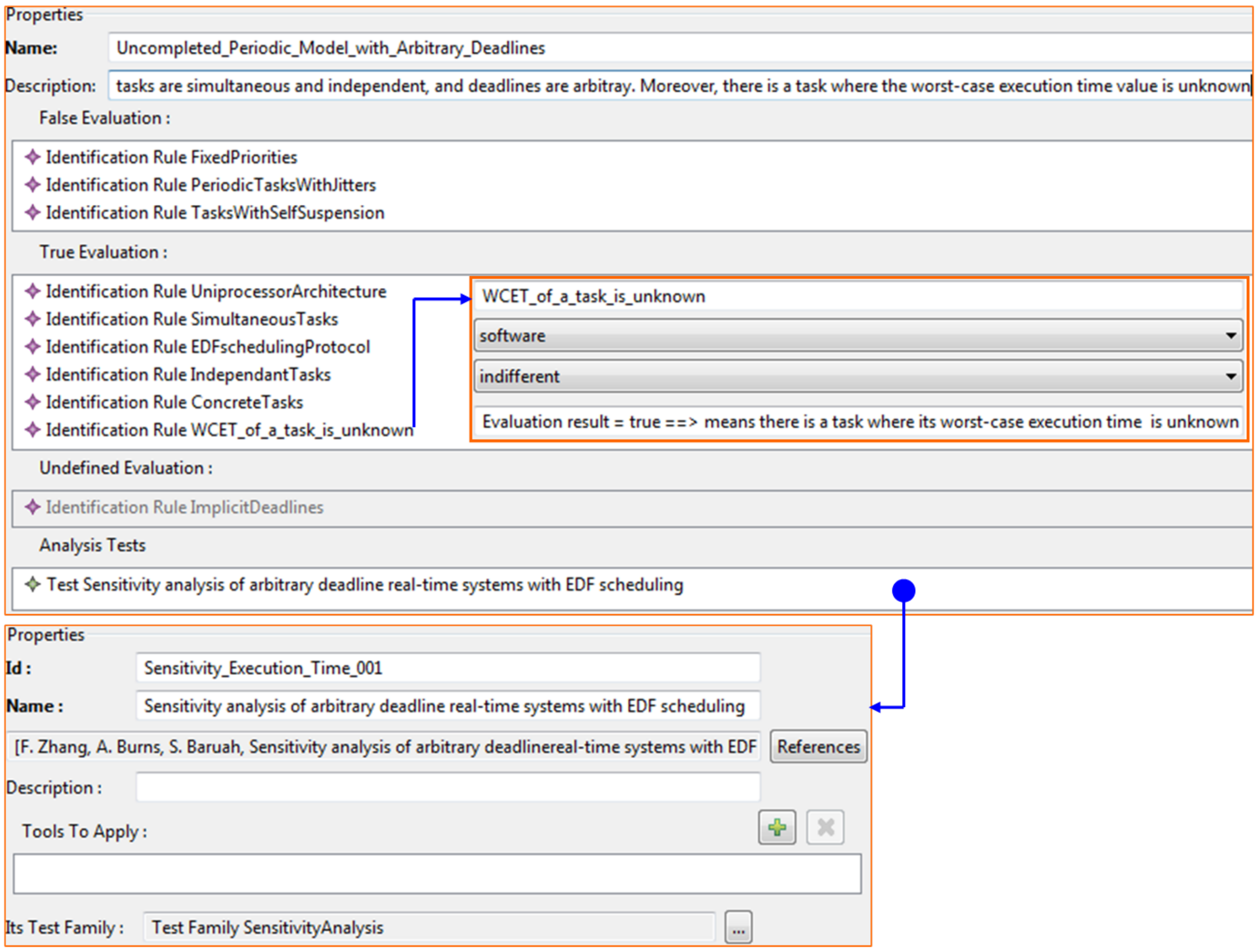
Properties

Id: MAST_Periodic_Task_Editor

Name: Mast version 1.4.0.0

Description: Once the system modeling is done under the Mast tool, choose the "Do Schedulability Analysis" button

Figure 7.16: The *RTA_ 002* response time analysis test, its properties and its tool provider



Properties
Name: Uncompleted_Periodic_Model_with_Arbitrary_Deadlines
Description: tasks are simultaneous and independent, and deadlines are arbitray. Moreover, there is a task where the worst-case execution time value is unknown

False Evaluation :
- Identification Rule FixedPriorities
- Identification Rule PeriodicTasksWithJitters
- Identification Rule TasksWithSelfSuspension

True Evaluation :
- Identification Rule UniprocessorArchitecture
- Identification Rule SimultaneousTasks
- Identification Rule EDFschedulingProtocol
- Identification Rule IndependantTasks
- Identification Rule ConcreteTasks
- Identification Rule WCET_of_a_task_is_unknown

WCET_of_a_task_is_unknown
software
indifferent
Evaluation result = true ==> means there is a task where its worst-case execution time is unknown

Undefined Evaluation :
- Identification Rule ImplicitDeadlines

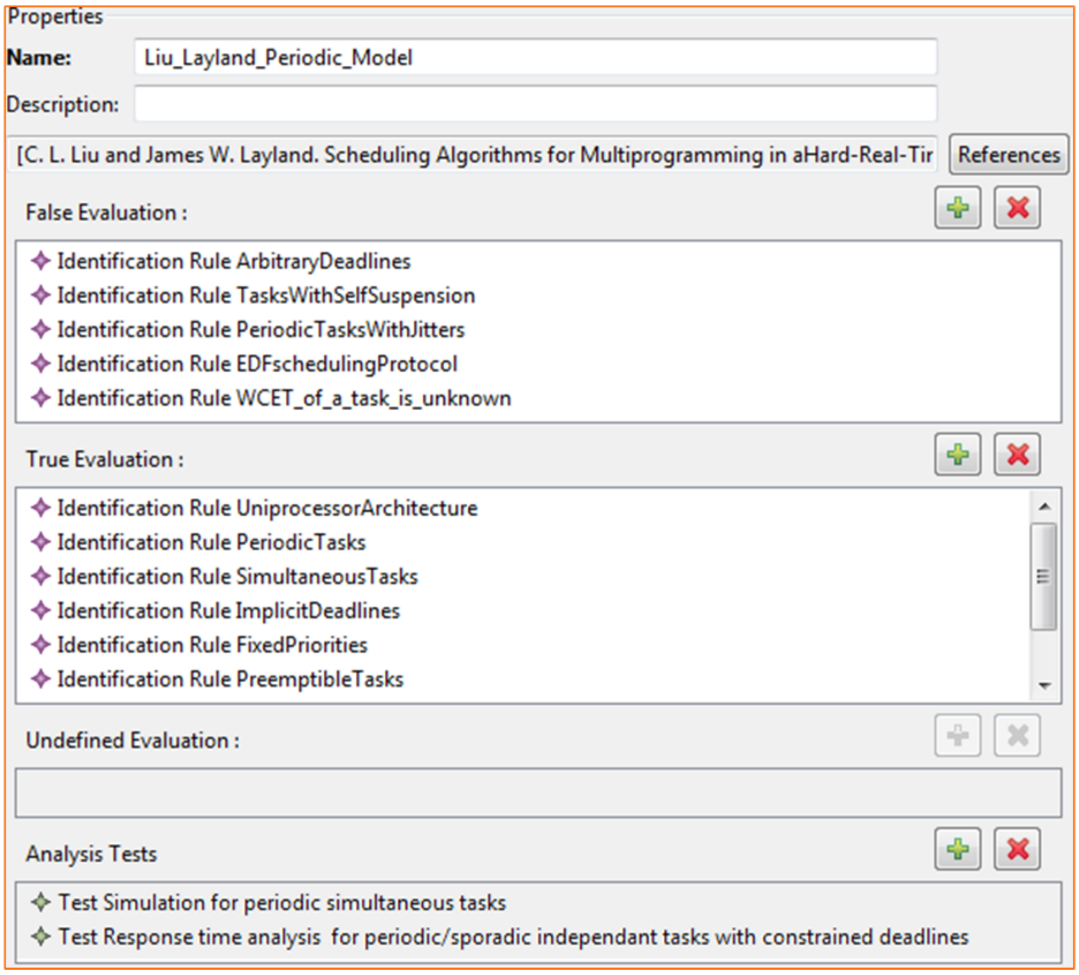
Analysis Tests
- Test Sensitivity analysis of arbitrary deadline real-time systems with EDF scheduling

Properties
Id : Sensitivity_Execution_Time_001
Name : Sensitivity analysis of arbitrary deadline real-time systems with EDF scheduling
[F. Zhang, A. Burns, S. Baruah, Sensitivity analysis of arbitrary deadlinereal-time systems with EDF
Description :
Tools To Apply :
Its Test Family : Test Family SensitivityAnalysis

Figure 7.17: *Uncompleted_ Periodic_ Model_ with_ Arbitrary_ Deadlines* real-time context and its properties

IV.3. Discussion

In this section, we have focused on basic models in order to explain clearly the capabilities of the MoSaRT Analysis Repository. Moreover, we have chosen relatively simple contexts which possess one or several analysis tests and their implementation via analysis tools. Obviously, it is possible to instantiate any complex real-time context with or without analysis tests.
For a practical reason, it is preferable to instantiate at the beginning the identification rules before instantiating the real-time contexts. Although, the modification of existing contexts after adding an identification rule can be tedious (e.g. when the number of contexts is very high), the modification of those contexts can be viewed as a precision task enabling to orient designer to choose the most accurate context. Moreover, as the real-time community is very active and the real-time contexts are in constant improvement, we cannot avoid the enrichment of the analysis repository instances by new identification rules if we aim to get an analysis repository instance up to date.


Properties
Name: Liu_Layland_Periodic_Model
Description:
[C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in aHard-Real-Tir

False Evaluation :
- Identification Rule ArbitraryDeadlines
- Identification Rule TasksWithSelfSuspension
- Identification Rule PeriodicTasksWithJitters
- Identification Rule EDFschedulingProtocol
- Identification Rule WCET_of_a_task_is_unknown

True Evaluation :
- Identification Rule UniprocessorArchitecture
- Identification Rule PeriodicTasks
- Identification Rule SimultaneousTasks
- Identification Rule ImplicitDeadlines
- Identification Rule FixedPriorities
- Identification Rule PreemptibleTasks

Undefined Evaluation :

Analysis Tests
- Test Simulation for periodic simultaneous tasks
- Test Response time analysis for periodic/sporadic independant tasks with constrained deadlines

Figure 7.18: Liu & Layland real-time context after the repository enrichment

Technical solutions which are used to implement the MoSaRT analysis repository are presented in Appendix B.

V. Conclusion

In this chapter, we have presented the MoSaRT Analysis Repository. Each instantiation of the MoSaRT Analysis Repository offers a repository based on a set of identification rules, their evaluation can lead to the real-time context of the design model. Moreover, the meta-model of the MoSaRT Analysis Repository is flexible and can be enriched by analysts in order to add other characteristics. Every analysis repository instance can be also viewed as a warehouse of models and tests allowing real-time researchers to compare their results with existing ones. A particular use of the analysis repository is when designers know the context of their system, they can set their choice at the beginning of the design (in this case the utilization of the analysis repository seems like a design pattern). Then, at

each design change (adding tasks, removing processors, etc.) different analyses will be recalculated interactively to provide results within the modeling.
Independently from any design language, we have presented an example showing the instantiation of the MoSaRT analysis repository. At this stage, the identification rules have been presented in an informal way. The core of identification rules and how they check the design model have not been yet discussed. Yet, how model transformation through the analysis repository instance eases the use of analysis tools has not been presented.
In the next chapter, we link the MoSaRT analysis repository with the MoSaRT design language in order to obtain a complete design framework. This enables the design at the same time as to orient designers (especially modelers) to choose the most suitable analytical models and schedulability tests. Several case studies will also be stressed in the next chapter.

Chapter

8

MoSaRT Framework

Contents

Abstract

This chapter presents the MoSaRT framework which is designed as a bridge between model-based design methods and scheduling analysis third-party tools. The chapter starts by introducing the general objectives of this framework. Next, it presents the structure of the MoSaRT framework and its various capabilities. Furthermore, the framework is stressed through three detailed case studies.

I. Introduction

First of all, the utilization of MoSaRT language solely is insufficient. Secondly, our goal is to help designers to conclude about the schedulability of their designs. Thirdly, the use of MoSaRT analysis repository eases the task of seeking for the appropriate analysis tests and their providers. Consequently, referring to the context of real-time systems development, we believe that the existence of a canvas as a result of the correlation between the MoSaRT design language and the MoSaRT analysis repository can be very beneficial. Thence, this canvas is the MoSaRT framework with its two components: the MoSaRT front-end based on the design language and the MoSaRT back-end based on the analysis repository.
This chapter is structured as follows. Next section presents the objectives of the framework and the connection of the MoSaRT design language and the MoSaRT analysis repository. Section III introduces the architecture of the MoSaRT framework and its capabilities. Furthermore, some case studies sketch its feasibility in Section IV. Finally, Section V concludes this chapter.

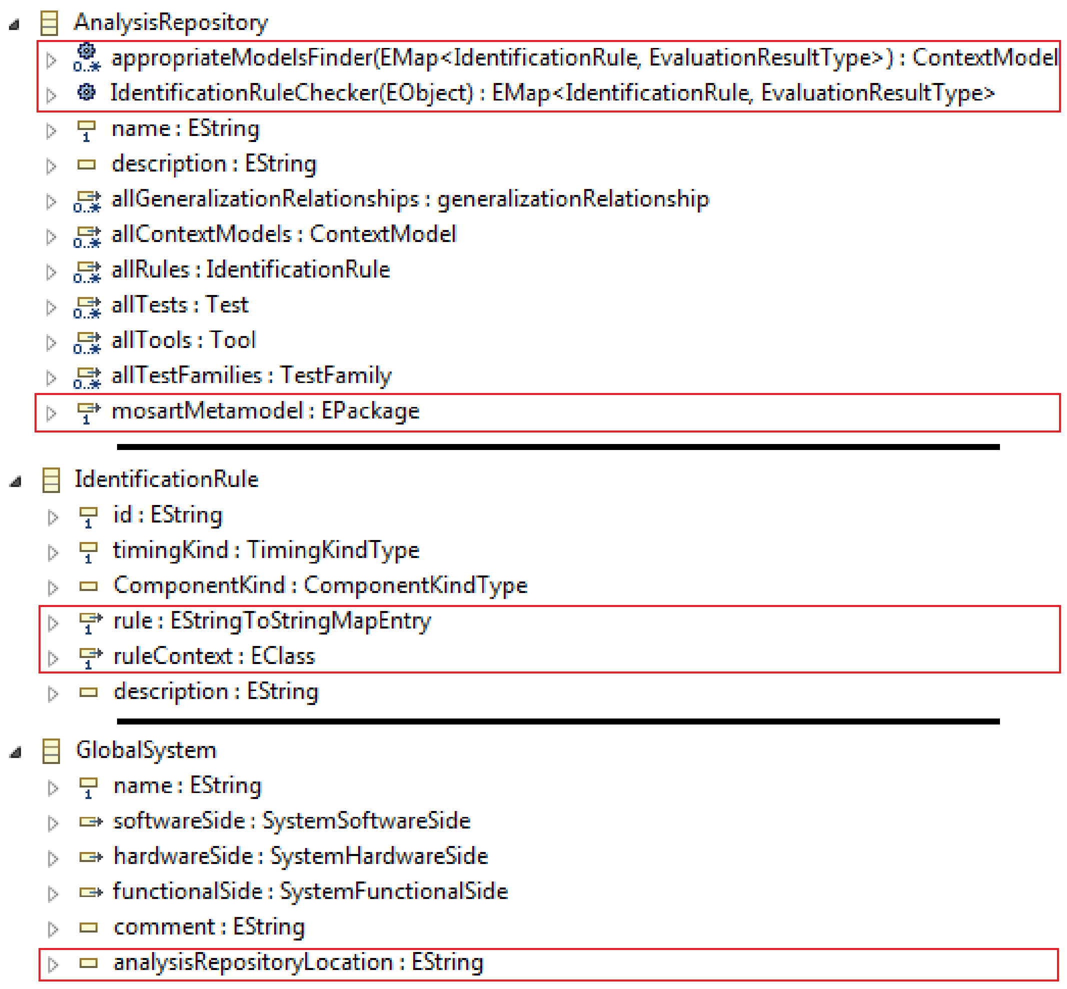
II. MoSaRT framework

Objectives The objective behind the MoSaRT framework is to benefit from analysts/researchers skills and modelers/architects skills in order to unify their efforts, then to avoid wrong design choices at an early design phase. So, by proposing the MoSaRT framework, we aim:

- to help designers to cope with the scheduling analysis and to be more autonomous during the analysis stage.
- to help analysts to alleviate their efforts related to the analysis of design models.

Connection impacts To ensure a coherent usage of the framework, the connection between the MoSaRT analysis repository and the MoSaRT language needs to add some features. Then, identification rules $\mathcal{R}$ and analysis repository Ar elements are impacted by this connection. Figure 8.1 depicts the enrichment of the `AnalysisRepository` class and the `IdentificationRule` class. The `GlobalSystem` class of the MoSaRT design language has been also enriched by adding the `analysis-RepositoryLocation` property which serves to locate an instance of the MoSaRT analysis repository. As the `AnalysisRepository` class is the root element of the analysis repository model, we have added the `mosartMetamodel` property referencing the link to the MoSaRT language.
In the `IdentificationRule` class, the `rule` and `ruleContext` properties have been added. The `rule` property is dedicated to contain formal expressions which have to be checked to get the evaluation result. Every `rule` is related to a special `ruleContext`. The `ruleContext` property defines the element of the MoSaRT language on which the `rule` can be evaluated.

Formal expressions of the `rule` properties are expressed in OCL (Object Constraint Language). We have chosen OCL for two reasons. The first reason is that OCL operates well with Ecore (meta-modeling language) which is used not only in the conception of the MoSaRT design language, but also in the conception of AADL, MARTE, EAST-ADL, etc. Thus, that facilitates the use of the MoSaRT analysis repository with several design languages. The second reason is related to a model-driven



```
AnalysisRepository
  appropriateModelsFinder(EMap<IdentificationRule, EvaluationResultType>) : ContextModel
  IdentificationRuleChecker(EObject) : EMap<IdentificationRule, EvaluationResultType>
  name : EString
  description : EString
  allGeneralizationRelationships : generalizationRelationship
  allContextModels : ContextModel
  allRules : IdentificationRule
  allTests : Test
  allTools : Tool
  allTestFamilies : TestFamily
  mosartMetamodel : EPackage

IdentificationRule
  id : EString
  timingKind : TimingKindType
  ComponentKind : ComponentKindType
  rule : EStringToStringMapEntry
  ruleContext : EClass
  description : EString

GlobalSystem
  name : EString
  softwareSide : SystemSoftwareSide
  hardwareSide : SystemHardwareSide
  functionalSide : SystemFunctionalSide
  comment : EString
  analysisRepositoryLocation : EString
```

Figure 8.1: Elements enrichment (new features are framed) of both MoSaRT analysis repository and MoSaRT language

engineering perspective. So, no code has to be provided by users. Indeed, every formal expression of a `rule` instance represents an instantiation of the OCL metamodel, hence that enables a runtime use of the repository despite its modification (enrichment or suppression of the repository elements).
Listing 8.1 shows an example of an OCL expression for checking if all tasks of the design model (instance of the MoSaRT language) are characterized by implicit deadlines. While these expressions require a good knowledge of MoSaRT language elements, we provide a set of identification rules with descriptions and understandable names. That facilitates the instantiation/enrichment of existing instances of the MoSaRT analysis repository without requiring a high knowledge of MoSaRT design language nor an OCL knowledge.

Listing 8.1: Identification rule expressed in OCL for checking implicit deadlines

```
(self.softwareSide.systemBehaviour.schedulingActivities
->select(sa|sa.oclIsTypeOf(SoftwareBehaviour::SbTaskActivity)))
->forAll(ta|ta.oclAsType(SoftwareBehaviour::SbTaskActivity
.rtpDeadline.deadline.equals
(ta.oclAsType(SoftwareBehaviour::SbTaskActivity)
.trigger.oclAsType(SoftwareBehaviour::SbTimeTrigger)
.rtpPeriodicity.oclAsType
(RealTimeProperties::RtpTypes::RtpPeriodicType).period))
```

The `AnalysisRepository` class proposes two services.
The first service is represented by the `identificationRuleChecker` operation. It needs the root element of the design model (instance of the `GlobalSystem` class) and returns a result summarizing the evaluation of each rule referring to the input design model. Once the evaluation is done, the second service represented by the `appropriateModelsFinder` operation is solicited. This operation is based on the obtained evaluation result (i.e. result of `identificationRuleChecker` operation) and checks if it exists any appropriate real-time context in the repository.

III. Utilization scenarios of MoSaRT framework: different capabilities

Besides the MoSaRT language, the MoSaRT analysis repository can also be linked to other modeling languages like MARTE[OMG09] or AADL[oAES]. However, the current standard design languages contain many artifacts to annotate models for many analysis concerns (not only the scheduling analysis). This richness makes the use of these standards for the scheduling analysis expensive due to several reasons discussed in Chapter 5.
To reduce the gap between the modeling and analysis sides and to face many issues related to this gap, we propose the intermediate MoSaRT framework to fill the gap between the design languages and the scheduling analyses. Then, via the MoSaRT framework we look at a wide utilization taking benefits from this variety of existing standard design languages and analysis tools. Hereafter, we introduce the capabilities of the two MoSaRT framework ends: the back-end and the front-end.

III.1. Back-end capabilities

The back-end of the MoSaRT framework offers to analysis experts (industrial analysts, researchers) several capabilities. Figure 8.2 indicates some of them. One can suggest a new analysis repository (related to a specific company/laboratory/research team) by instantiating the analysis repository model (Action (1) of Figure 8.2). In this case, one has to define at least the contexts and tests to ensure the usability of the repository. Any analysis repository can be enriched progressively by adding new contexts, tests, tools, etc (Action (2) of Figure 8.2). Moreover, every real-time context existing in such a repository can be refined by specifying more accurate identification rules, more characteristics of the analysis tests, computerizing the transformation to analysis tools, etc. (Action (3) of Figure 8.2). That enables to get a sound and coherent repository. Once an analysis repository becomes ready for use, it can be shared at run-time in order to be used by designers (Action (4) of Figure 8.2).

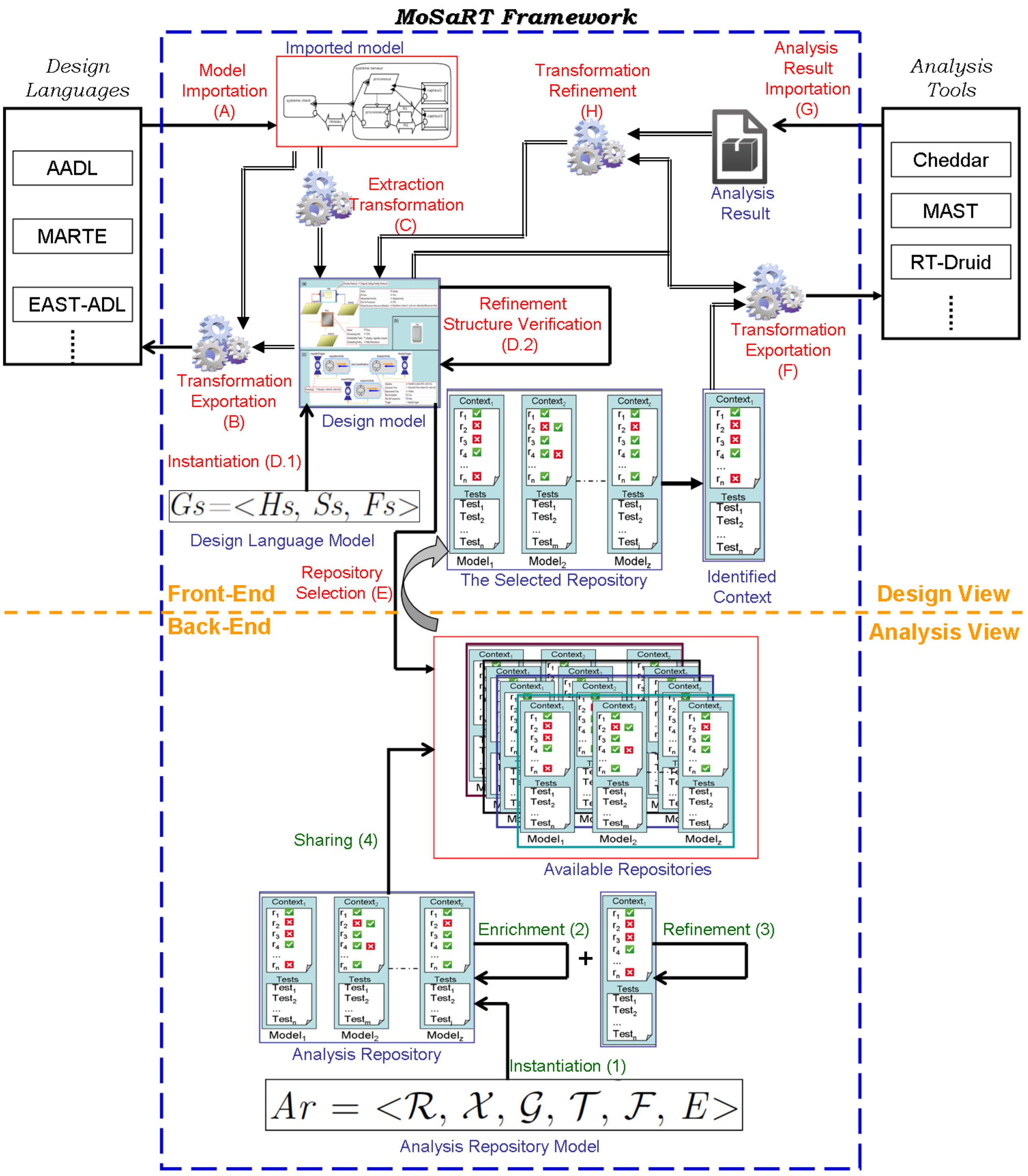


Figure 8.2: Some relevant utilization scenarios related to the MoSaRT Framework

In some industrial areas like avionics, where the engineers engaged in design outnumber the real-time analysts, the repository analysis concept can be very helpful. It can also be used as a design pattern supporting designers to obtain specific models respecting certain context previously set by an analyst.

III.2. Front-end capabilities

Via the front-end of the MoSaRT framework, designers (in particular modelers and architects) can import the design models expressed in a standard design language, like AADL (Action A of Figure 8.2). Furthermore, designers can use the transformation process offered by the MoSaRT framework in order to transform their models to the MoSaRT design language. The transformation process from a standard design language to MoSaRT language is based on the extraction of timing details (Action C of Figure 8.2). Actions A and C of Figure 8.2 are not mandatory. The MoSaRT framework gives also the possibility to design a model directly (Action D.1 of 8.2). Once designers obtain a model expressed in MoSaRT language, they can refine it and check the structural correctness (Action D.2 of Figure 8.2). Next, designers move to the analysis phase by selecting one of the available analysis repositories (Action E of Figure 8.2). Then, the identification process can be launched. When identifying the corresponding real-time context and if the repository is rich enough, a customizable transformation to analysis tools could be provided (Action F of Figure 8.2). After obtaining the analysis result, MoSaRT framework offers the possibility to import the tool output files (Action G of Figure 8.2). Due to technical transformation reasons, Action H of Figure 8.2 requires both the original design expressed in the MoSaRT language and the analysis result file. The refinement and the analysis actions can be repeated many times until getting accurate models. The transformation of the analyzed model to an external design language can be done only if the model was previously imported (Action B of Figure 8.2).

IV. Case studies: proof of concepts

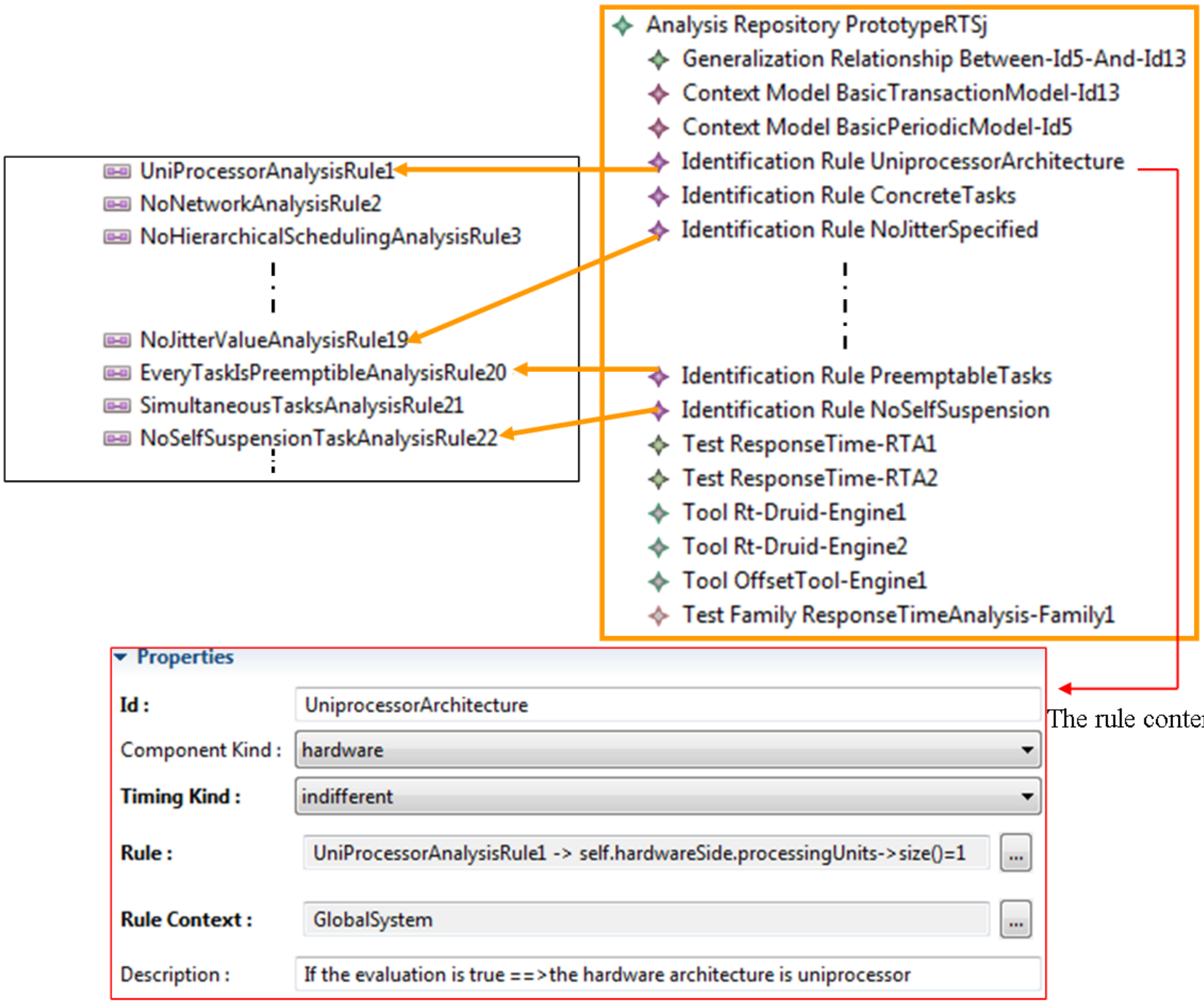
We illustrate the design and the analysis concepts presented in this thesis by devoting this section to present three case studies.

IV.1. Case study 1

While our objective is to motivate the real-time scheduling community to increase the usability of the scheduling theory, in this case study, we emphasize the impact of the analysis repository's utilization on the design.

IV.1.a. Instantiation of the MoSaRT analysis repository

First of all we provide an analysis repository containing a real-time context corresponding to the periodic model [JP86]. This context is based on several identification rules (some of them are shown in Figure 8.3). Each rule is mapped to a formal expression related to the MoSaRT design language. Then, we add the real-time context of the transaction model [Tin94]. This latter represents a generalization (for both worst-case and best-case behaviors) of the periodic model which had been previously created. We embody the generalization relationship, and we connect it with a transformation program enabling


Analysis Repository PrototypeRTSj
Generalization Relationship Between-Id5-And-Id13
Context Model BasicTransactionModel-Id13
Context Model BasicPeriodicModel-Id5
Identification Rule UniprocessorArchitecture
Identification Rule ConcreteTasks
Identification Rule NoJitterSpecified
Identification Rule PreemptableTasks
Identification Rule NoSelfSuspension
Test ResponseTime-RTA1
Test ResponseTime-RTA2
Tool Rt-Druid-Engine1
Tool Rt-Druid-Engine2
Tool OffsetTool-Engine1
Test Family ResponseTimeAnalysis-Family1
UniProcessorAnalysisRule1
NoNetworkAnalysisRule2
NoHierarchicalSchedulingAnalysisRule3
NoJitterValueAnalysisRule19
EveryTaskIsPreemptibleAnalysisRule20
SimultaneousTasksAnalysisRule21
NoSelfSuspensionTaskAnalysisRule22
The rule content
Properties

Id :	UniprocessorArchitecture
Component Kind :	hardware
Timing Kind :	indifferent
Rule :	UniProcessorAnalysisRule1 -> self.hardwareSide.processingUnits->size()=1
Rule Context :	GlobalSystem
Description :	If the evaluation is true ==>the hardware architecture is uniprocessor

Figure 8.3: Example of the MoSaRT analysis repository instance

the transition from the periodic model to the transaction model (when it is possible). The transformation program is done on ATL (Atlas Transformation Language) [JK05]. It represents an endogenous transformation (i.e. a transformation from MoSaRT design language to MoSaRT design language). We create a RTA (Response Time Analysis) test family which contains two items of the response time analysis. The first one is dedicated to analyze the periodic model. This test is implemented by several tools, like MAST [MPHD01] and Rt-Druid [GNS$^+$07]. The second RTA test is devoted to analyze the transaction model. We mention RT-Druid and Offset-Tool [oY94] as providers of the second RTA test. In order to ease the use of the analysis tools, we provide two transformation programs. The first one serves to transform the design to the Rt-Druid input formalism, this program is also based on ATL. The second transformation program serves to transform the design to the Offset-Tool input formalism. Since the Offset-Tool formalism is based on a textual format, the transformation is implemented in

Acceleo [Com].

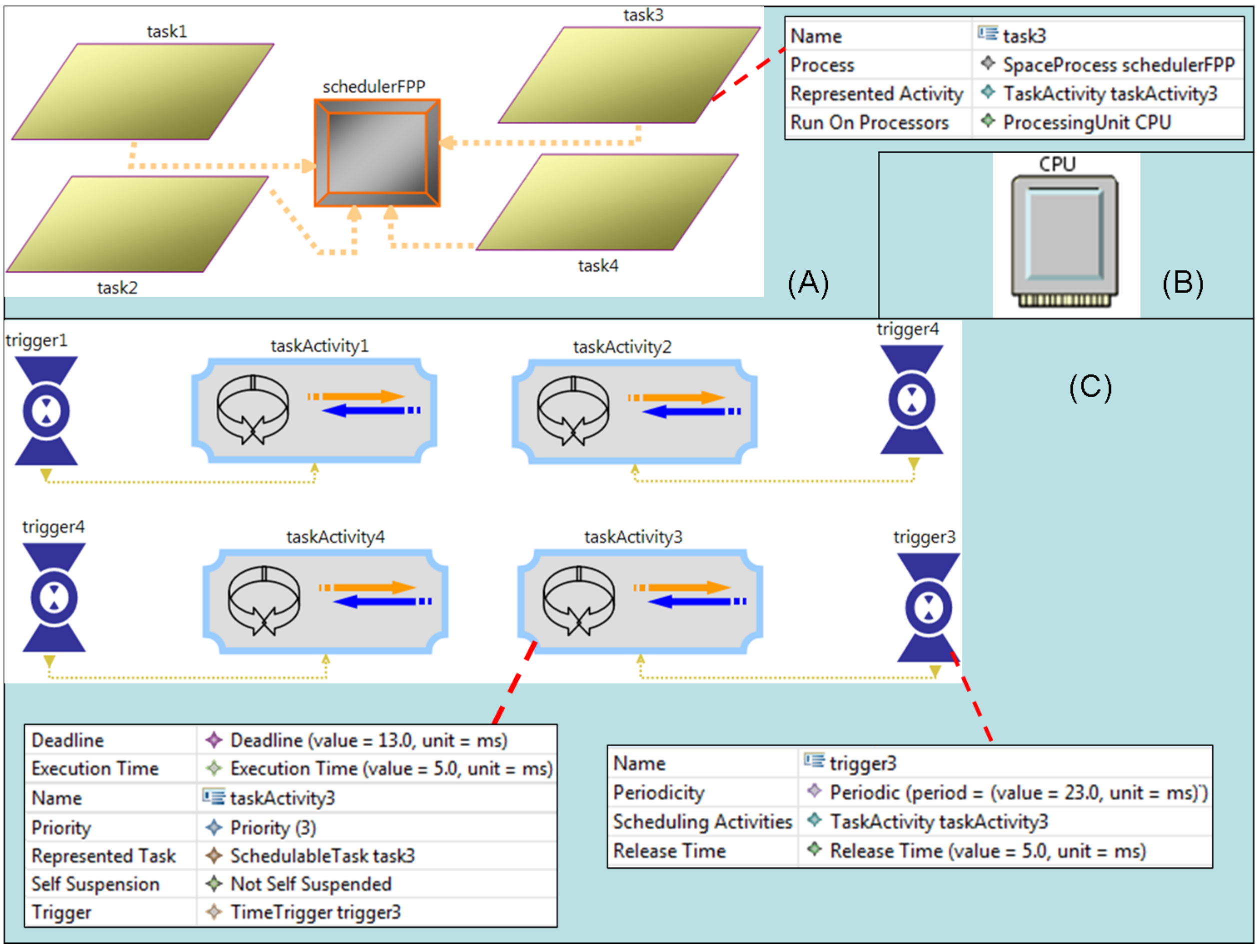
IV.1.b. Utilization of the MoSaRT design language

To check our analysis repository, we consider an application composed of four periodic independent tasks where each task is defined by a set of properties (See Table 8.1). The preemptible task-set is executed on an uniprocessor architecture and follows the fixed priority scheduling policy, where `task1` is the high-priority task and `task4` is the less-priority task.

Task	Execution time	Deadline	Period	Release time
Task1	3 ms	15 ms	20 ms	2 ms
Task2	4 ms	8 ms	23 ms	0 ms
Task3	5 ms	13 ms	23 ms	5 ms
Task4	9 ms	13 ms	23 ms	7 ms

Table 8.1: Values of the task-set characteristics

Figure 8.4 shows the design corresponding to our example. Parts (A) and (B) represents the architecture layer of the system (the software architecture mapped to the hardware architecture). Part (C) shows the software behavioral model of the system (like task activation, precedence relationships, task communication, etc.), where every task of the software model is represented by a `taskActivity` element. The behavioral model shows also the triggering kind of each task and its timing characteristics (Figure 8.4 also shows some relevant properties related to some elements).
We verify that the design structure is valid, hence we can select the analysis repository previously instantiated (in Section IV.1.a) . Figure 8.5 shows the result found after calling the identification process. Referring to the identification process's result, the analytical model tackled by [JP86] corresponds to the real-time context of the treated example. The result window (See Figure 8.5) contains also the characteristics tab which permits to summarizes the assumptions related to the found context, and the test tab for orienting designers to the appropriate tools (**note that the correctness of this result depends on the repository content**).
The design can be analyzed by the response time analysis (`rta1`). As it is mentioned by the result of the identification process, the test is implemented by Rt-Druid tool. So, a model respecting the Rt-Druid formalism can be generated after launching the transformation program. Figure 8.6 evinces the analysis result provided by Rt-Druid. The result means that the system is not schedulable, because the response time of `task4` does not respect the deadline. However, this result is pessimistic because the test provided by the analysis repository is only sufficient (because the critical instant may be not achievable).
In this case, a designer can try to change the task set characteristics or to modify the hardware architecture by using a faster CPU or multicore system for example. Nevertheless, this design model can be analyzed differently in order to avoid the over-dimensioning. Thanks to the identification process's result, the designer is informed that the design model can be refined in order to be corresponding to a transaction model (see the bottom of Figure 8.5).
This refinement is related to the generalization relationship between the transaction model and the periodic model. The refinement is an endogenous transformation based on the following rules:



Name	task3
Process	SpaceProcess schedulerFPP
Represented Activity	TaskActivity taskActivity3
Run On Processors	ProcessingUnit CPU

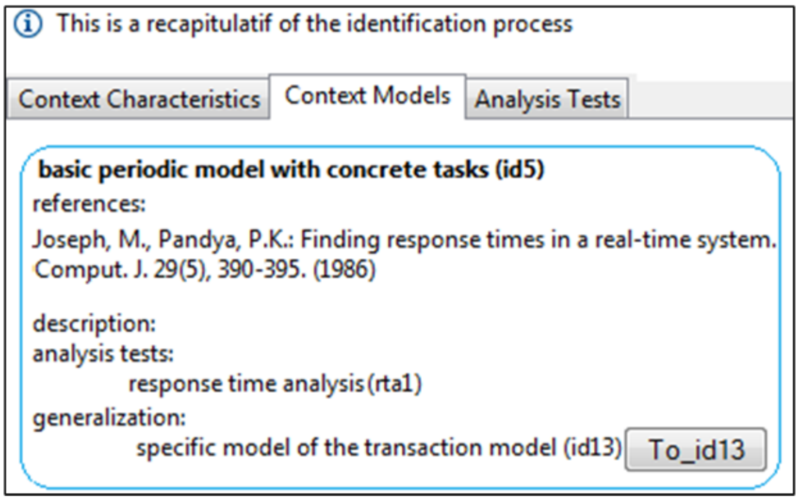
Deadline	Deadline (value = 13.0, unit = ms)
Execution Time	Execution Time (value = 5.0, unit = ms)
Name	taskActivity3
Priority	Priority (3)
Represented Task	SchedulableTask task3
Self Suspension	Not Self Suspended
Trigger	TimeTrigger trigger3

Name	trigger3
Periodicity	Periodic (period = (value = 23.0, unit = ms))
Scheduling Activities	TaskActivity taskActivity3
Release Time	Release Time (value = 5.0, unit = ms)

Figure 8.4: Different views of the design model corresponding to the treated example. (A): Software architecture. (B): Hardware architecture. (C): Software behavior

- For every set of independent tasks with the same period, tasks are grouped in the same transaction;
- The release-time of every trigger becomes an offset-time of the triggered task;

Figure 8.7 shows the behavioral model of the new system design produced after transformation. The new system design consists of Parts (A) and (B) of Figure 8.4 and the behavioral model illustrated in Figure 8.7. Hence, the first transaction consists of `task1`, and the second transaction consists of `task2`, `task3`, and `task4`. Some properties of `taskActivity3` are shown in Figure 8.7.
We note that the system maintains the same architecture, and the transition from the periodic model to the transaction model is only based on the behavioral part. We can say that the refinement is not based on adding more specifications, but it is related to a semantic precision when the system behavior can be viewed differently. Yet, the analysis repository can be called for a second time. Thus, the identification process proposes the transaction response-time analysis as a schedulability analysis test.



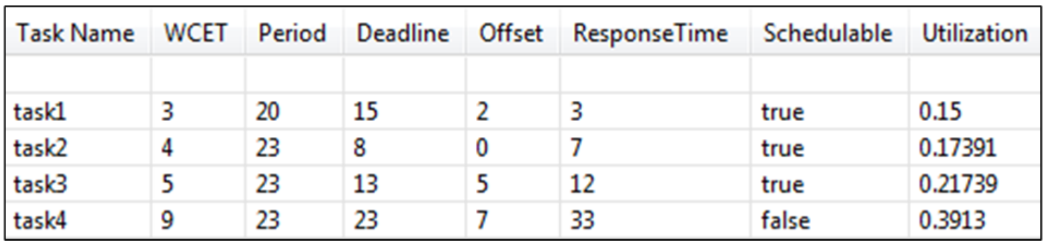
This is a recapitulatif of the identification process

Context Characteristics | Context Models | Analysis Tests

basic periodic model with concrete tasks (id5)
references:
Joseph, M., Pandya, P.K.: Finding response times in a real-time system. Comput. J. 29(5), 390-395. (1986)

description:
analysis tests:
response time analysis (rta1)
generalization:
specific model of the transaction model (id13) To_id13

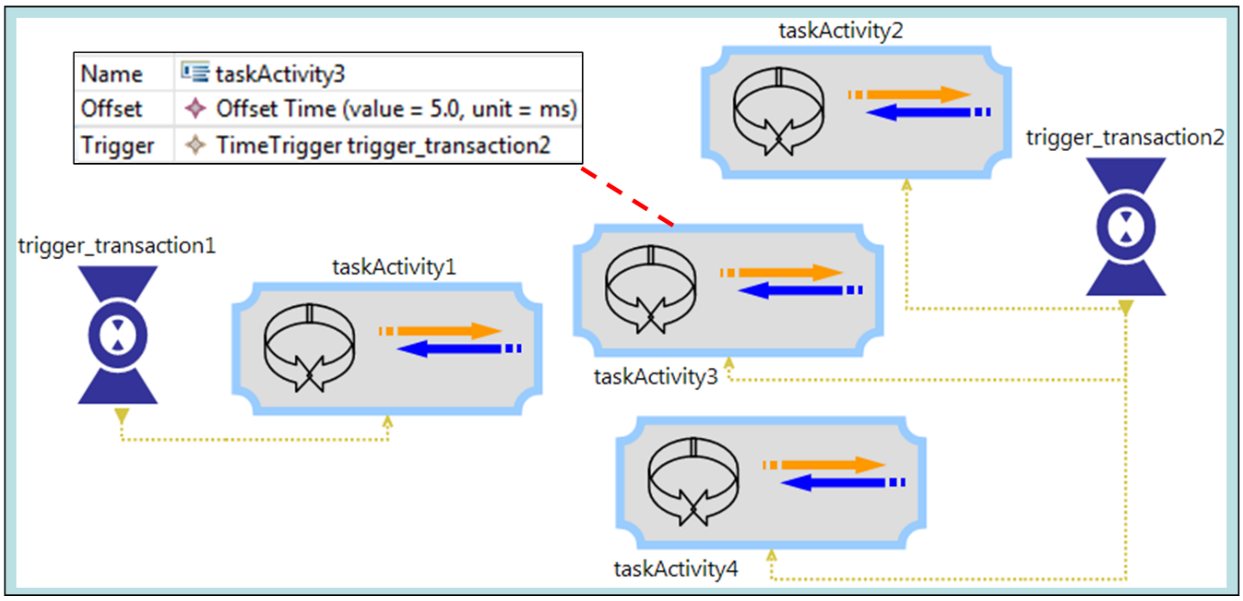
Figure 8.5: Extract of identification result provided by the analysis repository



Task Name	WCET	Period	Deadline	Offset	ResponseTime	Schedulable	Utilization
task1	3	20	15	2	3	true	0.15
task2	4	23	8	0	7	true	0.17391
task3	5	23	13	5	12	true	0.21739
task4	9	23	23	7	33	false	0.3913

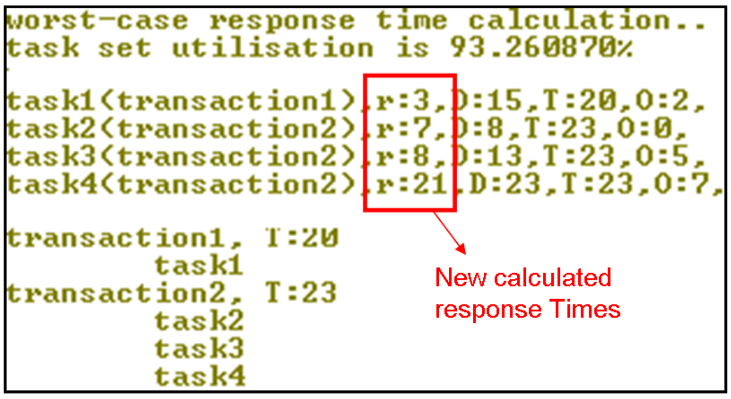
Figure 8.6: Analysis results provided by Rt-Duid

This latter is provided by Offset-Tool [oY94]. The result of the last analysis is illustrated in Figure 8.8. It shows that the design model is schedulable. Moreover, a comparison of the found response-time values to the first result indicates the pessimism of the first analysis test.



Name	taskActivity3
Offset	Offset Time (value = 5.0, unit = ms)
Trigger	TimeTrigger trigger_transaction2

Figure 8.7: Behavioral model produced after refinement (transformation)



```
worst-case response time calculation..
task set utilisation is 93.260870%

task1(transaction1),r:3,D:15,T:20,O:2,
task2(transaction2),r:7,D:8,T:23,O:0,
task3(transaction2),r:8,D:13,T:23,O:5,
task4(transaction2),r:21,D:23,T:23,O:7,

transaction1, T:20
        task1
transaction2, T:23
        task2
        task3
        task4
```

Figure 8.8: Analysis results provided by Offset-Tool [oY94]

IV.2. Case study 2

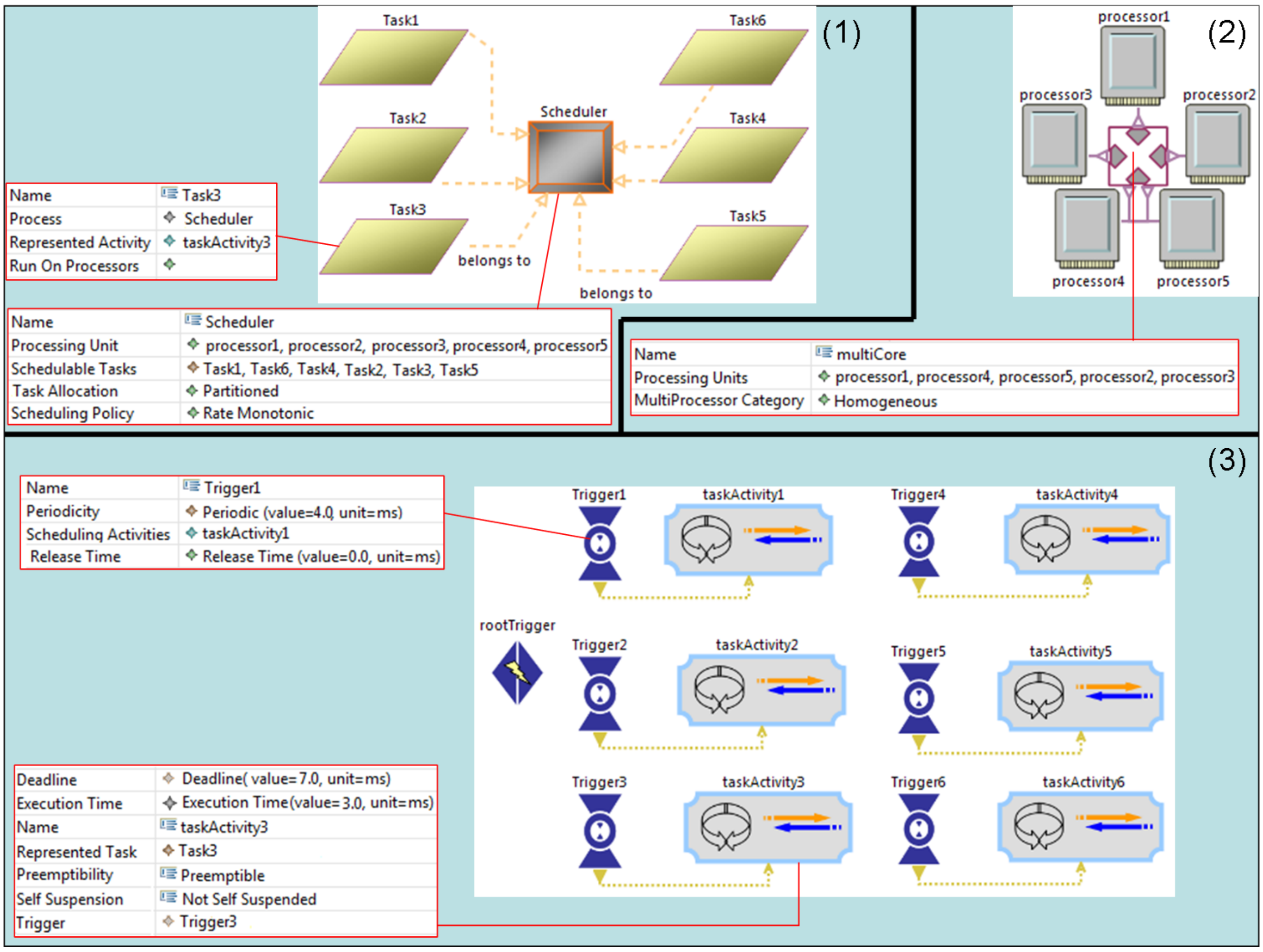
We have chosen an application composed of six synchronous independent tasks where each task is defined by a set of properties. Table 8.2 summarizes the task characteristics. The task set will be executed on a multicore architecture which contains five homogeneous cores.

IV.2.a. Modeling aspect

Figure 8.9 shows the design model of the treated example. This model is done by using the MoSaRT language. Part (1) shows the software architecture of the real-time system. It contains the various tasks and the scheduler. This latter schedules the system following the Rate Monotonic priority assignment.

Task	Execution time	Deadline	Period
Task1	3 ms	4 ms	4 ms
Task2	5 ms	6 ms	6 ms
Task3	3 ms	7 ms	7 ms
Task4	2 ms	6 ms	6 ms
Task5	7 ms	8 ms	8 ms
Task6	2 ms	9 ms	9 ms

Table 8.2: Values of the task-set characteristics



Name	Task3
Process	Scheduler
Represented Activity	taskActivity3
Run On Processors	

Name	Scheduler
Processing Unit	processor1, processor2, processor3, processor4, processor5
Schedulable Tasks	Task1, Task6, Task4, Task2, Task3, Task5
Task Allocation	Partitioned
Scheduling Policy	Rate Monotonic

Name	multiCore
Processing Units	processor1, processor4, processor5, processor2, processor3
MultiProcessor Category	Homogeneous

Name	Trigger1
Periodicity	Periodic (value=4.0, unit=ms)
Scheduling Activities	taskActivity1
Release Time	Release Time (value=0.0, unit=ms)

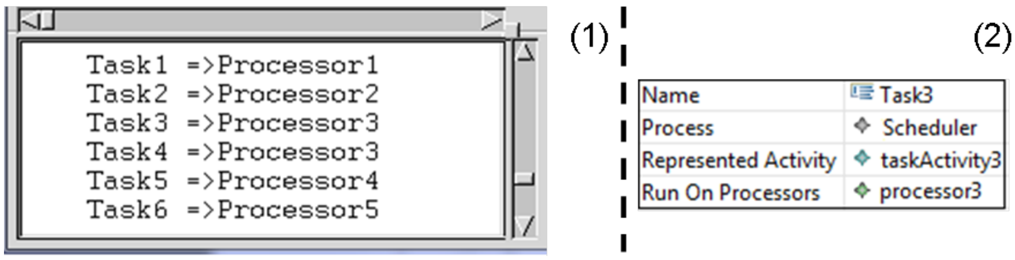
Deadline	Deadline(value=7.0, unit=ms)
Execution Time	Execution Time(value=3.0, unit=ms)
Name	taskActivity3
Represented Task	Task3
Preemptibility	Preemptible
Self Suspension	Not Self Suspended
Trigger	Trigger3

Figure 8.9: Model views expressed in the MoSaRT design language representing the studied real-time system

The tasks are not allocated to processors yet. Moreover, the tasks allocation would follow the partition policy. Part (2) represents a multiprocessor architecture containing five homogeneous cores. The behavioral model of this real-time system is shown in part (3). It is composed of a set of triggers and task activities. Every task activity represents the behavior of each task (existing in the software model), and is characterized by a deadline and a worst-case execution time. Every task activity is triggered by a time trigger. This latter is characterized by a periodic release date. Every time value is referring to the root trigger that plays the role of the event which is starting the real-time system.

IV.2.b. Analysis aspect

The next step consists in launching the identification process in order to check which timing analysis tests match the design. The test can be a dimensioning test or a feasibility test. At this step, the MoSaRT analysis repository orients the designer to Cheddar tool in order to allocate the task set to the set of processors. Part (1) of Figure 8.10 shows the result that is provided by Cheddar. The tasks allocation provided by Cheddar corresponds to the application of the "Next Fit" heuristic [DL78]. This analysis result should be taken into consideration by the design model. So, Part (2) of Figure 8.10 shows an excerpt of "Task3" properties after being allocated to "processor3" (dimensioning test).



```
Task1 =>Processor1
Task2 =>Processor2
Task3 =>Processor3
Task4 =>Processor3
Task5 =>Processor4
Task6 =>Processor5
```

Name	Task3
Process	Scheduler
Represented Activity	taskActivity3
Run On Processors	processor3

Figure 8.10: (1): Tasks allocation provided by Cheddar. (2): Excerpt of Task3 properties after the partition phase

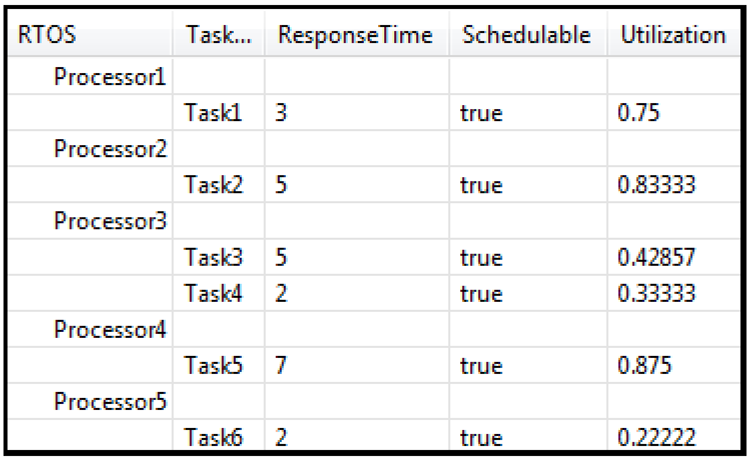
At this stage, the designer can solicit the analysis repository for a second time. As a result the repository should suggest analysis tests which are based on some identification rules. For instance, "implicit deadlines", "periodic tasks", "fixed priority scheduling", "tasks are allocated", "architecture is multiprocessor", etc. Then, the result of the identification process proposes some tests, the tools that provide these tests and the transformations to these tools. Figure 8.11 shows the result of the response time analysis provided by the RT-Druid analysis tool. This tool is called after transforming the MoSaRT model (after the processor allocation) to the RT-Druid input formalism.

IV.3. Case study 3

In this case study, we illustrate the model transformation from AADL to the MoSaRT design language, and we highlight the modeling and scheduling analysis capabilities of the MoSaRT framework. As an example, we have chosen to design a simplified mini UAV (Unmanned Aerial Vehicle) [TRA07].

IV.3.a. System description

The UAV is an aircraft able to fly and perform a mission without human presence on board. The practical application of this system is based on a set of interactions and communications between the aircraft and the ground station, where the system application embedded on the aircraft represents a



RTOS	Task...	ResponseTime	Schedulable	Utilization
Processor1				
	Task1	3	true	0.75
Processor2				
	Task2	5	true	0.83333
Processor3				
	Task3	5	true	0.42857
	Task4	2	true	0.33333
Processor4				
	Task5	7	true	0.875
Processor5				
	Task6	2	true	0.22222

Figure 8.11: Scheduling analysis results provided by RT-Druid analysis tools

critical point because of its resource limitations and aerodynamics constraints. In this case study we are interested in the assisted mode: in this mode, the operator gives a high level order to the UAV, like an angle of roll, a ground speed, and a vertical speed. The flight control system is embedded and is in charge of the control of the surfaces and engine in order to comply to the operator orders. Figure 8.12 represents different sensors and actuators related to the flight control and the surrounding functions.

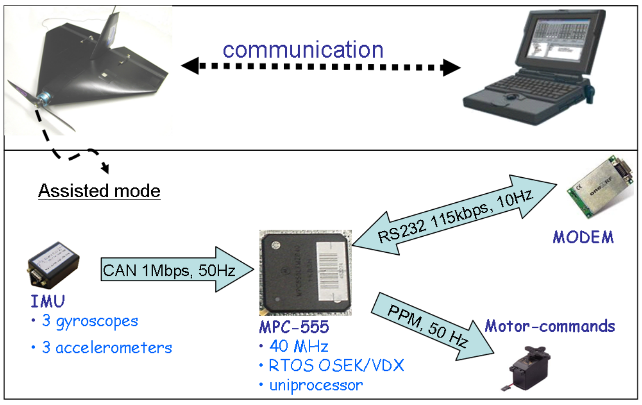


Figure 8.12: Different UAV components and their interactions during the assisted mode

The used components of the embedded system are:

- A modem for the transmission and reception of messages.
- An Inertial Measurement Unit, IMU, to acquire the attitude (roll, pitch, yaw) of the UAV.
- The servomotors are rotary actuators which help to control the angular position of the aileron-s/elevators.

- The micro-controller: an uniprocessor using the OSEK/VDX operating system. It is the main element of this architecture. It reads in parallel the information coming from different sensors, treats them and sends the commands to the actuators. Equipments connected to the microcontroller operate at different rhythms within specific time constraints. For example, the modem receiver connected to the serial port operates at a frequency of 10Hz, and sends its data to the microcontroller byte after byte at a rate of 115 kbits per second. The set of bytes transmitted during a period represents a frame. The microcontroller must read each byte before the arrival of the next byte to avoid the risk of losing the entire frame. The IMU has the same type of requirements. Similarly, the system has to refresh the commands sent to the actuators at a period of 20ms. These commands are sent as PPM signals (Pulse Position Modulation).

Table 8.3 summarizes the different properties of the task set. T is the period of a task, WCET its worst-case execution time, D its deadline and P its priority. (xN) assigned to “imuAcquisition” and “modemAcquisition” tasks means that the task has to be executed N times during its period to receive N CAN frames or N bytes. All the tasks are executed on a processor using a fixed priority scheduling policy. Resources are handled by the priority ceiling protocol.

Task	T	WCET	D	P	Shared data access time		
					PPM	Attitude	Instructions
imuAcquisition (x3)	20000	96	360	3	--	--	--
imuAcquisition_Regulate	20000	16000	18000	4	1500	50	50
modemAcquisition (x10)	100000	12	80	1	--	--	--
modemAcquisitionEnd	100000	240	80000	9	--	--	50
motorCommand	20000	1000	20000	6	500	--	--
transmission_to_Station	100000	3360	50000	7	--	50	--

Table 8.3: Values in microseconds (us) of task characteristics

IV.3.b. System conception using AADL

Figure 8.13 represents an excerpt of the architecture of the studied system. It shows the different tasks and their communication relationships. We present only a part of the model, expressed in the textual concrete syntax of AADL (see Listing 8.2) to illustrate some relevant properties. We use the property *First_Dispatch_Time* to represent the tasks which have to be executed N times during their periods. In our system, *First_Dispatch_Time* of Task$_{i+1}$ $\geq$ *Deadline* of Task$_i$. For example, *First_Dispatch_Time* of imuAcqTask1 is 0 us, *First_Dispatch_Time* of imuAcqTask2 is 360 us and *First_Dispatch_Time* of imuAcqTask3 is 720 us.

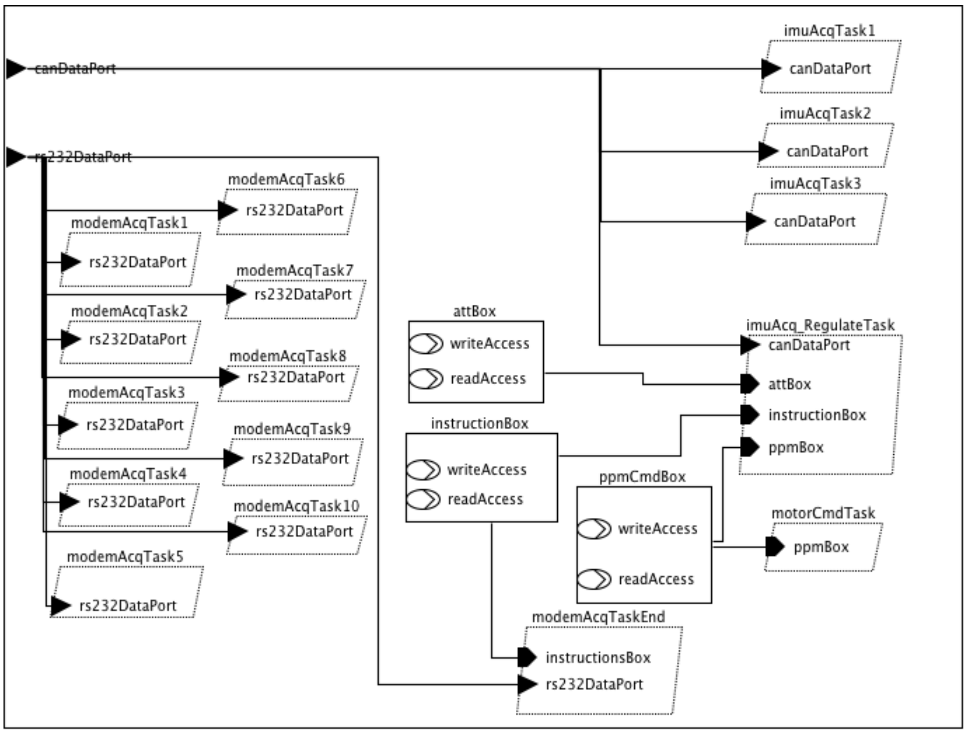


Figure 8.13: Part of the model of the case study example expressed in the AADL graphical concrete syntax

Listing 8.2: AADL model of the case study example

```
system uav

end uav;
.........
 thread implementation imuAcquisition_Regulate.impl
 properties
   Priority => 4;
   Dispatch_Protocol => periodic;
   Period => 20000us;
   Compute_Execution_Time => 16000us .. 16000us;
   Deadline => 18000us;
end imuAcquisition_Regulate.impl;
.........
process implementation scheduler.impl
 subcomponents
 imuAcqTask1 : thread imuAcquisition.impl
 {First_Dispatch_Time => 0us;};
 imuAcqTask2 : thread imuAcquisition.impl
 {First_Dispatch_Time => 360us;};
.........
 imuAcq_RegulateTask : thread imuAcquisition_Regulate.impl
 {First_Dispatch_Time => 1080us;};
 modemAcqTask1 : thread modemAcquisition.impl
 {First_Dispatch_Time =>0us;};
.........
```

```
 modemAcqTask10 : thread modemAcquisition.impl
 {First_Dispatch_Time =>720us;};
 modemAcqTaskEnd : thread modemAcquisitionEnd.impl
 {First_Dispatch_Time =>800us;};
.........
end scheduler.impl;
system implementation uav.impl
 subcomponents
     cpu : processor cpu;
     Proc : process scheduler.impl;
.........
 properties
 Actual_Processor_Binding =>
 (reference (cpu)) applies to Proc;
 end uav.impl;
```

IV.3.c. Transformation to the MoSaRT design language

The transformation from AADL to the MoSaRT design language is done in ATL [JK05] as shows in Figure 8.14. Indeed, in order to get an output file model (e.g. "InstanceExample.mosart") corresponding to the meta-model of the MoSaRT language (represented by "MoSaRT.ecore" file), the transformation process implemented by ATL (e.g. "AADL2MoSaRT.atl" program file) needs four kinds of files. The three meta-models of (i) AADL models (represented by "Aadl2.ecore" file), (ii) AADL model instances (represented by "Instance.ecore" file) and (iii) the MoSaRT language. The fourth file is the model instance file (e.g. "InstanceExample.aaxl2").
Listing 8.3 represents an example of a forking operation transforming the `Thread` AADL element to `SoSchedulableTask` and `SbTaskActivity` MoSaRT language elements.

Listing 8.3: Example of a transformation rule using ATL

```
lazy rule Thread_To_SoSchedulableTask {
from th:INST!ComponentInstance(th.category=#thread)
using{listOfProperty :
Sequence(AADL2!PropertyAssociation)=
th.ownedPropertyAssociation;}
to st:MoSaRT!"MoSaRT::SoftwarePlatform
::SoftwareOperators::SoSchedulableTask"(
name <- th.name,
representedActivity<-thisModule.Thread_To_SbTaskActivity(th))
do {for (prop in listOfProperty) {
thisModule.PropertiesOfTask(prop,st);}}}
```

The design model expressed in the MoSaRT language and obtained after the transformation is shown in Figure 8.15. The part (a) of Figure 8.15 represents the software architecture of the system. The part (b) is the hardware architecture of the system which is uniprocessor. Then, the part (c) of Figure 8.15 shows the software behavior of the system, this part shows the temporal characteristics of tasks, their relationships and the way that each task is triggered. Part (d) will be discussed later. We admit that the model respects all structural rules, and we move to the context identification phase.

IV.3.d. Scheduling analysis phase

This step starts by using an instance of the MoSaRT analysis repository to run the identification process. The obtained result depends on the content and the richness of the analysis repository.

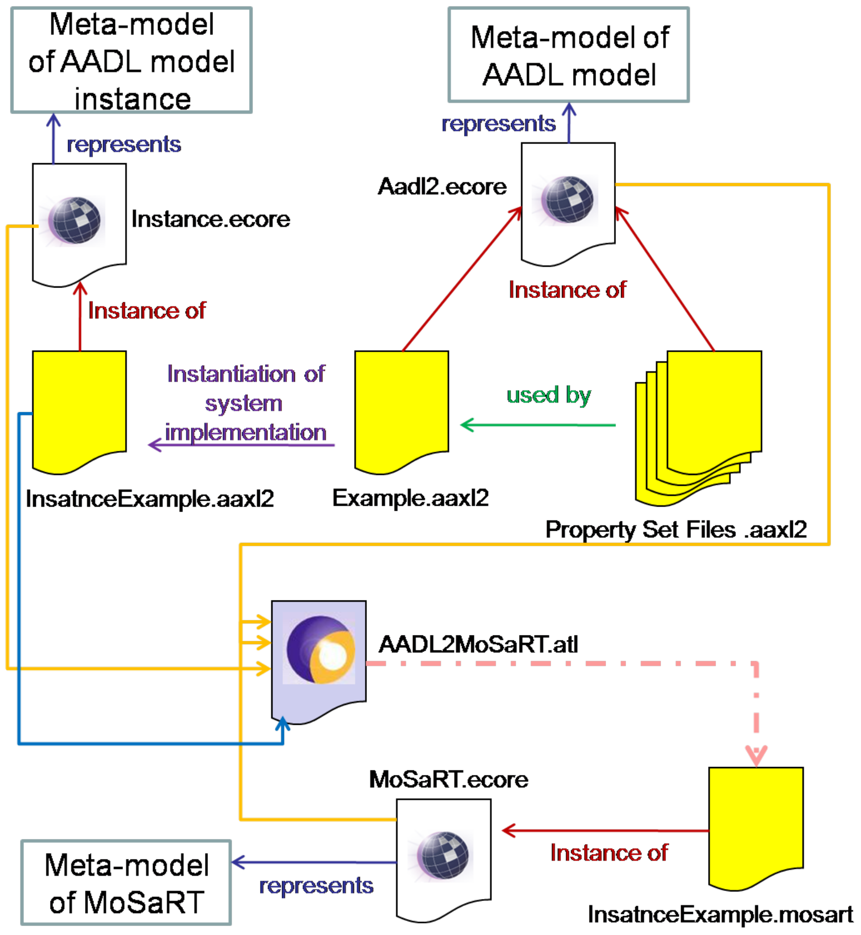
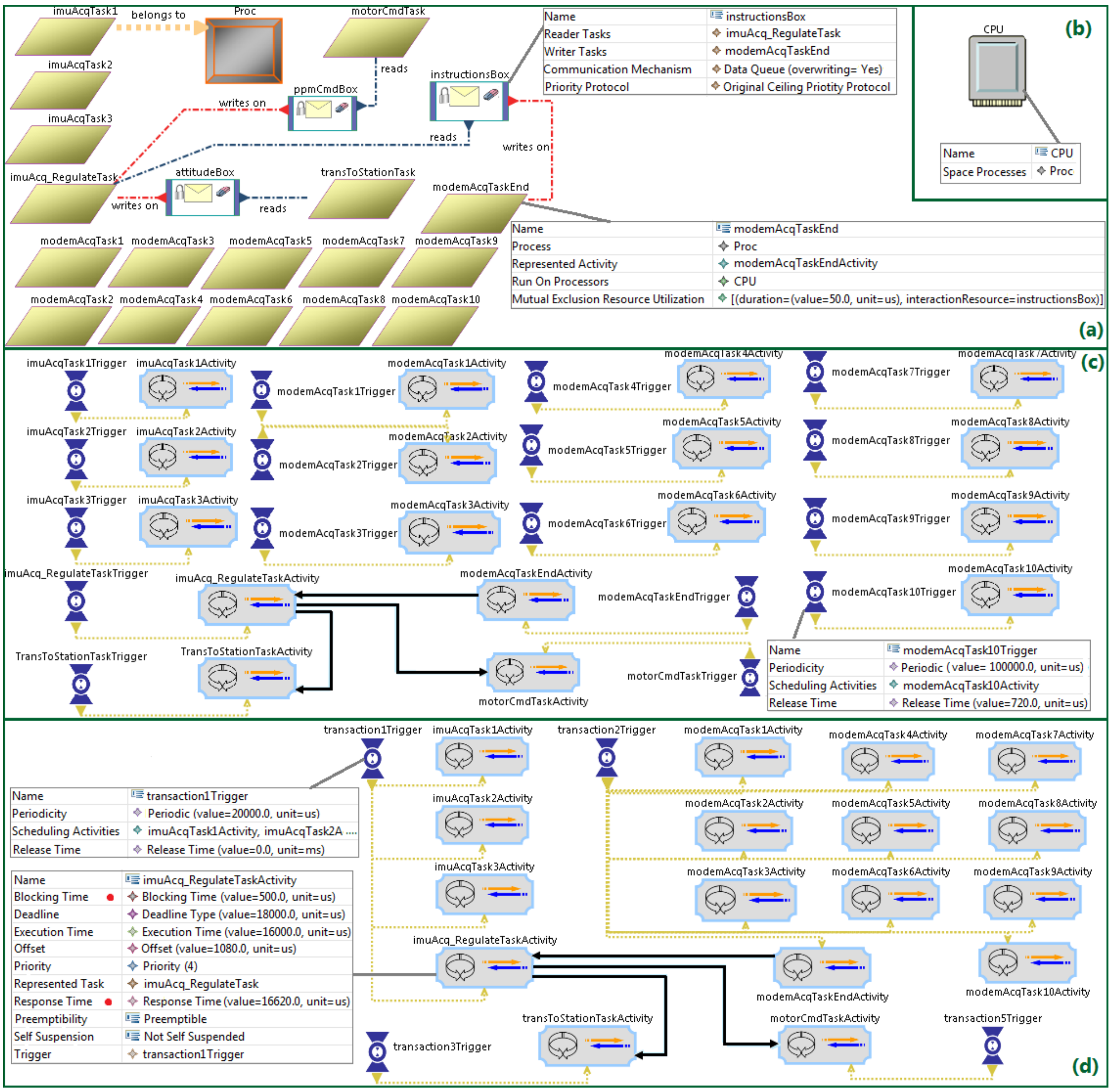


Figure 8.14: Architecture of the transformation process between AADL and MoSaRT

Figure 8.16 shows the result provided by the used repository. Part (a) of Figure 8.16 presents the task set characteristics extracted from the design model (Parts (a), (b) and (c) of Figure 8.15). Part (b) of Figure 8.16 presents the schedulability tests which correspond to the real-time context of the model and the scientific papers where designers can find more details about the proposed tests. This result also offers to the designers some tools which support the appropriate analysis techniques and transforms the model to different input formalism tools (see Part (c) of Figure 8.16).
Cheddar and MAST are among analysis tools proposed. After the transformation process to these tools, the result is illustrated in Parts (x) and (y) of Figure 8.17. The Mast result means that the model of our case study is not schedulable because the processor utilization is over 100%. However, the result provided by Cheddar (Part (y)) means that the Cheddar tool is not able to propose a test supporting the temporal characteristics of the model.
In this case, a designer can try to change the task set characteristics or to modify the hardware architecture by using a multicore system for example.
While the first choice requires an additional development time, the choice of adding a processor can



Name	instructionsBox
Reader Tasks	imuAcq_RegulateTask
Writer Tasks	modemAcqTaskEnd
Communication Mechanism	Data Queue (overwriting= Yes)
Priority Protocol	Original Ceiling Priotity Protocol

Name	CPU
Space Processes	Proc

Name	modemAcqTaskEnd
Process	Proc
Represented Activity	modemAcqTaskEndActivity
Run On Processors	CPU
Mutual Exclusion Resource Utilization	[(duration=(value=50.0, unit=us), interactionResource=instructionsBox)]

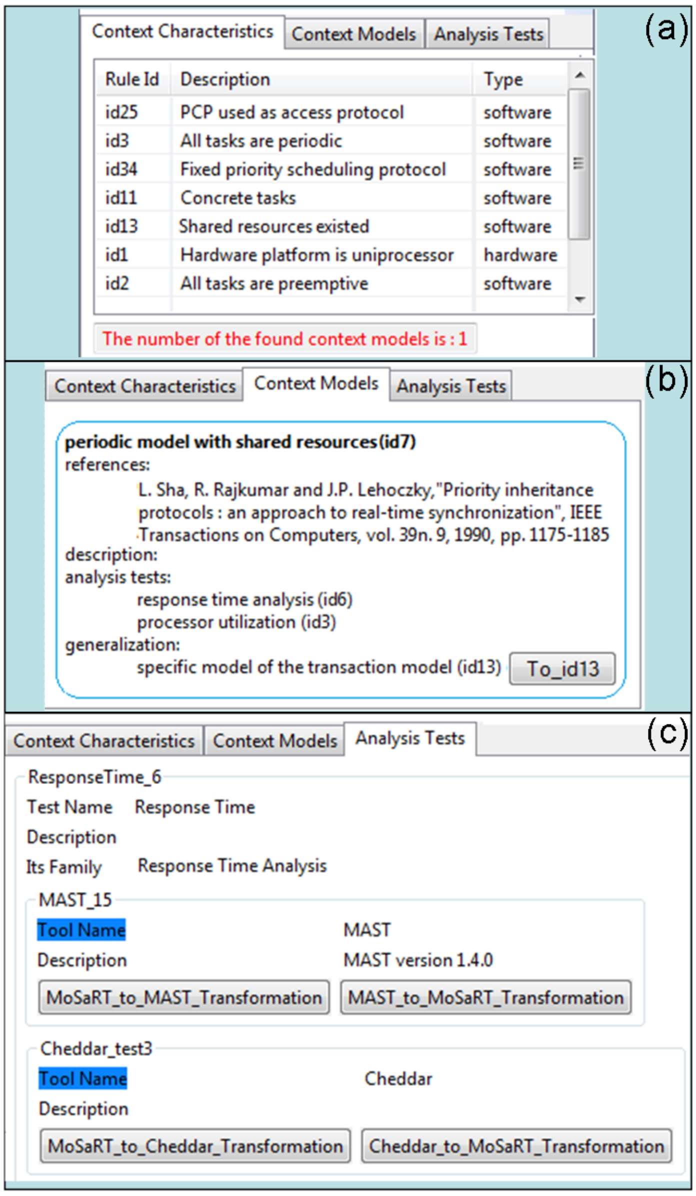
Name	modemAcqTask10Trigger
Periodicity	Periodic (value= 100000.0, unit=us)
Scheduling Activities	modemAcqTask10Activity
Release Time	Release Time (value=720.0, unit=us)

Name	transaction1Trigger
Periodicity	Periodic (value=20000.0, unit=us)
Scheduling Activities	imuAcqTask1Activity, imuAcqTask2A ....
Release Time	Release Time (value=0.0, unit=ms)

Name	imuAcq_RegulateTaskActivity
Blocking Time ●	Blocking Time (value=500.0, unit=us)
Deadline	Deadline Type (value=18000.0, unit=us)
Execution Time	Execution Time (value=16000.0, unit=us)
Offset	Offset (value=1080.0, unit=us)
Priority	Priority (4)
Represented Task	imuAcq_RegulateTask
Response Time ●	Response Time (value=16620.0, unit=us)
Preemptibility	Preemptible
Self Suspension	Not Self Suspended
Trigger	transaction1Trigger

Figure 8.15: Parts (a), (b) and (c) represent an extract from the MoSaRT model obtained via the transformation process from AADL model. Part (d) represents an extract from the refined model [The refinement is realized by a transformation from MoSaRT to MoSaRT]

be an over-dimensioning solution. However, we note that a generalized task model is proposed by the analysis repository (parts (b) of Figure 8.16). This means that the design model can be refined in order to be supported by this generalized model. In this case, the refinement is not based on adding more specifications, but it is related to a semantic specification, i.e. the behavior of the model can be



Rule Id	Description	Type
id25	PCP used as access protocol	software
id3	All tasks are periodic	software
id34	Fixed priority scheduling protocol	software
id11	Concrete tasks	software
id13	Shared resources existed	software
id1	Hardware platform is uniprocessor	hardware
id2	All tasks are preemptive	software

The number of the found context models is : 1

periodic model with shared resources(id7)
references:
L. Sha, R. Rajkumar and J.P. Lehoczky,"Priority inheritance protocols : an approach to real-time synchronization", IEEE Transactions on Computers, vol. 39n. 9, 1990, pp. 1175-1185
description:
analysis tests:
response time analysis (id6)
processor utilization (id3)
generalization:
specific model of the transaction model (id13)

ResponseTime_6
Test Name Response Time
Description
Its Family Response Time Analysis

MAST_15
Tool Name MAST
Description MAST version 1.4.0

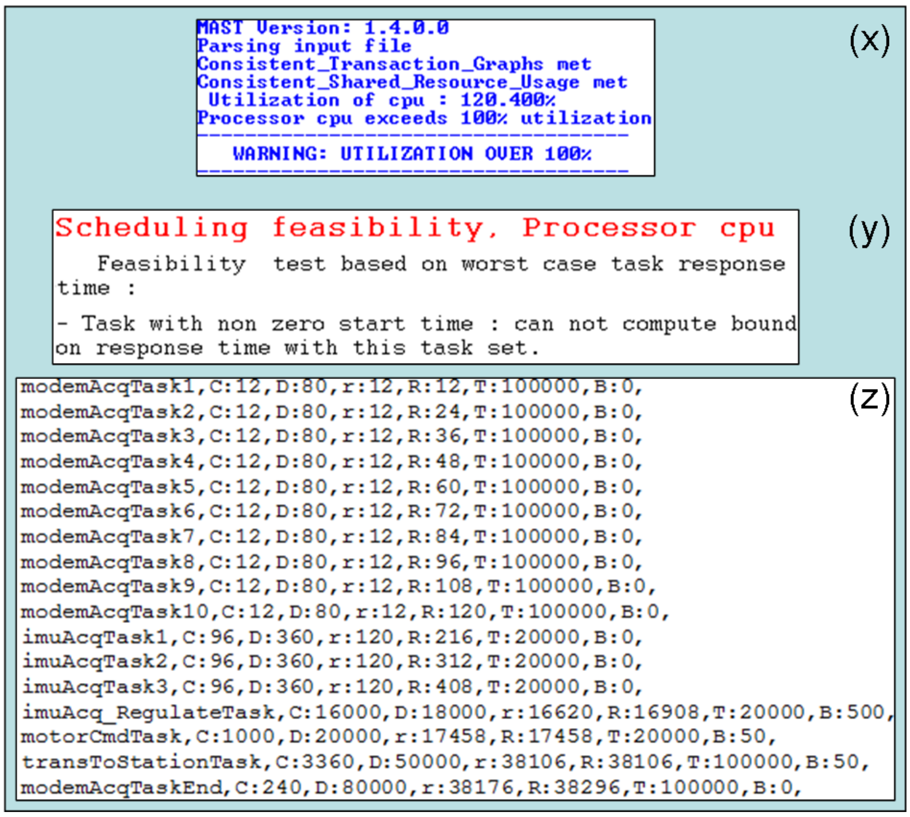
Cheddar_test3
Tool Name Cheddar
Description

MoSaRT_to_Cheddar_Transformation
Cheddar_to_MoSaRT_Transformation

Figure 8.16: The result provided by MoSaRT Analysis Repository after applying the model shown in Parts (a), (b) and (c) of Figure 8.15

viewed differently by maintaining the same architecture. The used analysis repository proposes this refinement process based on an endogenous transformation.
An excerpt of the refinement result is illustrated on Part (d) of Figure 8.15. The new model maintains the same hardware and software architecture. The sole impact is related to the behavioral side. Thus, the new model is composed from Parts (a), (b) and (d) of Figure 8.15. Due to this refinement, the independent tasks having the same period are regrouped in one transaction. Hence, we obtained an

offset-based model instead of having a periodic model. The new model is composed of four transactions. Moreover, the analysis of the new model gives an interesting result. It is provided by a third-party offset analysis tool based on Tindell's work [TC94]. Part (z) of Figure 8.17 shows the analysis result of the refined model. "C" is the worst-case execution time, "D" represents the relative deadline, "T" is the period and "B" is the blocking time related to the use of protected shared resources. "r" corresponds to the response time and "R" corresponds to the original response time. The comparison between "r" and "R" values shows a significant difference related to the pessimism of the periodic model compared to the transaction model.



```
MAST Version: 1.4.0.0
Parsing input file
Consistent_Transaction_Graphs met
Consistent_Shared_Resource_Usage met
 Utilization of cpu : 120.400%
Processor cpu exceeds 100% utilization
-------------------------------------
   WARNING: UTILIZATION OVER 100%
-------------------------------------
```

```
Scheduling feasibility, Processor cpu
   Feasibility test based on worst case task response
time :

- Task with non zero start time : can not compute bound
on response time with this task set.
```

```
modemAcqTask1,C:12,D:80,r:12,R:12,T:100000,B:0,
modemAcqTask2,C:12,D:80,r:12,R:24,T:100000,B:0,
modemAcqTask3,C:12,D:80,r:12,R:36,T:100000,B:0,
modemAcqTask4,C:12,D:80,r:12,R:48,T:100000,B:0,
modemAcqTask5,C:12,D:80,r:12,R:60,T:100000,B:0,
modemAcqTask6,C:12,D:80,r:12,R:72,T:100000,B:0,
modemAcqTask7,C:12,D:80,r:12,R:84,T:100000,B:0,
modemAcqTask8,C:12,D:80,r:12,R:96,T:100000,B:0,
modemAcqTask9,C:12,D:80,r:12,R:108,T:100000,B:0,
modemAcqTask10,C:12,D:80,r:12,R:120,T:100000,B:0,
imuAcqTask1,C:96,D:360,r:120,R:216,T:20000,B:0,
imuAcqTask2,C:96,D:360,r:120,R:312,T:20000,B:0,
imuAcqTask3,C:96,D:360,r:120,R:408,T:20000,B:0,
imuAcq_RegulateTask,C:16000,D:18000,r:16620,R:16908,T:20000,B:500,
motorCmdTask,C:1000,D:20000,r:17458,R:17458,T:20000,B:50,
transToStationTask,C:3360,D:50000,r:38106,R:38106,T:100000,B:50,
modemAcqTaskEnd,C:240,D:80000,r:38176,R:38296,T:100000,B:0,
```

Figure 8.17: Parts (x) and (y) are the results provided by MAST and Cheddar tools, these results are related to the model of Figure 8.15 (Parts a, b and c). Part (z) is provided by Tindell tool and it is related to the refined model of Figure 8.15 (i.e. Parts a, b and d)

IV.3.e. Restitution of the analysis result

Once the scheduling analysis phase is finished, we move to the restitution phase. That starts by enriching the design model expressed in the MoSaRT language through its output properties like "blocking time" and "response time" (see Part (d) of Figure 8.15) (the model enrichment consists of transforming the output formalism of the analysis result to the MoSaRT language). Then, we run the reverse transformation from the MoSaRT language to AADL.

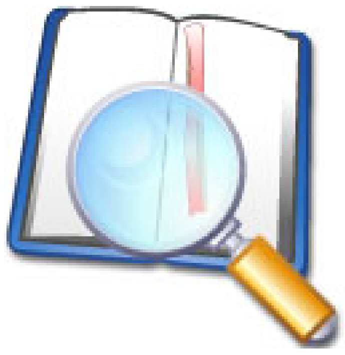
Listing 8.4 represents a piece of the adjusted model. The transaction is interpreted as a "thread group" element. Each "thread group" contains a set of tasks having the same period and being independent from each other. Thus, we only changed the semantic view of the model without modifying the architecture of the UAV control system.
The $First\_Dispatch\_Time$ property has been converted into $Dispatch\_Offset$.
$Worst\_Case\_Response\_Time$ and $Blocking\_Time$ have been added in the AADL model as ad-hoc properties through the extension mechanism to refine scheduling metrics.

Listing 8.4: AADL model adjusted and produced by MoSaRT after being analyzed

```
.........
thread group transaction1
features
canDataPort : in data port;
attBox : requires data access attitudeBox.impl;
ppmBox : requires data access ppmCmdBox.impl;
instructionsBox : requires data access instructionsBox.impl;
end transaction1;
.........
thread group implementation transaction1.impl
subcomponents
 imuAcqTask1 : thread imuAcquisition.impl
 {Dispatch_Offset => 0us;
        Worst_Case_Response_Time => 120us;};
 imuAcqTask2 : thread imuAcquisition.impl
 {Dispatch_Offset => 360us;
        Worst_Case_Response_Time => 120us;};
 imuAcqTask3 : thread imuAcquisition.impl
 {Dispatch_Offset => 720us;
 Worst_Case_Response_Time => 120us;};
 imuAcq_RegulateTask : thread imuAcquisition_Regulate.impl
 {Dispatch_Offset => 1080us;
 Worst_Case_Response_Time => 16620us;
 Blocking_Time=> 500us;};
 .........
 properties
 Period => 20000us;
 end transaction1.impl;
.........
process implementation scheduler.impl
 subcomponents
 transaction1 : thread group transaction1.impl;
 transaction2 : thread group transaction2.impl;
 transaction3 : thread group transaction3.impl;
 transaction4 : thread group transaction4.impl;
.........
end scheduler.impl;
.........
```

V. Conclusion

In this chapter we have presented the MoSaRT framework which is a result of matching the MoSaRT analysis repository to the MoSaRT design language. We have also presented the different capabilities related to the usage of the MoSaRT framework, and we have focused on how that can help designers in order to cope with the difficulty of the scheduling analysis phase. The objective of this work is to benefit from analyst skills and designer skills in order to unify their efforts, then to avoid wrong design choices at an early design stage. This unification can be done through the MoSaRT framework while maintaining the independence between the design modeling area and the system analysis area. The usefulness and the feasibility of the MoSaRT framework have been also highlighted through several case studies. Technical solutions which are used to implement the MoSaRT framework are presented in Appendix B.



Part III

Conclusion and Perspectives

Conclusions

In this thesis, we presented the modeling and validation of real-time systems in terms of schedulability and dimensioning for analyses during the design phase. This thesis is related to different research areas: real-time scheduling theory, design of real-time systems and model-driven engineering. Thus, the cornerstone contribution of this thesis is the use of model-driven engineering in order to ease the usage of the real-time scheduling theory while designing real-time systems.
In research foundations part of this report, we have introduced the software development process of real-time systems by focusing on the design and analysis phases and their relevance to produce a correct product and to reduce the time-to market (see Chapter 2). The difficulty which faces the enhancement of the design and analysis of real-time systems is not only related to the existence of different actors with different backgrounds, but also related to the lack of an easy way to transfer the real-time scheduling theory results from academia to industrial practice.
We have presented some notions of real-time scheduling theory and their importance to analyze design models (see Chapter 3). Since we focus on the model-driven development of real-time systems, Chapter 4 is devoted to the presentation of some concepts related to the model-driven engineering. Moreover, we have also presented AADL and UML-MARTE as standard design languages which enable to build design models for analysis purposes. Brief descriptions of several analysis tools providing some implementations of analysis tests have also been highlighted.
To well define our needs, the second part of the manuscript started by synthesizing the state of art and presenting the limitations, problems and objectives motivating our proposals (see Chapter 5). We summarize those problems in two points. (1) Existing model-driven solutions (design languages and tools) are based on classical abstract models which are not accurate enough. Even if they ease the use of some scheduling analysis techniques and automatize the computation process, this usage concerns only of a very limited subset of analysis techniques, and it can lead to a pessimism and over-dimensioning. (2) There exists several research works with interesting result which can allow designers to improve their designs. Unfortunately, only few research works have been exploited by the industry. Therefore, the following contributions make up for lacks and respond to the needs.

1. The MoSaRT design language (see Chapter 6)

   - In this thesis, we have presented the MoSaRT design language in order to allow designers to design the software part of real-time systems for scheduling analysis purposes. Since our objective is to fill up the chasm related to the use of other design languages, the MoSaRT design language is proposed to be very close to the scheduling analysis area. The main differences between MoSaRT and other standard design languages are:
     - The way MoSaRT allows to design a real-time system: unlike other design languages, the MoSaRT design language starts by defining the operational model, which can be refined by the functional model. Therefore, the MoSaRT design language can be used by different actors (modelers, architects, analysts).
     - The MoSaRT design language capability to support most kind of formats of real-time properties: concepts are generic, this enables an easy extension, and a sustainable solution to cover a large number of real-time contexts.
     - The MoSaRT design language capability to verify if a real-time system can be analyzable: a set of structural rules has been injected, to check the ability of the design model for scheduling analysis.
   - Discussion: Since our objective is not to propose "yet a new design language", it is useful to compare the MoSaRT design language with standard design languages.

Characteristics	AADL	MARTE	MoSaRT lang.
Enables functional model	☒	$\surd$	$\surd$
Does not require functional model	$\surd$	☒	$\surd$
Stores analysis result	☒	$\surd$	$\surd$
Supports different kinds of embedded systems	$\surd$	$\surd$	☒
Checks the structural correctness	☒	☒	$\surd$
Expressiveness and semantic distinctions of theoretical task-set	☒	☒	$\surd$

Table 9.1: Comparison of AADL, MARTE and MoSaRT language according to some characteristics

   Table 9.1 shows the comparison between AADL, MARTE and the MoSaRT design language according to several criteria. When the design language support the characteristic we note $\surd$, otherwise we note ☒. The goal of this comparison is not to highlight capabilities of each design language, but to justify the interest of using our work via a tool-chain driven development of real-time systems.

   As the goal is to use the MoSaRT design language as a pivot language (only for real-time scheduling analysis), the design composition has not been treated. So, one can transform only the component on which one is interested to the MoSaRT language.

   - Perspectives: beside elements extension to cover complex systems (especially distributed systems) and also experimental results at an industrial level, in this thesis we have not detailed the model transformation from AADL or MARTE to the MoSaRT design language. The current transformations are done in a traditional manner based on ATL transformation language. The transformation process is ad-hoc and it will probably not satisfy all users of a design language such as MARTE, which is used in different methodologies. An interesting research direction is to construct a domain specific transformation language, where its instantiation can generate a model transformation (for instance expressed in ATL) depending on the methodology adopted.

2. The MoSaRT analysis repository (see Chapter 7)

   - In this thesis, we also proposed the MoSaRT analysis repository which orients designers to choose the most adapted analysis tests, and also increases the applicability of the real-time scheduling theory. The MoSaRT analysis repository is represented as an open metamodel relying on model-driven engineering. The underlying idea behind the proposed metamodel is to allow a research group to easily instantiate or enrich an existing instance of the analysis repository by adding a new real-time context and related schedulability tests. Thus, the applicability of the real-time scheduling theory can be increased.
   - Discussion: as far as we know, it does not exist any work completely similar to ours. Although analysis tools provide several analysis tests, they represent a particular instance of the MoSaRT analysis repository. While the MoSaRT analysis repository can be used with design languages such as AADL or UML-MARTE, it can help users to set their own methodologies related to the standard languages (e.g. Optimum methodology related to UML-MARTE). In this case, the content of the analysis repository instance will be based only on one context. In the examples shown in this thesis, we do not emphasize the integration of off-line analysis methods during the instantiation of the analysis repository. Obviously, the concept of real-time context is also valid for real-time off-line contexts, hence it is possible to integrate in the repository analysis instance analysis tools based on Petri net formalism, linear temporal logic, etc.
   - Perspectives: the richer the repository instance is, the more accurate is the result (if any real-time context is proposed). It means that the instance should contain a large number of identification rules. However, we have shown that modifying contexts already existing in the repository instance can be tedious. A possible way to avoid this problem is to take the semantics of the identification rules into consideration. This could be expressed as rule relationships. For instance, contradiction relation, extension relation, cohabitation relation, etc. That may also have an impact on the identification process. Therefore, instead of checking every identification rule, we can imagine an identification process based on a decision tree (see Figure 9.1).

     For example, the identification process first checks if a hardware platform exists. If true, then it will verify if the hardware architecture is uniprocessor, else it will verify if the hardware architecture is multi-processor, else it will deduce that the hardware architecture is distributed. By following this way of identification, every tree exploration can lead to schedulability/dimensioning analysis tests. This kind of analysis can prune several rules, hence it would accelerate the identification process.

3. The MoSaRT framework (see Chapter 8)

   - The objective behind the MoSaRT framework is to benefit from the analyst skills and

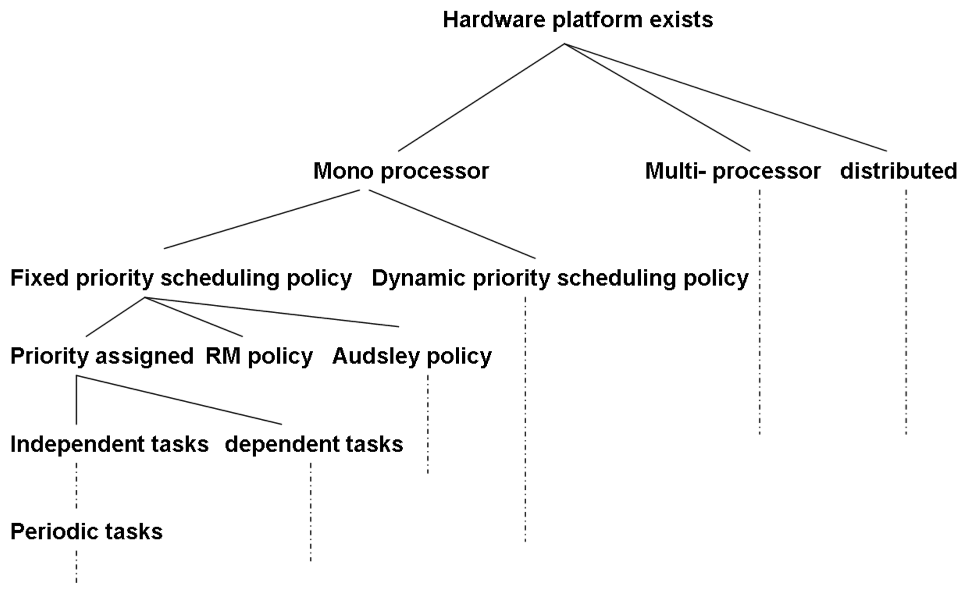


Figure 9.1: A decision tree of identification rules

designer skills in order to unify their efforts, then to avoid wrong design choices at an early design phase. The framework is based on the MoSaRT design language and the MoSaRT analysis repository, and presents a helpful modeling support for both designers and analysts. The contribution of this work can be quantified in different ways: academic and industrial. On the one hand, that can help real-time scheduling theory researchers to easily integrate their research works in tool chain based on a design methodology. On the other hand, designers can be oriented depending on the content of the analysis repository instance used during the analysis phase. So, the repository instance can have an impact on the design, for example the design can be refined or dimensioned.

We provided three case studies, where we have presented the usability of the MoSaRT framework. We stressed how the utilization of the MoSaRT framework can help designers in order to lead to efficient models at a very early design stage. We think that the gap between the research community and the industry is also due to different existing implementations, design languages or tools. For most existing implementations, the concepts are based on industrial use cases. By proposing MoSaRT framework, we have tried to have concepts based on theoretical studies and dedicated to a concrete industrial utilization.

- Discussion: referring to the questions raised in Chapter 5, the question “If the system is not schedulable, what needs to be changed to make it schedulable?” is not completely treated by the MoSaRT framework, because we have focused on the operational model. In case of non schedulability of the system, the MoSaRT framework proposes only a dimensioning analysis if it exists (it depends on the system context). However, some works like Optimum (applied to UML-MARTE) proposes to change down-right the system architecture if the functional model exists. Such work can be connected to the framework in order to be called after the restitution step (e.g. obtaining the analysis result).

- Perspectives: it will be helpful to generate a design expressed in MoSaRT language by choosing a priori the real-time context to which the design corresponds. In this case, the real-time context is used as a design pattern. For example, generating a design model respecting the pattern Ravenscar Profile which is widely used in the industry can be very interesting. Many designers prefer to use such design pattern during the modeling phase because they already know the compatible analysis tests which they have to apply during the analysis phase. Therefore, by instantiating the real-time contexts of the MoSaRT analysis repository, we can get an instance containing design patterns.

  Through the MoSaRT framework, we have used the MoSaRT analysis repository with the MoSaRT design language. The identification rules are expressed as OCL invariants, which depend on the MoSaRT design language. The utilization of the MoSaRT analysis repository with another design language (e.g. AADL) needs also to have rules expressed as OCL invariants depending on that design language. It will be interesting to develop a solution transforming the rules from OCL invariants depending on a language to OCL invariants depending on other languages in order to facilitate the exchange of repository instances related to different design languages.

Appendices

Examples of Instantiation

I. Some task models expressed in MoSaRT design language

In this appendix we present models for the multiframe model [MC96], the generalized multi-frame model [BCGM99], and the transaction model [RGRR12]. Those examples will be expressed in the MoSaRT design language.
In the following, we will focus only on the behavioral models of systems represented by the task models above-mentioned.

I.1. Example of multiframe model

A multiframe model enables to design tasks characterized by the variance of worst-case execution times. Indeed, the execution time of a task may vary from one instance to another, but the variation follows a known pattern. The multiframe model captures this variation by specifying the execution time of a task not as a single number, but as a finite list of numbers from which the execution time of successive instances of the task will be generated. This list of numbers will be repeated ad infinitum.

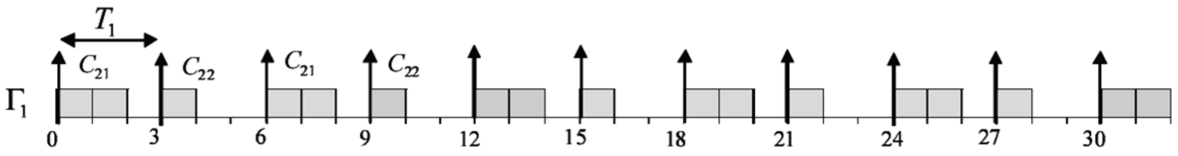
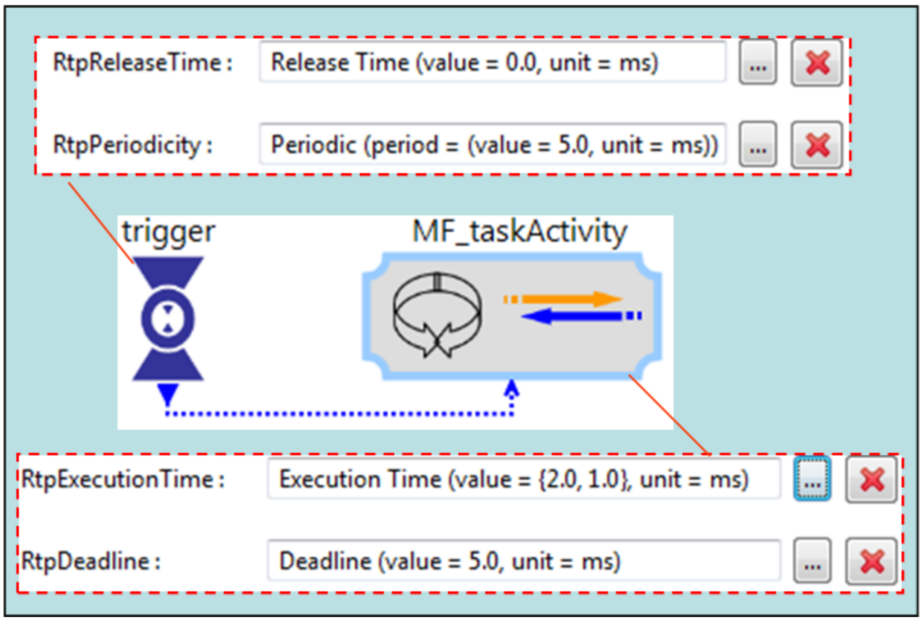


Figure A.1: Gantt diagram showing an example of a multiframe task

A multiframe real-time task is a tuple (Γ_i, T_i), where Γ_i is an array of N (N $\geq$ 1) frames (τ_{i1}, τ_{i2}, ..,τ_{iN}) with execution times (C_{i1}, C_{i2}, ..,C_{iN}) respectively. T_i is the minimum separation time, i.e., the ready times of two consecutive frames must be at least T_i time units apart. The deadline of each frame is T_i after its ready time.
Figure A.1 shows an example of a periodic multiframe task (Γ_1, T_1) = ((C_{11}, C_{12}),T_1) = ((2,1),3).

The same example is expressed in MoSaRT language (see Figure A.2), where the period is 3 ms, the deadline is 3 ms, and the execution times are 2 ms for odd frames and 1 ms for pair frames.



RtpReleaseTime : Release Time (value = 0.0, unit = ms)

RtpPeriodicity : Periodic (period = (value = 5.0, unit = ms))

RtpExecutionTime : Execution Time (value = {2.0, 1.0}, unit = ms)

RtpDeadline : Deadline (value = 5.0, unit = ms)

Figure A.2: MoSaRT behavioral model of a real-time system with a multiframe task

I.2. Example of generalized multiframe model

A generalized multiframe real-time task Γ_i is composed of N frames (τ_{i1}, τ_{i2}, ..,τ_{iN}). Every frame is characterized by an execution time C_{ij}, a deadline D_{ij} and the minimum time T_{ij} separating the activations of two successive frames.

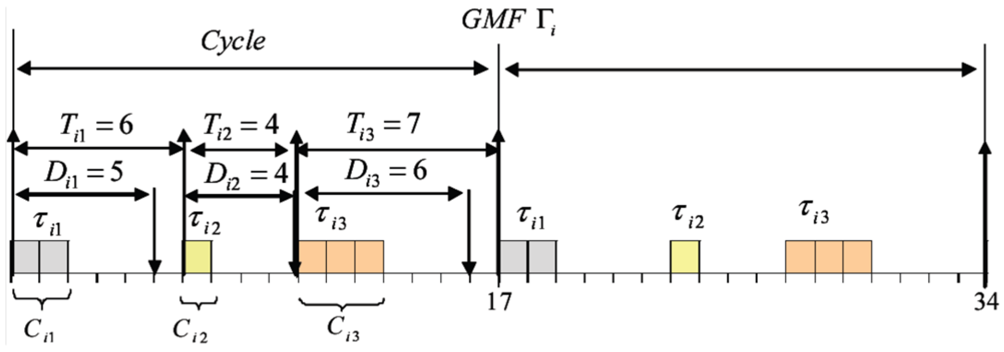
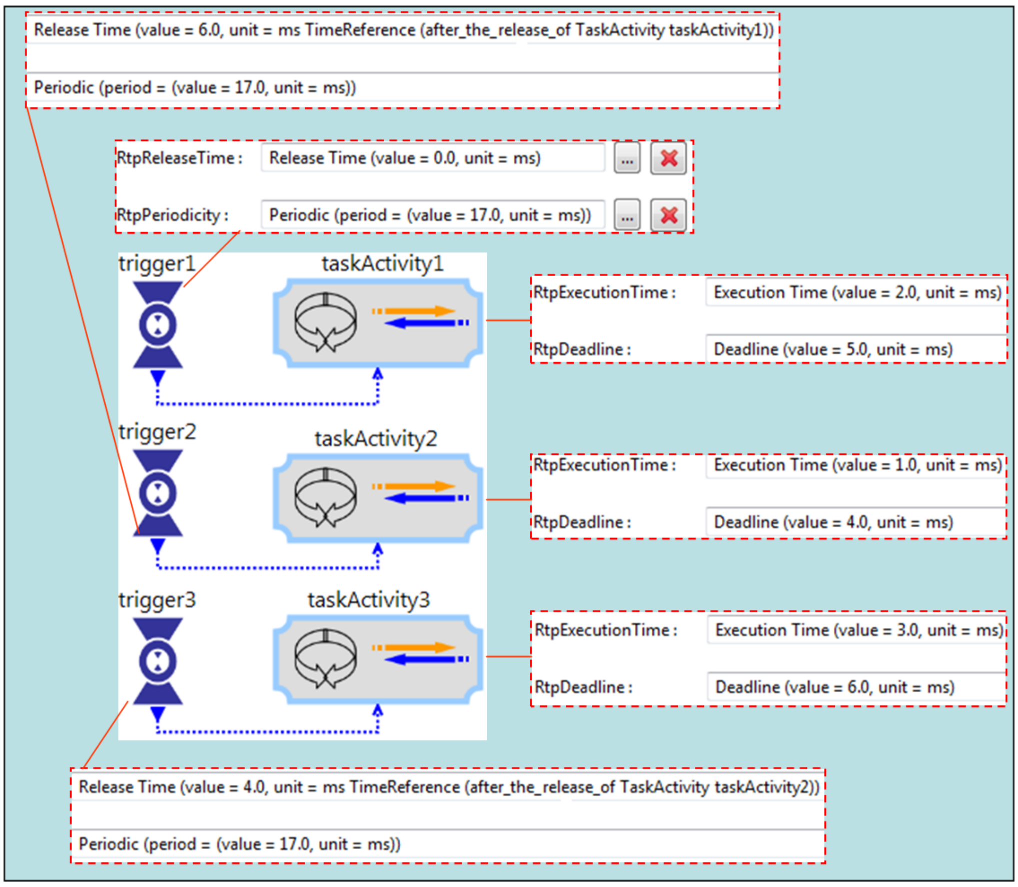


Figure A.3: Gantt diagram showing an example of a generalized multiframe task

Figure A.3 shows an example of a periodic generalized multiframe task Γ_1 containing three frames (τ_{11}, τ_{12}, τ_{13}). The first frame τ_{11} is characterized by an execution time $C_{11} = 2$ ms and a deadline $D_{11} =$

4 ms. τ_{12} is released 6 ms after the activation of τ_{11}.

Figure A.4 shows the behavioral part of the design corresponding to the periodic generalized multiframe task Γ_1. By using the MoSaRT design language, every frame is modeled as a separated task (e.g. $taskActivity1$ corresponds to τ_{11}). The release of $taskActivity2$ is related to the activation $taskActivity1$, and the release of $taskActivity3$ is related to $taskActivity2$. The three triggers ($trigger1$, $trigger2$ and $trigger3$) have the same period which corresponds to the cycle shown in Figure A.3. As in this example we have chosen to design a periodic generalized multiframe task with concrete release time, it is also possible to design sporadic and non-concrete generalized multiframe tasks.



Release Time (value = 6.0, unit = ms TimeReference (after_the_release_of TaskActivity taskActivity1))
Periodic (period = (value = 17.0, unit = ms))

RtpReleaseTime : Release Time (value = 0.0, unit = ms)
RtpPeriodicity : Periodic (period = (value = 17.0, unit = ms))

RtpExecutionTime : Execution Time (value = 2.0, unit = ms)
RtpDeadline : Deadline (value = 5.0, unit = ms)

RtpExecutionTime : Execution Time (value = 1.0, unit = ms)
RtpDeadline : Deadline (value = 4.0, unit = ms)

RtpExecutionTime : Execution Time (value = 3.0, unit = ms)
RtpDeadline : Deadline (value = 6.0, unit = ms)

Release Time (value = 4.0, unit = ms TimeReference (after_the_release_of TaskActivity taskActivity2))
Periodic (period = (value = 17.0, unit = ms))

Figure A.4: MoSaRT behavioral model of a real-time system with a generalized multiframe task

I.3. Example of transactions model

In this example we consider a lite version of the transaction model, where a real-time transaction Γ_i is composed of N tasks (τ_{i1}, τ_{i2}, ..,τ_{iN}) having the same period T_i. Every task τ_{ij} is defined by an execution-time C_{ij}, an offset (minimal time between the release of the transaction and the release of the task) O_{ij} and a deadline D_{ij} (i.e. in worst-case a task τ_{ij} must finish the execution at $O_{ij}+D_{ij}$).

A task may also be characterized by a jitter Jij.

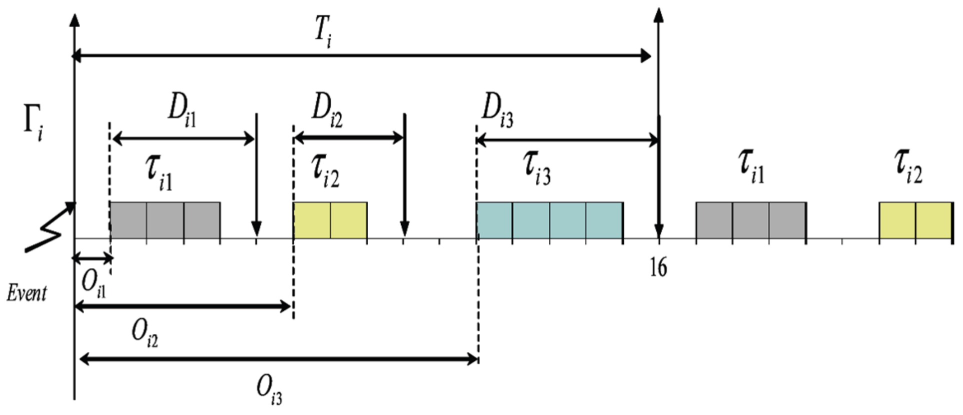
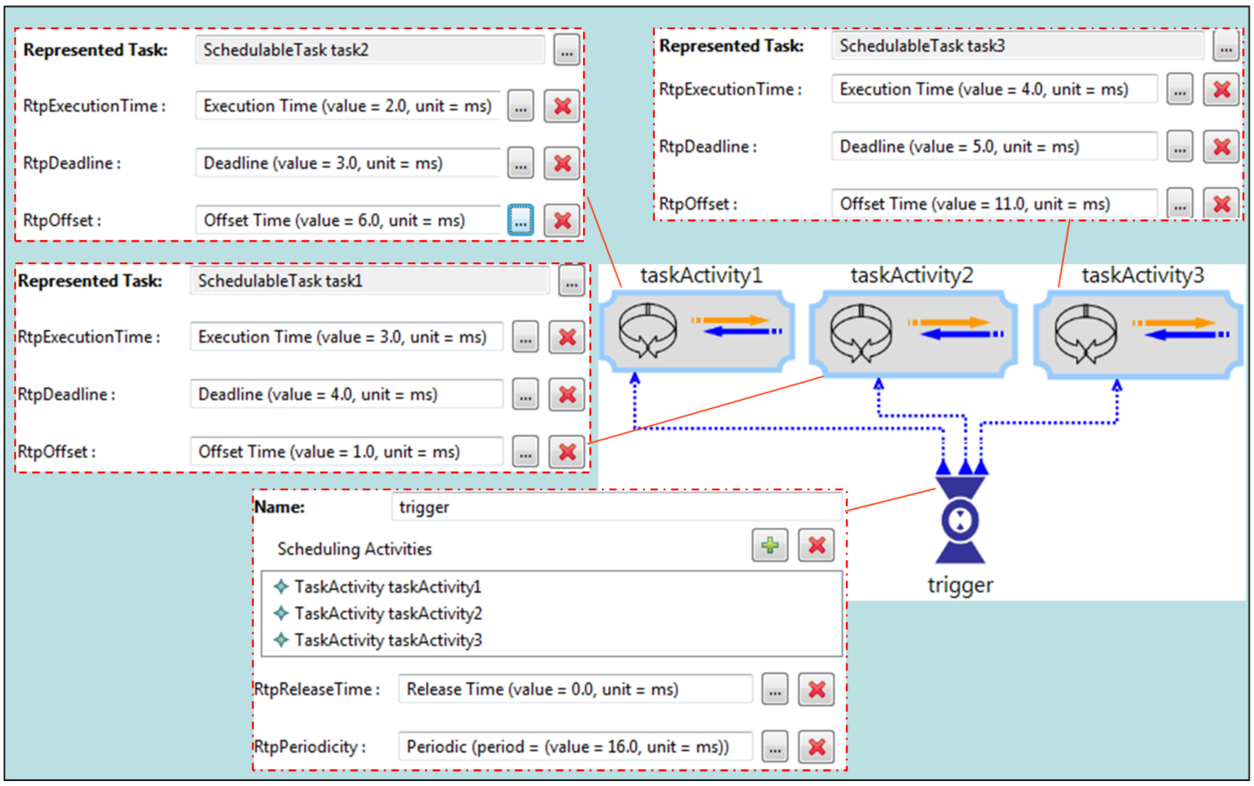


Figure A.5: Gantt diagram showing an example of a transaction

Figure A.5 depicts an example of a concrete periodic transaction Γ_i composed of tasks τ_{i1}, τ_{i2} and τ_{i3}. Such that, $C_{i1} = 3\ ms$, $C_{i2} = 2\ ms$ and $C_{i3} = 4\ ms$. The three tasks have the same period $T_i = 16$ ms. Moreover, $O_{i1} = 1\ ms$, $O_{i2} = 6\ ms$ and $O_{i3} = 11 ms$. Offsets are related to the release time of the transaction Γ_i.

Figure A.6 shows the behavioral part of the design corresponding to the example depicted in Figure A.5. The transaction is modeled as a set of tasks triggered by the same trigger called *trigger* and every task of the transaction is modeled as a separated task (e.g. $taskActivity1$ corresponds to τ_{i1}). Moreover, every task is characterized by a $RtpExecutionTime$, a $RtpDeadline$ and a $RtpOffset$. The activation pattern is defined by $RtpPeriodicity$. Since the trasaction is concrete, the release time is defined by $RtpReleaseTime$ property.



Represented Task:	SchedulableTask task2
RtpExecutionTime :	Execution Time (value = 2.0, unit = ms)
RtpDeadline :	Deadline (value = 3.0, unit = ms)
RtpOffset :	Offset Time (value = 6.0, unit = ms)

Represented Task:	SchedulableTask task3
RtpExecutionTime :	Execution Time (value = 4.0, unit = ms)
RtpDeadline :	Deadline (value = 5.0, unit = ms)
RtpOffset :	Offset Time (value = 11.0, unit = ms)

Represented Task:	SchedulableTask task1
RtpExecutionTime :	Execution Time (value = 3.0, unit = ms)
RtpDeadline :	Deadline (value = 4.0, unit = ms)
RtpOffset :	Offset Time (value = 1.0, unit = ms)

Name:	trigger
Scheduling Activities	TaskActivity taskActivity1 TaskActivity taskActivity2 TaskActivity taskActivity3
RtpReleaseTime :	Release Time (value = 0.0, unit = ms)
RtpPeriodicity :	Periodic (period = (value = 16.0, unit = ms))

Figure A.6: MoSaRT behavioral model of a real-time system with a transaction

Implementations

This appendix is devoted to the presentation of the technical part of the MoSaRT framework. Indeed, we introduce the techniques used in order to get the front-end and the back-end of the framework.

I. Implementation of the MoSaRT design language

I.1. Creation of the metamodel

We have used the Eclipse Modeling Framework (EMF)[1] to implement the metamodel. EMF is a modeling framework and code generation facility for building tools and applications based on a structured data model. EMF offers graphical and textual concrete syntaxes to build models.
For instance, Figure B.1 shows the `GolablSystem` concept (the root element of the MoSaRT language) expressed in three different ways, using two different graphical notations (Parts 1 and 2), and the textual notation (Part 3) which eases the injection of OCL constraints in the metamodel.

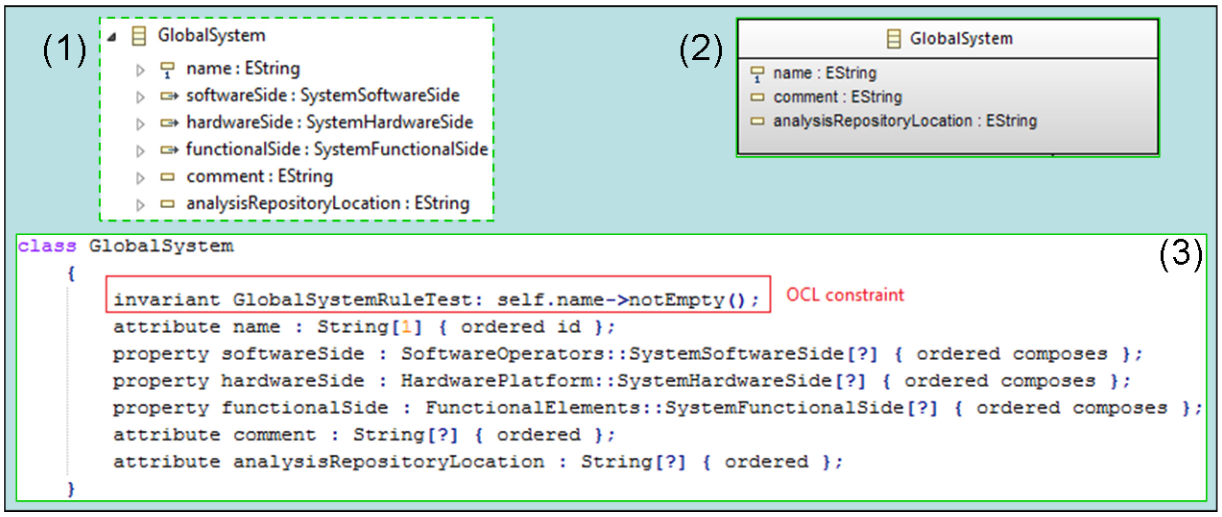
I.1.a. Integration of UML

Currently, in the MoSaRT design language we consider functional elements expressed in UML. This latter has been integrated through the `SystemFunctionalSide` element (see Figure B.2).

I.1.b. Structural rules

Structural rules have been created as OCL constraints enriching the metamodel. Part (3) of Figure B.1 shows one of different manners to add an OCL constraint to the metamodel. Every constraint is related to a context (i.e. Ecore class that have to be checked during its instantiation). The keyword *self* means the context of the constraint (like the keyword *this* in Java programming language).

[1]www.eclipse.org/modeling/emf/


(1)
GlobalSystem
name : EString
softwareSide : SystemSoftwareSide
hardwareSide : SystemHardwareSide
functionalSide : SystemFunctionalSide
comment : EString
analysisRepositoryLocation : EString
(2)
GlobalSystem
name : EString
comment : EString
analysisRepositoryLocation : EString
(3)
OCL constraint

```
class GlobalSystem
    {
        invariant GlobalSystemRuleTest: self.name->notEmpty();
        attribute name : String[1] { ordered id };
        property softwareSide : SoftwareOperators::SystemSoftwareSide[?] { ordered composes };
        property hardwareSide : HardwarePlatform::SystemHardwareSide[?] { ordered composes };
        property functionalSide : FunctionalElements::SystemFunctionalSide[?] { ordered composes };
        attribute comment : String[?] { ordered };
        attribute analysisRepositoryLocation : String[?] { ordered };
    }
```

Figure B.1: Different concrete syntaxes of Ecore

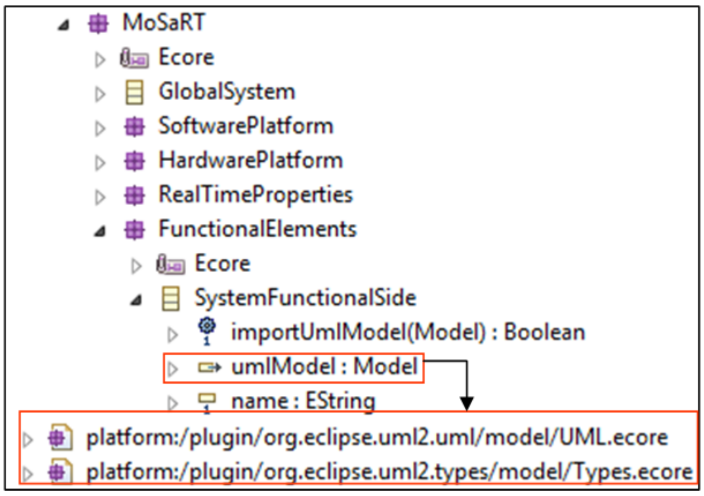


Figure B.2: Integration of UML into MoSaRT design language

I.1.c. Code generation

Once the metamodel is created and enriched by the OCL constraints, we generate the corresponding Java code. Thanks to the code generation we can handle instances of the MoSaRT design language. The code generation process is provided by EMF, but required a set of libraries called plug-ins. They enable the customization of the generated code. For instance, *org.eclipse.ocl.ecore* permits to transform the OCL constraints to adequate Java operations using OCL library for checking the model instances. Figure B.3 shows the plug-ins required for the code generation process.
EMF generates a set of Java files (e.g. *Interface*, *Implementation*, *Validator*, *Factory*, etc.) for each Ecore class of the meta-model. Those files contain the implementation of the properties and operations

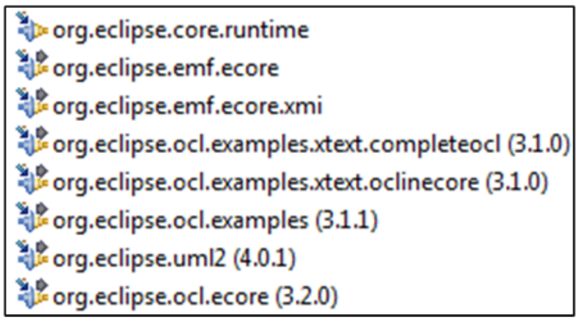
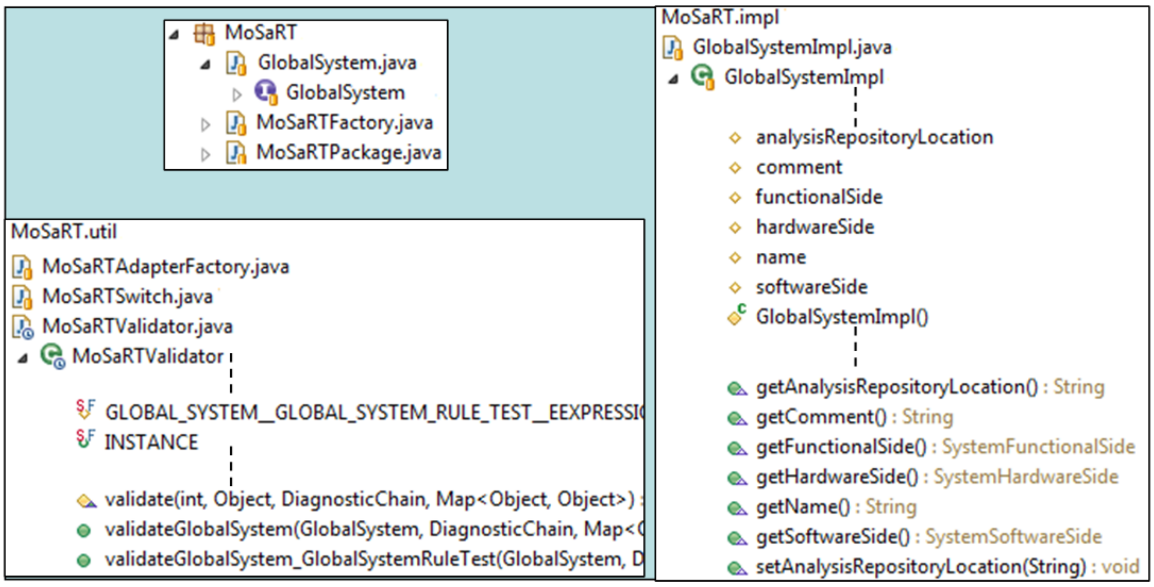

org.eclipse.core.runtime
org.eclipse.emf.ecore
org.eclipse.emf.ecore.xmi
org.eclipse.ocl.examples.xtext.completeocl (3.1.0)
org.eclipse.ocl.examples.xtext.oclinecore (3.1.0)
org.eclipse.ocl.examples (3.1.1)
org.eclipse.uml2 (4.0.1)
org.eclipse.ocl.ecore (3.2.0)

Figure B.3: Plug-ins required for the code generation

of each class. Moreover, in case where the class has some related OCL constraints, every constraint is interpreted as an operation in the *validator* class. Figure B.4 shows the generated classes corresponding to the `GolablSystem` class. At this step, every *implementation* class can be refined manually in order to ease its usage or to enrich its capabilities. For example, Figure B.5 shows the refinement of the operation `convertsTo` of the `RtpExecutionTimeType` class.


MoSaRT
GlobalSystem.java
GlobalSystem
MoSaRTFactory.java
MoSaRTPackage.java

MoSaRT.util
MoSaRTAdapterFactory.java
MoSaRTSwitch.java
MoSaRTValidator.java
MoSaRTValidator
GLOBAL_SYSTEM__GLOBAL_SYSTEM_RULE_TEST__EEXPRESSI
INSTANCE
validate(int, Object, DiagnosticChain, Map<Object, Object>)
validateGlobalSystem(GlobalSystem, DiagnosticChain, Map<
validateGlobalSystem_GlobalSystemRuleTest(GlobalSystem, D

MoSaRT.impl
GlobalSystemImpl.java
GlobalSystemImpl
analysisRepositoryLocation
comment
functionalSide
hardwareSide
name
softwareSide
GlobalSystemImpl()
getAnalysisRepositoryLocation() : String
getComment() : String
getFunctionalSide() : SystemFunctionalSide
getHardwareSide() : SystemHardwareSide
getName() : String
getSoftwareSide() : SystemSoftwareSide
setAnalysisRepositoryLocation(String) : void

Figure B.4: The generated Java files corresponding to `GolablSystem`

I.2. Implementation of the graphical syntax

Even if EMF offers the possibility to generate by default a "tree" editor to manipulate the instances of the metamodel. In order to ensure a good clarity and an utilization fluency during the instantiation of the MoSaRT design language, we have opted for an academic version of the *Obeo-Designer* tool[2]

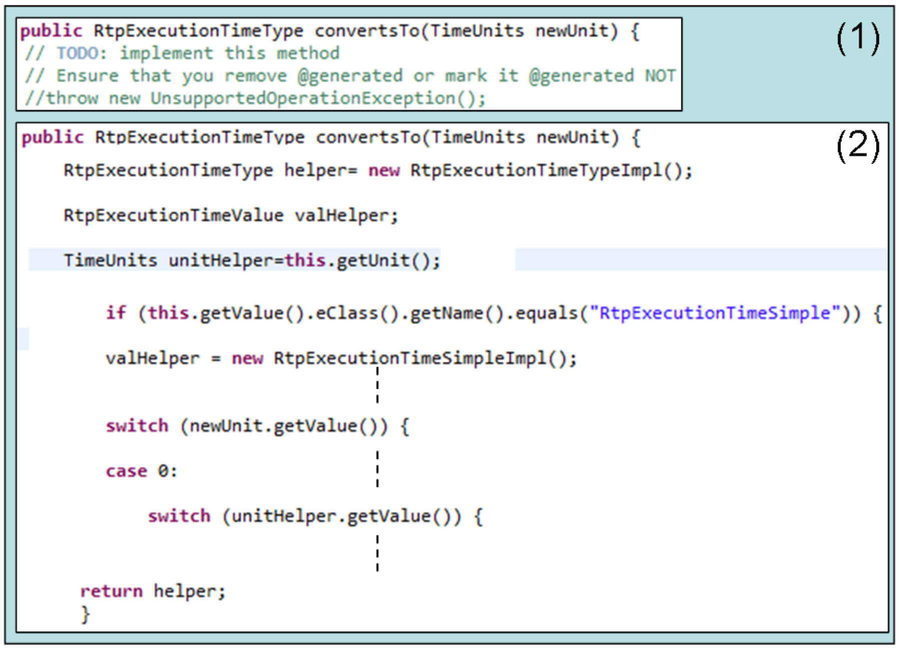
[2]www.obeodesigner.com

to develop the concrete syntax of the MoSaRT design language. By using this tool which is based on GMF (Graphical Modeling Framework), we have created the graphical concrete syntax of each element. Moreover, we have also set up diagrams allowing users of our design language to have several views of the system under conception. Part (1) of Figure B.6 shows the model created to manipulate the diagrams. It corresponds to an abstraction of the GMF layer managed by the *Obeo-Designer* tool.
The MoSaRT design language also provides a powerful graphical syntax for an easy use of the *Real-Time Properties*. This graphical syntax is due to usage of the Extended Editing Framework (EEF)[3] that provides advanced editing components for the properties of EMF elements. Thus, by instantiating metamodels provided by EEF (see Part (2) of Figure B.6), the properties related to every element of the MoSaRT metamodel are implemented as graphical boxes. Since the properties implementation is driven by EEF capabilities, the extension can be done easily, and the modification of a property implies the modification of EEF models.

I.3. Environment of the MoSaRT design language

Figure B.7 shows some capabilities provided by the front-end of the MoSaRT framework like the diagrams, a specific tab properties, palettes of every diagram, etc.
Figure B.8 illustrates the steps followed to specify that a trigger is periodic and the period is equal to 15 *us*. The transition from a dialog box to another one leads to get understandable information (e.g. "Periodic (period = (value 15.0)unit = us))").

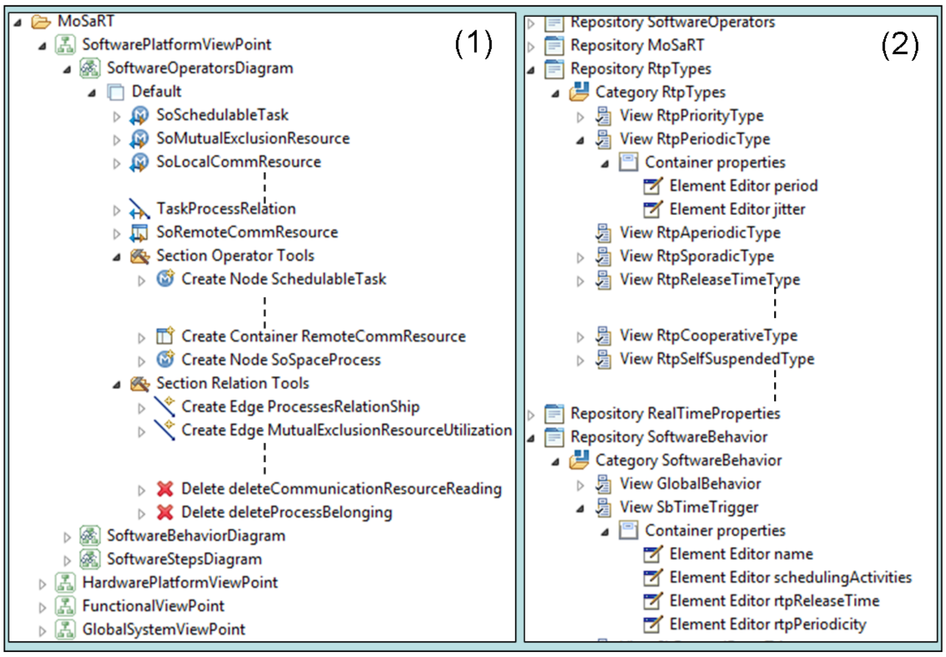
[3]www.eclipse.org/proposals/eef/



```
public RtpExecutionTimeType convertsTo(TimeUnits newUnit) {
 // TODO: implement this method
 // Ensure that you remove @generated or mark it @generated NOT
 //throw new UnsupportedOperationException();
```

```
public RtpExecutionTimeType convertsTo(TimeUnits newUnit) {
    RtpExecutionTimeType helper= new RtpExecutionTimeTypeImpl();

    RtpExecutionTimeValue valHelper;

    TimeUnits unitHelper=this.getUnit();

        if (this.getValue().eClass().getName().equals("RtpExecutionTimeSimple")) {

        valHelper = new RtpExecutionTimeSimpleImpl();

        switch (newUnit.getValue()) {

        case 0:

            switch (unitHelper.getValue()) {

      return helper;
      }
```

Figure B.5: The operation `convertsTo` of the `RtpExecutionTimeType` class. (1): before refinement, (2): after refinement



```
(1)
MoSaRT
  SoftwarePlatformViewPoint
    SoftwareOperatorsDiagram
      Default
        SoSchedulableTask
        SoMutualExclusionResource
        SoLocalCommResource
        TaskProcessRelation
        SoRemoteCommResource
        Section Operator Tools
          Create Node SchedulableTask
          Create Container RemoteCommResource
          Create Node SoSpaceProcess
        Section Relation Tools
          Create Edge ProcessesRelationShip
          Create Edge MutualExclusionResourceUtilization
          Delete deleteCommunicationResourceReading
          Delete deleteProcessBelonging
    SoftwareBehaviorDiagram
    SoftwareStepsDiagram
  HardwarePlatformViewPoint
  FunctionalViewPoint
  GlobalSystemViewPoint
```

```
(2)
Repository SoftwareOperators
Repository MoSaRT
Repository RtpTypes
  Category RtpTypes
    View RtpPriorityType
    View RtpPeriodicType
      Container properties
        Element Editor period
        Element Editor jitter
    View RtpAperiodicType
    View RtpSporadicType
    View RtpReleaseTimeType
    View RtpCooperativeType
    View RtpSelfSuspendedType
Repository RealTimeProperties
Repository SoftwareBehavior
  Category SoftwareBehavior
    View GlobalBehavior
    View SbTimeTrigger
      Container properties
        Element Editor name
        Element Editor schedulingActivities
        Element Editor rtpReleaseTime
        Element Editor rtpPeriodicity
```

Figure B.6: Models for the graphical concrete syntax

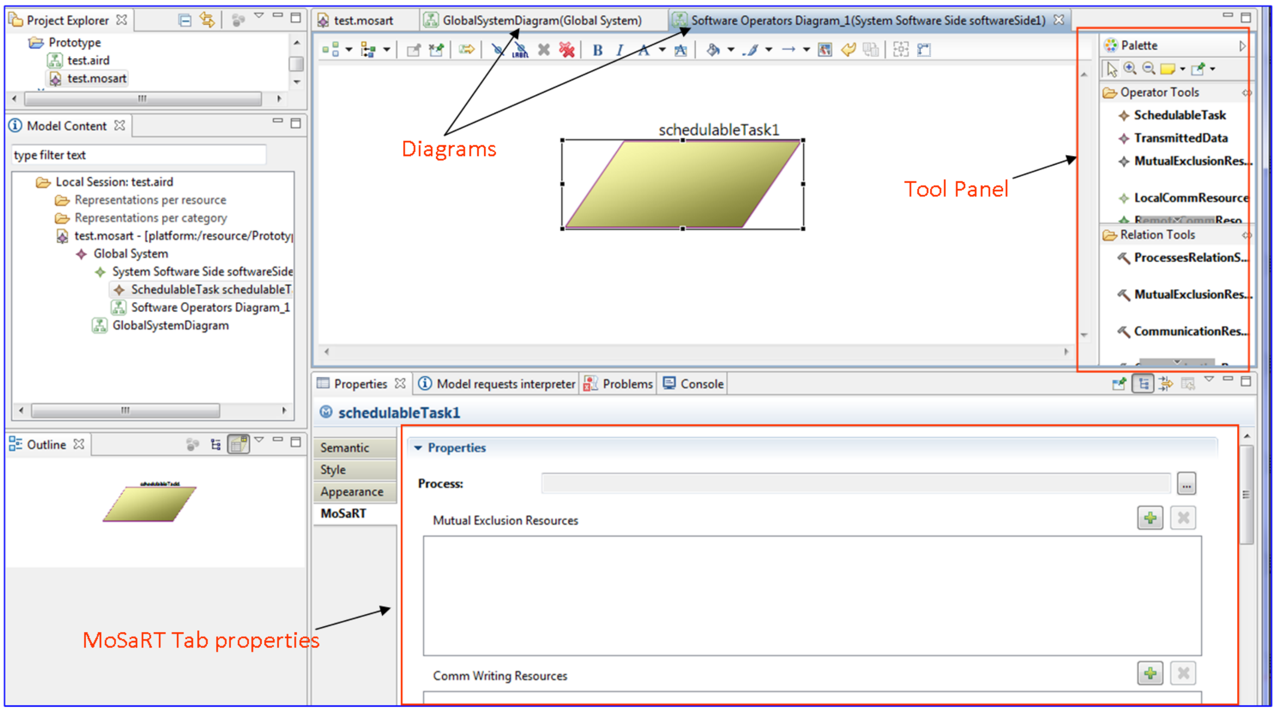
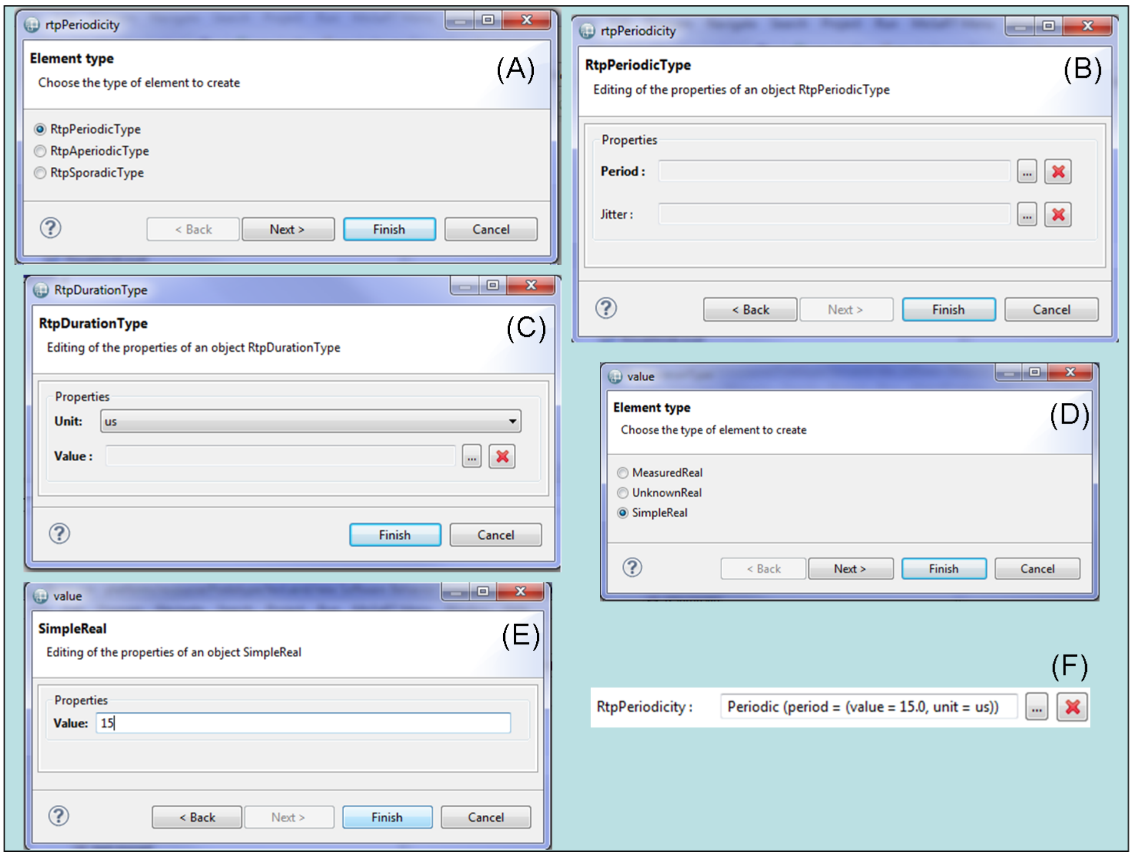


Figure B.7: Front-end of the MoSaRT framework



(A) [✓ RtpPeriodicType] RtpAperiodicType RtpSporadicType

(C) Unit: us

(D) MeasuredReal UnknownReal [✓ SimpleReal]

(E) Value: 15

(F) RtpPeriodicity : Periodic (period = (value = 15.0, unit = us))

Figure B.8: Example showing step by step how to specify the activation pattern of a trigger

II. Implementation of the MoSaRT analysis repository

During the implementation of the MoSaRT analysis repository, we have followed the same steps for the implementation of the MoSaRT design language as explained in the previous section, but with some modifications.

- Since we have opted to use OCL to express the formal content of every identification rule, the metamodel of the MoSaRT analysis repository integrates `EStringToStringMapEntry` and `Eclass`, which are two concepts of Ecore. During instantiation of the MoSaRT analysis repository, those elements link identification rules to OCL rules related to a design language.
- The metamodel of the MoSaRT analysis repository does not contain structural rules.
- We do not need diagrams to instantiate The MoSaRT analysis repository. We use the graphical syntax editor generated by default by EMF (i.e. *tree* editor). Moreover, we use EEF to get adequate forms.

II.1. Implementation of the identification process

Operations of the `AnalysisRepository` class have been refined in order to ensure the identification process. In particular, `identificationRuleChecker` and `appropriateModelsFinder` operations:

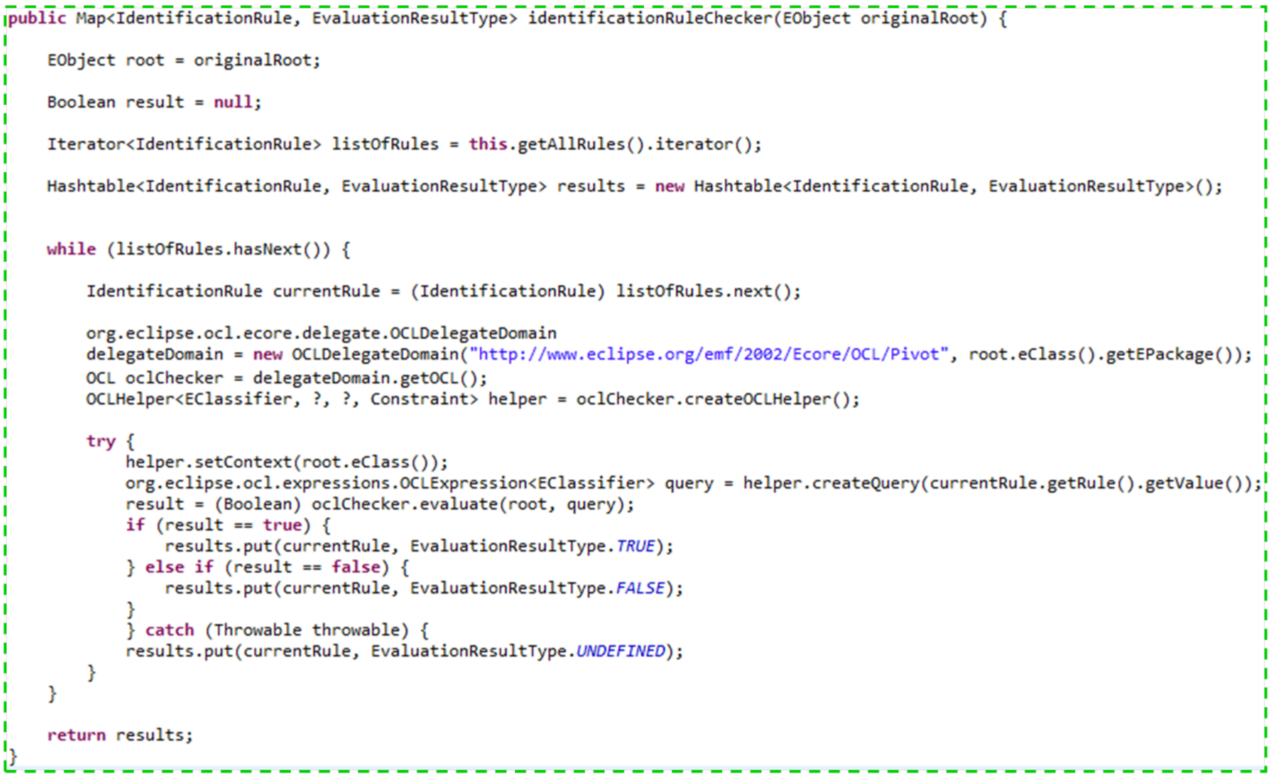
- `identificationRuleChecker` loads the design model and checks the identification rules by using the OCL API (Application Programming Interface) as shown in Figure B.9.
- `appropriateModelsFinder` is based on the result provided by `identificationRuleChecker`. It compares the result with the content of the analysis repository in roder to find the real-time context which corresponding to the design model.

II.2. Environment of the MoSaRT analysis repository

A version of the design language metamodel containing OCL analysis rules is provided in order to facilitate the use of the MoSaRT analysis repository with the MoSaRT design language. This version is called *MoSaRT4Identification.ecore*. Every analysis rule can be linked to an identification rule as shows in Figure B.10.

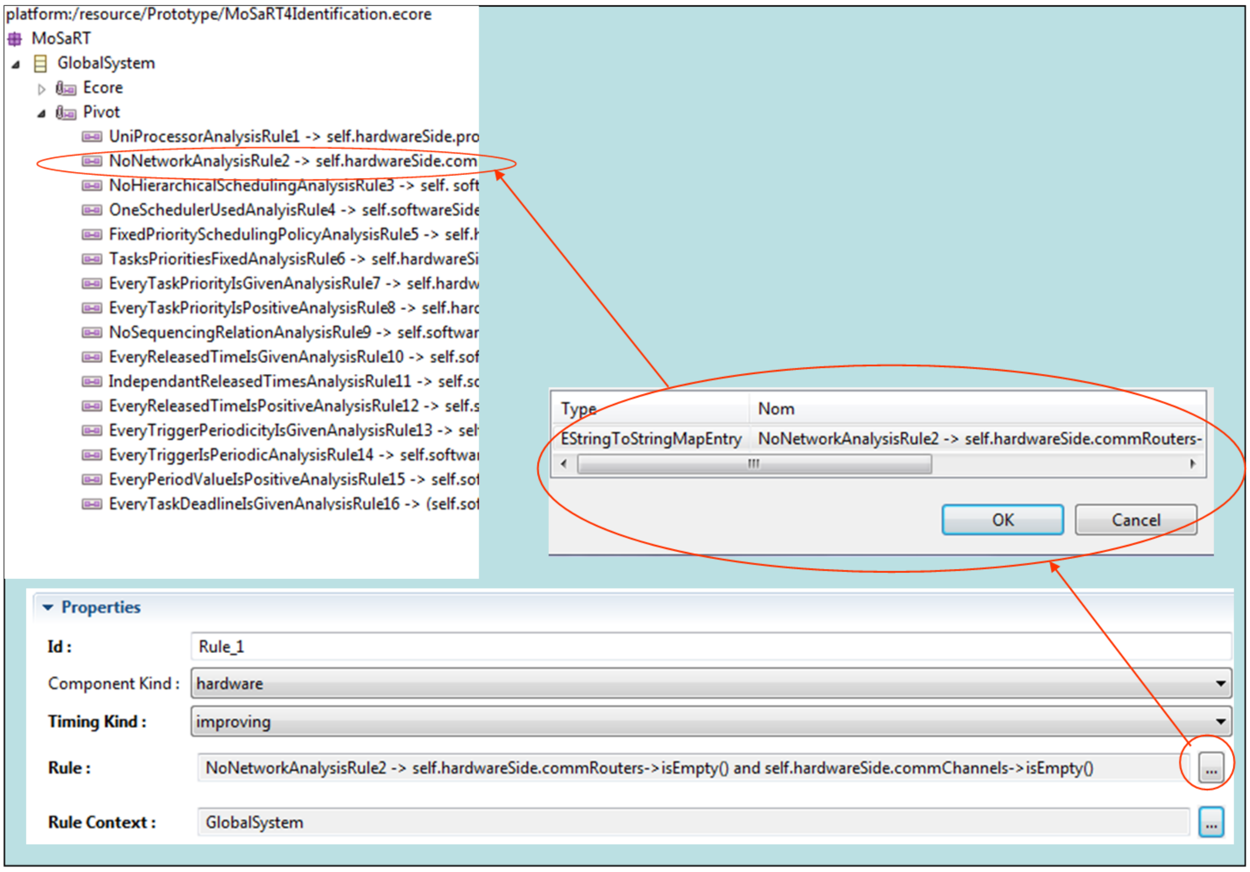
III. Implementation of identification process command

The identification process is implemented in Java. The command that launches the identification process has been integrated in MoSaRT as a handler (see Figure B.11). It allows to load the repository chosen for analysis, extracts the root element (i.e. instance of the `AnalysisRepository` class), and calls the identification process service. Once the results are ready, the command presents them in a convenient way. In cases where the process falls, the command shows error messages.



```java
public Map<IdentificationRule, EvaluationResultType> identificationRuleChecker(EObject originalRoot) {

    EObject root = originalRoot;

    Boolean result = null;

    Iterator<IdentificationRule> listOfRules = this.getAllRules().iterator();

    Hashtable<IdentificationRule, EvaluationResultType> results = new Hashtable<IdentificationRule, EvaluationResultType>();


    while (listOfRules.hasNext()) {

        IdentificationRule currentRule = (IdentificationRule) listOfRules.next();

        org.eclipse.ocl.ecore.delegate.OCLDelegateDomain
        delegateDomain = new OCLDelegateDomain("http://www.eclipse.org/emf/2002/Ecore/OCL/Pivot", root.eClass().getEPackage());
        OCL oclChecker = delegateDomain.getOCL();
        OCLHelper<EClassifier, ?, ?, Constraint> helper = oclChecker.createOCLHelper();

        try {
            helper.setContext(root.eClass());
            org.eclipse.ocl.expressions.OCLExpression<EClassifier> query = helper.createQuery(currentRule.getRule().getValue());
            result = (Boolean) oclChecker.evaluate(root, query);
            if (result == true) {
                results.put(currentRule, EvaluationResultType.TRUE);
            } else if (result == false) {
                results.put(currentRule, EvaluationResultType.FALSE);
            }
            } catch (Throwable throwable) {
            results.put(currentRule, EvaluationResultType.UNDEFINED);
        }
    }

    return results;
}
```

Figure B.9: refinement of the `identificationRuleChecker` operation



platform:/resource/Prototype/MoSaRT4Identification.ecore

MoSaRT

GlobalSystem

Ecore

Pivot

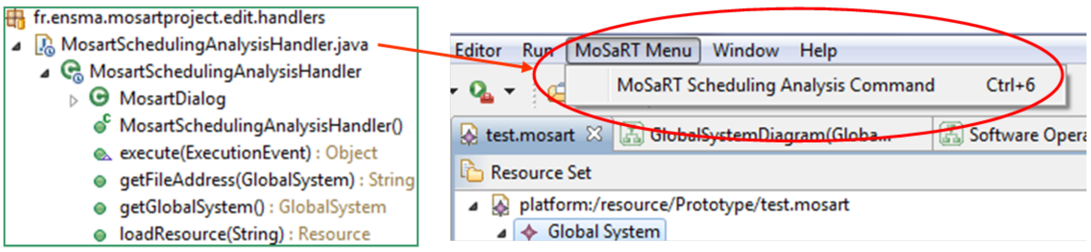
UniProcessorAnalysisRule1 -> self.hardwareSide.pro
NoNetworkAnalysisRule2 -> self.hardwareSide.com
NoHierarchicalSchedulingAnalysisRule3 -> self. soft
OneSchedulerUsedAnalyisRule4 -> self.softwareSide
FixedPrioritySchedulingPolicyAnalysisRule5 -> self.h
TasksPrioritiesFixedAnalysisRule6 -> self.hardwareSi
EveryTaskPriorityIsGivenAnalysisRule7 -> self.hardw
EveryTaskPriorityIsPositiveAnalysisRule8 -> self.harc
NoSequencingRelationAnalysisRule9 -> self.softwar
EveryReleasedTimeIsGivenAnalysisRule10 -> self.sof
IndependantReleasedTimesAnalysisRule11 -> self.sc
EveryReleasedTimeIsPositiveAnalysisRule12 -> self.s
EveryTriggerPeriodicityIsGivenAnalysisRule13 -> sel
EveryTriggerIsPeriodicAnalysisRule14 -> self.softwa
EveryPeriodValueIsPositiveAnalysisRule15 -> self.sof
EveryTaskDeadlineIsGivenAnalysisRule16 -> (self.sof

Type	Nom
EStringToStringMapEntry	NoNetworkAnalysisRule2 -> self.hardwareSide.commRouters-

Properties

Id :	Rule_1
Component Kind :	hardware
Timing Kind :	improving
Rule :	NoNetworkAnalysisRule2 -> self.hardwareSide.commRouters->isEmpty() and self.hardwareSide.commChannels->isEmpty()
Rule Context :	GlobalSystem

Figure B.10: Linking identification rules to OCL formal expressions



fr.ensma.mosartproject.edit.handlers
MosartSchedulingAnalysisHandler.java
MosartSchedulingAnalysisHandler
MosartDialog
MosartSchedulingAnalysisHandler()
execute(ExecutionEvent) : Object
getFileAddress(GlobalSystem) : String
getGlobalSystem() : GlobalSystem
loadResource(String) : Resource

Editor
Run
MoSaRT Menu
Window
Help
MoSaRT Scheduling Analysis Command
Ctrl+6
test.mosart
GlobalSystemDiagram(Globa...
Software Opera

Resource Set
platform:/resource/Prototype/test.mosart
Global System

Figure B.11: Command launching the identification process

Résumé

Contexte

L'utilisation des systèmes embarqués temps réel a connu une très grande croissance notamment dans les secteurs critiques tels que l'automobile ou l'avionique. Un système temps réel opère dans un environnement dynamique et doit constamment être adapté aux changements de cet environnement. Le bon fonctionnement d'un système temps réel dépend non seulement de la correction des résultats mais aussi des instants de production de ces résultats. Dans le cas des systèmes temps réel critiques, les anomalies temporelles peuvent entrainer des défaillances catastrophiques.
La forte utilisation des systèmes temps réel dans des secteurs divers a un grand impact sur l'évolution de la plate-forme matérielle et logicielle. La phase de conception d'un tel système peut prendre plusieurs mois voire plusieurs années, où le concepteur doit se montrer réflexif vis à vis l'évolution du matériel/logiciel et/ou des normes du domaine. Les mauvais choix effectués pendant la phase de conception peuvent avoir un grand impact sur le coût et la durée de développement (*Time-To-Market*), car souvent ils ne seront détectés qu'à la fin du cycle de vie logicielle. Dans cette thèse, nous nous intéressons aux choix de conception liés au comportement temporel des systèmes temps réel critiques, d'où la nécessité d'une phase d'analyse au moment de la conception pour vérifier l'ordonnançabilité des systèmes temps réel en cours de conception.
L'analyse d'ordonnançabilité est un moyen de vérifier que le fonctionnement d'une application temps réel respectera ses contraintes temporelles. A cet égard, plusieurs travaux autour de la théorie de l'ordonnancement temps réel ont proposé de nombreux modèles à différents niveaux d'abstraction, ainsi que plusieurs tests d'analyse asscociés avec différents degrés de précision et de complexité.

Motivation

Pour assurer l'évolutivité et la réutilisabilité des systèmes temps réel, leur développement est touché par la popularité de l'ingénierie dirigée par les modèles. Récemment, des méthodologies de conception de systèmes temps réel permettent de prendre en compte conjointement ou indépendamment l'aspect

logiciel, exécutif et matériel. Ces méthodologies sont basées sur l'ingénierie dirigée par les modèles facilitant l'utilisation de la théorie de l'ordonnancement pendant l'étape de conception mais au coût d'une complexité importante de prise en main par un concepteur.
En prenant en considération la criticité des systèmes temps réel, le choix du test de validation et/ou de dimensionnement le plus adapté est d'une grande importance. La théorie de l'ordonnancement donne lieu à de très nombreuses méthodes analytiques qui peuvent être d'une grande utilité dans le monde industriel, mais qui ne sont pas assez appliquées dans l'industrie. Actuellement, l'utilisation des méthodes analytiques est dirigée par l'expérience des concepteurs, alors que la théorie de l'ordonnancement temps réel est developpée continuellement depuis plus de quarante ans. De plus, chaque méthode d'analyse a un niveau d'exactitude qui est souvent lié à la situation du système qui nécessite l'analyse. Il devient alors nécessaire d'orienter le concepteur pour choisir les bons modèles d'analyses et les bons tests non seulement pour éviter les anomalies dues à la conception mais aussi pour réduire le pessimisme et le surdimensionnement qui peuvent être un résultat d'un mauvais choix d'analyse.

Objectifs de la thèse

Les contributions présentées dans cette thèse ont deux principaux objectifs.
1) Le premier objectif c'est qu'à travers les contributions on vise à aider le concepteur pour lui faciliter la tâche de l'analyse d'ordonnançabilité en le guidant vers le bon choix de tests d'analyse et en l'aidant à dimensionner le systèmes en cours de conception à travers des raffinement et des rectifications de modèles.
2) Le deuxième objectif est l'augmentation de l'applicabilité de la théorie de l'ordonnancement. En effet, actuellement le nombre des travaux de recherches qui sont adoptés et utilisés dans l'industrie est réduit comparativement à l'offre de la recherche. Par conséquent, il serait intéressant de fournir un moyen technologique flexible pour transférer les résultats théoriques du domaine académique vers le domaine industriel.

Solutions proposées

Pour atteindre nos objectifs, nous profitons des avantages de l'ingénierie dirigée par les modèles pour proposer un Framework intermédiaire entre les différent langages de modélisation des systèmes temps-réels et les outils d'analyses. Ce Framework est nommé MoSaRT (*Modeling-oriented Scheduling analysis of Real-Time systems*). MoSaRT est un Framework intermédiaire qui joue le rôle d'un pont entre les langages de modélisation et les différents outils et prototypes d'analyse, il contient deux types de services.
MoSaRT propose un langage de conception extensible et utilisable comme un langage pivot dédié aux concepteurs sans que ces derniers aient une grande connaissance dans la théorie de l'ordonnancement temps réel.
Le deuxième service offert par MoSaRT est un méta-modèle pour concevoir les référentiels d'analyse. Chaque référentiel d'analyse joue le rôle d'un conseiller d'analyse. Ce méta-modèle est dédié aux analystes/chercheurs afin d'introduire les tests d'analyse, leurs contextes et leurs caractéristiques selon chaque contexte. Le but est que chaque équipe de recherche puisse introduire le résultat de ses travaux même s'ils ne sont pas encore implémentés dans des outils.

Chaque modèle conçu selon le langage de conception offert par MoSaRT peut être contrôlé par un référentiel d'analyse. En conséquence, l'orientation du concepteur vers le test qui correspond au modèle est fortement liée à l'exactitude et la richesse du contenu du référentiel choisi.

Plan du Mémoire

La première partie du manuscrit contient trois chapitres. Le chapitre 2 présente la structure générale des systèmes temps réel, il donne également un aperçu sur la phase de conception et de son importance tout au long du processus de développement.
Le chapitre 3 présente brièvement les aspects liés à la théorie de l'ordonnancement temps réel. Il présente certaines propriétés temporelles qui permettent d'identifier la nature du système temps réel.
Le chapitre 4 est consacré à la présentation des concepts liés à l'ingénierie dirigée par les modèles. Il présente l'utilisation de l'ingénierie dirigée par les modèles dans la conception et l'analyse des systèmes temps réel. Des langages de modélisation standards comme *AADL* et *UML-MARTE* ainsi qu'un ensemble d'outils d'analyses y sont présentés.
La deuxième partie du manuscrit est consacrée à la contribution. Elle commence par le chapitre 5, qui décrit le positionnement par rapport aux problèmes énoncés. Le chapitre 5 décrit les objectifs que nous visons à atteindre dans cette thèse.
Le chapitre 6 présente le langage de conception MoSaRT design language en se référant à la théorie de l'ordonnancement temps réel. Il décrit les éléments fonctionnels et opérationnels, ainsi que leur sémantique. Des règles structurelles permettant une bonne utilisation du langage de conception MoSaRT sont aussi présentées.
Le chapitre 7 met en évidence les différentes caractéristiques qui aident les concepteurs à identifier le contexte du système à analyser et leur facilitent le choix de tests d'analyse. Il présente une approche basée sur le référentiel d'analyse (*analysis repository* en anglais) afin d'améliorer la façon dont les concepteurs vérifient leurs modèles. Les référentiels d'analyses sont des instances (du méta-modèle de *MoSaRT analysis repository*) qui permettront d'accroître l'applicabilité de la théorie de l'ordonnancement temps réel en lien avec les ateliers de conception logicielle basés modèles.
Le chapitre 8 présente les objectifs du Framework MoSaRT, décrit la structure du Framework et ses différentes fonctionnalités. Des études de cas qui utilisent le Framework y sont présentées.
Le chapitre 9 résume et conclut la thèse en discutant les résultats obtenus et les perspectives.

Related Publications

[OGH13]: Yassine Ouhammou, Emmanuel Grolleau, and Jérôme Hugues. Mapping AADL models to a repository of multiple schedulability analysis techniques. In IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), pages 8, 2013.

[OGRR12b]: Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Model Driven Timing Analysis for Real-Time Systems. In IEEE International Conference on Embedded Software and Systems (ICESS), pages 1458−1465, 2012.

[OGRR12c]: Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Reducing the gap between Design and Scheduling. In Real-Time and Network Systems (RTNS), pages 21−30. ACM, 2012.

[OGRR12a]: Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. From Model-based design to Real-Time Analysis. In International Conference on Advances in System Testing and Validation Lifecycle (VALID), pages 45−50, 2012.

[OGRR12d]: Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Un méta-modele pour la conception et la validation des architectures embarquées temps-réel complexes. In Conférence francophone sur les Architectures Logicielles (CAL), pages 142−147, 2012.

[OGRR11]: Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Towards a Simple Meta-Model for Complex Real-Time and Embedded Systems. In International Conference on Model and Data Engineering (MEDI), pages 226−236. Springer LNCS, 2011.

Submitted: other conference/journal articles are under submission.

List of Figures

List of Tables

Bibliography

[AGAT10] Saoussen Anssi, Sébastien Gérard, Arnaud Albinet, and François Terrier. Requirements and Solutions for Timing Analysis of Automotive Systems. In *Workshop on System Analysis and Modelling (SAM)*, pages 209–220, 2010.

[AMK98] Elan Amir, Steven McCanne, and Randy Katz. An active service framework and its application to real-time multimedia transcoding. *ACM SIGCOMM Computer Communication Review*, 28(4):178–189, 1998.

[APK$^+$11] Saoussen Anssi, Sara Tucci Piergiovanni, Stefan Kuntz, Sébastien Gérard, and François Terrier. Enabling Scheduling Analysis for AUTOSAR Systems. In *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 152–159, 2011.

[Aud91] N.C. Audsley. Optimal Priority Assignment And Feasibility Of Static Priority Tasks With Arbitrary Start Times. Technical report, University of York, Department of Computer Science, 1991.

[AUT] AUTOSAR. Official website. `www.autosar.org`. Last access: 10/09/2013.

[AvdBEV12] Suzana Andova, Mark G. J. van den Brand, Luc J. P. Engelen, and Tom Verhoeff. MDE Basics with a DSL Focus. In *International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM): Formal Methods for Model-Driven Engineering*, pages 21–57. Springer, 2012.

[Bar98] Sanjoy K. Baruah. Feasibility analysis of recurring branching tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 138–145, 1998.

[Bar03] Sanjoy K. Baruah. Dynamic and Static priority Scheduling of Recurring Real-time Tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[Bar10] Sanjoy Baruah. The Non-cyclic Recurring Real-Time Task Model. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 173–182, 2010.

[BB91] Albert Benveniste and Gérard Berry. The synchronous approach to reactive and real-time systems. *Proceedings of the IEEE*, 79(9):1270–1282, 1991.

[BB02] Jean Bézivin and Xavier Blanc. MDA: Vers un important changement de paradigme en génie logiciel. *Développeur référence v2*, 16:15, 2002.

[BB06] Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006.

[BB08] Alan Burns and Sanjoy K. Baruah. Sustainability in Real-time Scheduling. *Journal of Computing Science and Engineering (JCSE)*, 2(1):74–97, 2008.

[BBAL06] Cesare Bartolini, Antonia Bertolino, Guglielmo De Angelis, and Giuseppe Lipari. A UML Profile and a Methodology for Real-Time Systems Design. In *EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA)*, pages 108–117, 2006.

[BBY13] Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. Limited Preemptive Scheduling for Real-Time Systems. A Survey. *IEEE Trans. Industrial Informatics*, 9(1):3–15, 2013.

[BCGM99] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized Multiframe Tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[BDNB08] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39:5–30, 2008.

[BLN05] Cesare Bartolini, Giuseppe Lipari, and Marco Di Natale. From Functional Blocks to the Synthesis of the Architectural Model in Embedded Real-time Applications. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 458–467, 2005.

[Bro87] Frederick P Brooks. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20(4):10–19, 1987.

[But11] Giorgio C. Buttazzo. *Hard-real time computing systems : predictable scheduling algorithms and applications*, volume 24. Springer, 2011.

[BW01] Alan Burns and Andrew J Wellings. *Real-Time Systems and Programing Lenguajes: Ada 95, Real Time Java and Real Time Posix.* Addison Wesley, 2001.

[BW09] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX.* Addison Wesley, 4 edition, 2009.

[C$^+$97] Airlines Electronic Engineering Committee et al. ARINC report 651-1: Design guidance for integrated modular avionics. *Aeronautical radio, Inc., Annapolis, Maryland*, 1997.

[CEP03] Luis Alejandro Cortés, Petru Eles, and Zebo Peng. Modeling and formal verification of embedded systems based on a Petri net representation. *Journal of Systems Architecture*, 49(12-15):571–598, 2003.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[CFH$^+$04] John Carpenter, Shelby Funk, Philip Holman, Anand Srinivasan, James Anderson, and Sanjoy Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In *HANDBOOK ON SCHEDULING ALGORITHMS, METHODS, AND MODELS*. Chapman Hall/CRC, Boca, 2004.

[CH06] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[CMTG07] Arnaud Cuccuru, Chokri Mraidha, François Terrier, and Sébastien Gérard. Templatable Metamodels for Semantic Variation Points. In *European Conference Model Driven Architecture - Foundations and Applications (ECMDA-FA)*, pages 68–82. Springer LNCS, 2007.

[Com] Obeo Company. Acceleo model to text transformation language. `www.acceleo.org`. Last access: 10/09/2013.

[Coo71] Stephen A Cook. The Complexity of Theorem-Proving Procedures. In *ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[CP00] Antoine Colin and Isabelle Puaut. Worst case execution time analysis for a processor with branch prediction. *Real-Time Systems*, 18(2-3):249–274, 2000.

[CRS11] Joao Craveiro, José Rufino, and Frank Singhoff. Architecture, mechanisms and scheduling analysis tool for multicore time-and space-partitioned systems. *ACM SIGBED Review*, 8(3):23–27, 2011.

[DBBL07] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network CAN schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[dC] Universidad de Cantabria. MAST: Modeling and Analysis Suite for Real-Time Applications. `mast.unican.es`. Last access: 10/09/2013.

[Dec03] David Decotigny. *Une infrastructure de simulation modulaire pour l'évaluation de performances de systèmes temps-réel.* PhD thesis, Université Rennes 1, 2003.

[Der74] Michael L. Dertouzos. Control Robotics: The Procedural Control of Physical Processes. In *IFIP Congress*, pages 807–813, 1974.

[DH92] Kevin Driscoll and Kenneth Hoyme. The airplane information management system: An integrated real-time flight-deck control system. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 267–270, 1992.

[DL78] S K Dhall and C L Liu. On a real-time scheduling problem. *Operations Research*, 26(1):127–140, 1978.

[DRGR08] François Dorin, Michaël Richard, Emmanuel Grolleau, and Pascal Richard. Minimizing the number of processors for real-time distributed systems. In *Real-Time and Network Systems (RTNS)*, 2008.

[dS13] AFIS : Association Française d'Ingénierie Systéme. Norme ISO 15288/Systems Engineering, System Life-Cycle Processes. `www.afis.fr/nm-is/Pages/Normes%20IS/Norme%20ISO%2015288.aspx`, June 2013. Last access: 10/09/2013.

[DSLT05] Vincent Debruyne, Françoise Simonot-Lion, and Yvon Trinquet. EAST-ADL: An Architecture Description Language. In *Architecture Description Languages*, chapter 12, pages 181–195. Springer, 2005.

[Ecl] Eclipse. Eclipse Modeling Project. `www.eclipse.org/modeling/`. Last access: 10/09/2013.

[EP93] Cem Ersoy and Shivendra S. Panwar. Topological design of interconnected LAN/MAN networks. *IEEE Journal on Selected Areas in Communications*, 11(8):1172–1182, 1993.

[Esp07] Huascar Espinoza. *An Integrated Model-Driven Framework for Specifying and Analyzing Non-Functional Properties of Real-Time Systems.* PhD thesis, Evry University - CEA LIST, 2007.

[FFVM04] Lidia Fuentes-Fernández and Antonio Vallecillo-Moreno. An introduction to UML profiles. *European Journal for the Informatics Professional*, 2, 2004.

[Fly72] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 100(9):948–960, 1972.

[G^+] OSEK Group et al. OSEK/VDX Operating System Specification. `www.osek-vdx.org`. Last access: 10/09/2013.

[GETS07] Sébastien Gérard, Huáscar Espinoza, François Terrier, and Bran Selic. Modeling Languages for Real-Time and Embedded Systems - Requirements and Standards-Based Solutions. In *Model-Based Engineering of Embedded Real-Time Systems*, pages 129–154. Springer LNCS, 2007.

[GGKH03] Tracy Gardner, Catherine Griffin, Jana Koehler, and Rainer Hauser. A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard. In *MetaModelling for MDA Workshop*, pages 178–197, 2003.

[GNS+07] Paolo Gai, Marco Di Natale, Nicola Serreli, Luigi Palopoli, and Alberto Ferrari. Adding Timing Analysis to Functional Design to Predict Implementation Errors. SAE Technical Paper 2007-01-1272, 2007.

[Gom84] Hassan Gomaa. A Software Design Method for Real-Time Systems. *Communications of the ACM*, 27(9):938–949, 1984.

[Gom08] Hassan Gomaa. Advances in Software Design Methods for Concurrent, Real-Time and Distributed Applications. In *International Conference on Software Engineering Advances (ICSEA)*, pages 451–456. IEEE Computer Society, 2008.

[Gor95] Walter J Goralski. *Introduction to ATM networking.* McGraw-Hill New York, 1995.

[Hal92] Nicolas Halbwachs. *Synchronous programming of reactive systems*, volume 215. Springer, 1992.

[HG10] Matthias Hagner and Ursula Goltz. Integration of Scheduling Analysis into UML Based Development Processes Through Model Transformation. In *International Multiconference on Computer Science and Information Technology (IMCSIT)*, pages 797–804, 2010.

[HNN05] Hans Hansson, Mikael Nolin, and Thomas Nolte. Real-Time Systems. In Richard Zurawski, editor, *The Industrial Information Technology Handbook*, chapter Real-Time Embedded Systems, pages 1–28. CRC Press, Taylor & Francis Group, 2005.

[Hoa69] C. A. R. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580, 1969.

[HS09] Jérôme Hugues and Frank Singhoff. Développement de systémes á l'aide d'AADL-Ocarina/Cheddar. Tutorial presented at the summer school (ETR), 2009.

[JK05] Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In *MoDELS Satellite Events*, pages 128–138, 2005.

[JP86] Mathai Joseph and Paritosh K. Pandya. Finding Response Times in a Real-Time System. *Computer Journal*, 29(5):390–395, 1986.

[JS93] Kevin Jeffay and D Stone. Accounting for interrupt handling costs in dynamic priority task systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 212–221, 1993.

[Kla07] Benjamin Klatt. Xpand: A Closer Look at the Model2Text Transformation Language. *Chair of Software Design and Quality (SDQ)*, 10(16), 2007.

[Kop91] Hermann Kopetz. Event-triggered versus time-triggered real-time systems. In *Operating Systems of the 90s and Beyond*, pages 86–101. Springer, 1991.

[KWB03] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Practice and Promise of The Model Driven Architecture.* Addison Wesley Professional, 2003.

[Leh90] John P. Lehoczky. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 201–209, 1990.

[LHG98] Juan C López, Román Hermida, and Walter Geisselhardt. *Advanced techniques for embedded systems design and test.* Springer, 1998.

[LIS] CEA LIST. Papyrus: open source tool for graphical UML2 modelling. `www.papyrusuml.org`. Last access: 10/09/2013.

[LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, 1973.

[LM80] Joseph Y-T Leung and ML Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.

[LSD89] John Lehoczky, Lui Sha, and Ye Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 166–171, 1989.

[LW82] Joseph Y-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.

[MC96] Aloysius K. Mok and Deji Chen. A Multiframe Model for Real-Time Tasks. *IEEE Transactions on Software Engineering*, 23:635–645, 1996.

[MC11] Julio L. Medina and Álvaro García Cuesta. From composable design models to schedulability analysis with UML and the UML profile for MARTE. *SIGBED Rev.*, 8:64–68, 2011.

[MJL06] Jonathan Musset, Etienne Juliot, and Stéphane Lacrampe. Acceleo référence. Technical report, Obeo - Acceleo, 2006.

[MKH03] Jamison Masse, Saehwa Kim, and Seongsoo Hong. Tool Set Implementation for Scenario-based Multithreading of UML-RT Models and Experimental Validation. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2003.

[Mok83] A. K. Mok. *FUNDAMENTAL DESIGN PROBLEMS OF DISTRIBUTED SYSTEMS FOR THE HARD-REAL-TIME ENVIRONMENT.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[MPHD01] Julio L. Medina, Julio L. Medina Pasaje, Michael Gonzalez Harbour, and Jose M. Drake. MAST Real-Time View: A Graphic UML Tool for Modeling Object-Oriented Real-Time Systems. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 245–256, 2001.

[MTN04] Jukka Mäki-Turja and Mikael Nolin. Efficient Response-Time Analysis for Tasks with Offsets. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 462–471, 2004.

[MTPG11] Chokri Mraidha, Sara Tucci-Piergiovanni, and Sebastien Gerard. Optimum: a MARTE-based methodology for schedulability analysis at early design stages. *ACM SIGSOFT Software Engineering Notes*, 36:1–8, 2011.

[NIS02] NIST. The Economic Impacts of Inadequate Infrastructure for Software Testing. Planning Report 02-3 at `www.nist.gov/director/planning/upload/report02-3.pdf`, May 2002. National Institute of Standards and Testing.

[NPSB09] Min-Young Nam, Rodolfo Pellizzoni, Lui Sha, and Richard M. Bradford. ASIIST: Application Specific I/O Integration Support Tool for Real-Time Bus Architecture Designs. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 11–22. IEEE Computer Society, 2009.

[NSST98] Kunihiko Nabeshima, Tomoaki Suzudo, Katsuo Suzuki, and Erdinç TÜRKCAN. Real-time nuclear power plant monitoring with neural network. *Journal of Nuclear Science and Technology*, 35(2):93–100, 1998.

[oAES] Society of Automotive Engineers (SAE). The SAE Architecture Analysis & Design Language Standard. `www.aadl.info`. Last access: 10/09/2013.

[OGH13] Yassine Ouhammou, Emmanuel Grolleau, and Jerome Hugues. Mapping AADL models to a repository of multiple schedulability analysis techniques. In *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, page 8, 2013.

[OGRR11] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Towards a Simple Meta-Model for Complex Real-Time and Embedded Systems. In *International Conference on Model and Data Engineering (MEDI)*, pages 226–236. Springer LNCS, 2011.

[OGRR12a] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. From Model-based design to Real-Time Analysis. In *International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, pages 45–50, 2012.

[OGRR12b] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Model Driven Timing Analysis for Real-Time Systems. In *IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1458–1465, 2012.

[OGRR12c] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Reducing the gap between Design and Scheduling. In *Real-Time and Network Systems (RTNS)*, pages 21–30. ACM, 2012.

[OGRR12d] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Un méta-modele pour la conception et la validation des architectures embarquées temps-réel complexes. In *Conférence francophone sur les Architectures Logicielles (CAL)*, pages 142–147, 2012.

[OMG04] OMG. *Unified Modeling Language Version 2.0.* OMG Inc., 2004.

[OMG06] OMG. Object Constraint Language, OMG Available Specification, Version 2.0. `www.omg.org/spec/OCL/2.0/`, 2006.

[OMG07] OMG. OMG System Modeling Language (SysML) Version 1.0. `www.omgsysml.org`, 2007.

[OMG09] OMG. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems. `www.omgmarte.org`, 2009.

[OMG11a] OMG. OMG Meta Object Facility (MOF) Core Specification, Version 2.4.1. `www.omg.org/spec/MOF/2.4.1/`, 2011.

[OMG11b] OMG. OMG MOF 2 XMI Mapping Specification, Version 2.4.1. `www.omg.org/spec/XMI/2.4.1/`, 2011.

[oY94] The University of York. Offset analysis tool. `www.cs.york.ac.uk/ftpdir/pub/realtime/programs/src/offsets/`, 1994. Last access: 10/09/2013.

[PBKS07] Alexander Pretschner, Manfred Broy, Ingolf H Kruger, and Thomas Stauner. Software engineering for automotive systems: A Roadmap. In *Workshop on the Future of Software Engineering (FOSE)*, pages 55–71, 2007.

[PGH98] J. C. Palencia and M. González Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 26–37, 1998.

[QFGM13] Manar Qamhieh, Frederic Fauberteau, Laurent George, and Serge Midonnet. Global EDF Scheduling of Directed Acyclic Graphs on Multiprocessor Systems. In *Real-Time and Network Systems (RTNS)*, pages 289–298. ACM, 2013.

[REP12] Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Model Transformations. In *International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM): Formal Methods for Model-Driven Engineering*, pages 91–136, 2012.

[rg13] ATLAS INRIA & LINA research group. Atlas Transformation Language. `www.eclipse.org/atl/`, 2013.

[RGRR12] Ahmed Rahni, Emmanuel Grolleau, Michaël Richard, and Pascal Richard. Feasibility analysis of real-time transactions. *Real-Time Systems*, 48(3):320–358, 2012.

[RRC03] Michaël Richard, Pascal Richard, and Francis Cottet. Allocating and Scheduling Tasks in Multiple Fieldbus Real-Time Systems. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 137–144, 2003.

[RS04] Christine Rochange and Pascal Sainrat. Vers une prédictibilité temporelle des processeurs haute-performance. *RTS Embedded Systems*, 2004.

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008.

[Sch07] Detlev Schaadt. AFDX/ARINC 664 Concept, Design, Implementation and Beyond. *SYSGO AG White Paper*, 2007.

[SEGY11] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The Digraph Real-Time Task Model. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 71–80, 2011.

[Sel07] Bran Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC)*, pages 2–9, 2007.

[SLNM04] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda*, pages 1–8, 2004.

[SRL90] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. Comput.*, 39:1175–1185, 1990.

[Sta88] John A. Stankovic. Misconceptions About Real-Time Computing. *IEEE Computer*, 21(10):10–19, 1988.

[SVCBS04] Alberto L. Sangiovanni-Vincentelli, Luca P. Carloni, Fernando De Bernardinis, and Marco Sgroi. Benefits and challenges for platform-based design. In *Design Automation Conference (DAC)*, pages 409–414. ACM, 2004.

[SVN07] Alberto L. Sangiovanni-Vincentelli and Marco Di Natale. Embedded System Design for Automotive Applications. *IEEE Computer*, 40(10):42–51, 2007.

[SZP+03] John A. Stankovic, Ruiqing Zhu, Ram Poornalingam, Chenyang Lu, Zhendong Yu, Marty Humphrey, and Brian Ellis. VEST: An Aspect-Based Composition Tool for Real-Time Systems. In *IEEE Real Time Technology and Applications Symposium (RTAS)*, pages 58–69, 2003.

[TC94] Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, 1994.

[Tin94] Ken Tindell. Adding Time-Offsets To Schedulability Analysis. Technical Report YCS-1994-221, University of York, Department of Computer Science, 1994.

[TKK+98] Jorma Taramaa, Munish Khurana, Pasi Kuvaja, Jari Lehtonen, Markku Oivo, and Veikko Seppänen. Product-Based Software Process Improvement for Embedded Systems. In *EUROMICRO*, pages 20905–20912, 1998.

[TMNLM10] Noël Tchidjo Moyo, Éric Nicollet, Frédéric Lafaye, and Christophe Moy. On Schedulability Analysis of Non-Cyclic Generalized Multiframe Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, page 8 pages, 2010.

[TRA07] Karim TRAORE. *Analyse et validation des applications temps réel en présence de transactions : application au pilotage d'un drone miniature.* PhD thesis, ENSMA, 2007.

[Uni13] Carnegie Mellon University. OSATE: Open Source AADL Tool Environment. `wiki.sei.cmu.edu/aadl/index.php/Osate_2`, Sept 2013. Last access: 10/09/2013.

[Ves94] Steve Vestal. Fixed-Priority Sensitivity Analysis for Linear Compute Time Models. *IEEE Transactions on Software Engineering (TSE)*, 20(4):308–317, 1994.

[Ves07] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.

[Vol04] Giuseppe Volpato. The OEM-FTS relationship in automotive industry. *International Journal of Automotive Technology and Management*, 4(2):166–197, 2004.

[Wal95] Stephen R Walli. The POSIX family of standards. *StandardView*, 3(1):11–17, 1995.

[ZBB11] Fengxiang Zhang, Alan Burns, and Sanjoy Baruah. Sensitivity analysis of arbitrary deadline real-time systems with EDF scheduling. *Real-Time Systems*, 47(3):224–252, 2011.

[ZBG$^+$08] Jochen Zimmermann, Oliver Bringmann, Joachim Gerlach, Florian Schaefer, and Ulrich Nageldinger. Holistic system modeling and refinement of interconnected microelectronic systems. In *Design Automation and Test in Europe (DATE)- MARTE*, 2008.

[ZN13] Haibo Zeng and Marco DI Natale. An Efficient Formulation of the Real-time Feasibility Region for Design Optimization. *IEEE Transactions on Computers*, 62(4):644–661, 2013.

[ZZZ$^+$12] Qi Zhu, Haibo Zeng, Wei Zheng, Marco DI Natale, and Alberto Sangiovanni-Vincentelli. Optimization of task allocation and priority assignment in hard real-time distributed systems. *ACM Transactions on Embedded Computing Systems*, 11(4):85, 2012.

Résumé

Les systèmes embarqués temps réel nécessitent une analyse temporelle pour valider leur comportement temporel.
Afin de réduire le coût de développement, l'analyse doit être effectuée à une phase précoce au moment de la modélisation pour détecter les anomalies de conception. L'analyse d'ordonnançabilité est une des analyses temporelles qui permettent de s'assurer du bon fonctionnement du système conçu. Elle est issue de la théorie de l'ordonnancement temps réel.
Plusieurs méthodes et tests analytiques ont été proposés par la communauté académique mais peu sont les tests adoptés par les industriels. En effet, l'utilisation des tests d'analyse exige une large connaissance des travaux de recherches - mis à jour régulièrement - afin de choisir la méthode la plus adaptée aux systèmes conçus pour les valider ou les dimensionner pour le cas des systèmes qui sont en cours de conception.
Cette thèse s'intéresse à cette utilisation minimaliste de la théorie de l'ordonnancement dans l'industrie, et propose des solutions d'aide à la décision basées sur l'ingénierie dirigée par les modèles. Nos solutions visent à augmenter l'applicabilité de la théorie de l'ordonnancement, à faciliter le choix des tests appropriés et à réduire le surdimensionnement qui peut être généré au moment de la conception. Ces solutions sont implémentées dans un *Framework* appelé *MoSaRT* offrant des fonctionnalités pour les concepteurs (modeleurs et analystes) afin d'améliorer le processus de conception des systèmes temps réel en vue de leur ordonnançabilité.

Mots-clefs : systèmes temps réel, ordonnancement, analyse d'ordonnançabilité, ingénierie dirigée par les modèles, modélisation, conception.

Abstract

Real-time embedded systems need to be analyzed at an early stage in order to detect temporal vulnerabilities. Indeed, software development costs are sharply impacted by wrong design choices made in the early stages of development, in particular during the design phase, but often detected after the implementation. The schedulability analysis is one of the main analyses required to ensure the timing correctness of a real-time system. Indeed, the real-time scheduling theory has been devoted to propose different models providing several levels of expressiveness, and different analytical methods with different levels of accuracy. The utilization of the real-time scheduling theory in practical cases could be profitable. Unfortunately, it is not sufficiently applied and research results have been exploited in an industrial context only to a modest extent to date. Actually, a difficulty faced by the real-time designers is to find the appropriate analysis tests helping to validate properly the system and also to reduce the over-dimensioning. This thesis is interested in the chasm existing between real-time design community and real-time analysis community. The purpose of our work is to fill the gap between the modeling of real-time systems and the scheduling analysis. Then, we propose a decision aiding solution based on model-driven engineering. Our solution is dedicated (i) to increase the usage of the real-time scheduling theory, (ii) to facilitate the scheduling analysis tests choice and (iii) to reduce the pessimism during the design phase of real-time systems.
Our proposal is embodied as a framework unifying the designers and analysts efforts. The framework called *MoSaRT* offers a design language very close to the taxonomy of real-time scheduling theory. Moreover, *MoSaRT* provides an analysis repository concept to enhance the applicability of the scheduling theory and also to improve the way designers check their designs.

Keywords: real-time systems, real-time scheduling theory, schedulability analysis, model-driven engineering, design, modeling.

Secteur de recherche : Informatique et applications

LABORATOIRE D'INFORMATIQUE ET D'AUTOMATIQUE POUR LES SYSTEMES
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Téléport 2 – 1, avenue Clément Ader – BP 40109 – 86961 Chasseneuil-Futuroscope Cedex
Tél : 05.49.49.80.63 – Fax : 05.49.49.80.64