Schedulability analysis of embedded applications modelled using MARTE

Kenneth E. A. Jensen

Kongens Lyngby 2009
IMM-M.Sc.-2009-

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-M.Sc.: ISSN 0909-3192

Summary

Embedded systems are increasingly complex and have tight constraints in terms of cost, performance, energy consumption and reliability. Embedded systems have to be designed such that they correctly implement the required functionality. In the case of real-time embedded systems, the correctness of an application depends not only on the results of the computation, but also on the physical instant when the results are produced.

Modern embedded systems design relies on models for the application behavior and hardware platform. Starting from these models, the embedded systems system-level design tasks are responsible for finding a model of the implementation that satisfies all the imposed constraints. The Unified Modeling Language (UML) is a commonly accepted modeling language for modeling complex systems. Recently, an extension called Modeling and Analysis of Real-time and Embedded systems (MARTE) has been proposed as a standard way to model embedded real-time systems.

MARTE is currently under development, and supports two real-time operating systems (RTOS): OSEK/VDX, an automotive RTOS, and ARINC, which is used in avionics. The first objective of the thesis is to evaluate the currently proposed MARTE extension, for which no documentation is available beyond the specification document, and determine how it can be used to model real-time embedded systems. The main objective of the thesis is then to develop a generic library that is not tied to a specific RTOS, thus allowing designers to use MARTE in the early stages of the design process. Such a generic library is also useful for education purposes, and for enabling the rapid development of other, OS-specific, libraries.

We have implemented the generic library in the open-source Papyrus UML modeling tool. Once a model is developed in Papyrus, we are interested to determine if the system is schedulable. We have proposed an automatic translation technique from Papyrus MARTE models into input for MAST (Modeling and Analysis Suite for Real-Time Applications), which is a state-of-the-art schedulability analysis tool used in the academia. Through the use of examples, we have shown that, by using the proposed modeling method, real-time embedded systems can be successfully designed and analyzed.

Resumé

Indlejrede systemer bliver heler tiden mere og mere komplekse, samtidig har de specifike begrænsninger til pris, ydelse, energi forbrug og pålidelighed. Det er vigtigt at indlejrede systemer bliver designet således at de implementerer de krav der stilles til dem korrekt.
Når man arbejder med real tids indlejrede systemer, er det ikke nok at kigge på resultatet fra en kørsel for at vurdere om et program overholder sine krav korrekt, man er også nød til at vurdere det fysiske øjeblik hvor kørslen fandt sted.

Moderne indlejrede systemer benytter modeler for at simulere et programs opførsel og den hardware det skal eksekvere under. Det er baseret på disse modeler at designet af indlejrede systemer skal finde en model der kan realiseres som hardware og software, og som overholder de krav der er til det. The Unified Modeling Language (UML) er en udbredt og generelt accepteret modelerings sprog til at modelere komplekse systemer i. Der blev for nylig frigivet en ny udvidelse til UML. Modeling and Analysis of Real-time and Embedded systems (MARTE) er blevet designet som værende den nye standard metode benyttet til at modelere indlejrede real tids systemer.

MARTE er stadig et under udvikling, på nuværende tidspunkt understøtter den to real tids operativ systemer (RTOS): OSEK/VDX systemet som er et opreativ system brugt i automobiler, og ARINC systemet der benyttes i luftfart. Det første mål i dette speciale er at vurdere den nuværende version af MARTE udvidelsen, for hvilket der kun er den tekniske specification tilgængelig som dokumentation, og undersøge hvordan MARTE kan benyttes til at modelere indlejrede real tids systemer. Hoved formålet for specialet bliver så at designe et generisk bibliotek der ikke knytter sig til et specifikt RTOS, hvilket vil gøre det muligt for designere at bruge MARTE til at modelere meget tidlig

i en udviklings process. Et generisk bibliotek vil også være brugbart i et undervisningsmiljø, og vil kunne benyttes som støtte i udviklingen af andre RTOS specifice biblioteker end de der er tilgængelige i dag.

Det generiske bibliotek er blevet implementeret i et open source UML modelings værktøj kalder Papyrus. Når en model er lavet i Papyrus, har det interesse at kunne vurdere om det modelerede systemer er skedulerbart. En automatisk oversættelse fra en MARTE model lavet i Papyrus til input formattet for MAST (Modeling and Analysis Suite for Real-Time Applications) er blevet foreslået. MAST er et moderne skedulerings værktøj brugt i akademiske kredse. Gennem eksempler bliver det vist at man ved at bruge den foreslåede modelerings teknik kan designe og analysere indlejrede real tids systemer.

Preface

This report documents the work on my Master's Thesis conducted at DTU Informatics, Department of Informatics and Mathematical Modeling at the Technical University of Denmark, during the period February 2009 to August 2009.

A number of models is created as a result of the work done in this thesis. These models has been included on a CD-ROM delivered with the thesis. On the CD-ROM can also be found the installation file for the Papyrus Modeling Tool that has been used to create the models.

Audience

The intended audience of this thesis is computer scientists and computer science students.
The fields being covered in this thesis are all extensive and to give a full description of them all is not feasible, therefore it has to be assumed that the reader has basic knowledge both about the Unified Modeling Language and the general concept of real time and embedded systems.

While UML is presented in the thesis, there is only a brief explanation of what it is, and the reader is expected to know how class diagrams and associations work. For more information about UML, [2] is recommended.

With regards to real time and embedded systems, the reader is expected to understand terms like tasks, shared resources and scheduling. The workings and

differences of the protocols mentioned throughout the thesis aren't explained unless it is directly relevant for their use in connection with the thesis. This is because, while specific knowledge about the protocols might be required to use them correctly, it isn't required to enable a modeling environment to contain them.

Furthermore, concerning the schedulability analysis, focus is on the how to model the information required, and not on the workings of schedulability analysis. The reader is expected to understand the most basic schedulability concepts, like slack, earliest deadline scheduling, etc.

Aknowledgements

I'd like to thank my supervisor, Assoc. Prof. Paul Pop, for his help in guiding the project, especially when it hit dead ends, and for his feedback throughout the project.

Furthermore I'd like to thank Poul H. Jensen for his guidance and his help in understanding the many terms and concepts in both the MARTE framework and the MAST analysis tool.

Finally I'd like to thank Lisa Agerskov Jensen for proof reading the entirety of the thesis.

Contents

List of Figures

Chapter 1

Introduction

Real time and embedded software is quickly becoming an integrated part of modern society, but developing embedded software creates a number of new issues that should be considered. With embedded software comes a limitation to available resources, and in many applications there are different kinds of requirements to the response time of the software, since many embedded units conducts one or more time critical operations.
A classic example being the fuel injector of a modern car, where the amount of fuel injected is controlled by the cars computer, and should be calculated to the optimum amount before every injection. In an 8 cylinder car engine going at 3000 rpm this has to be done 400 times per second, which means that the calculation of fuel to be injected has to be done in 1/400 of a second, or every 2,5ms.

In order to determine whether a real time and embedded system can run within the timing constraints put upon it, a number of different algorithms has been designed to analyze a system and determine whether it is schedulable, and the amount of spare resources available. For complex systems this scheduling is too complex to be done manually, within a realistic time frame, and instead analysis tools designed specifically for the purpose are used.

Besides from adding new requirements to the implementation and the hardware used, this new dimension also adds complications when trying to model software. UML is a commonly accepted modeling language for modeling com-

plex systems, it has the capabilities to model not only hardware and software, but also situations not directly connected to the implementation of a system. Use cases, business flow, and similar is modeled through UML. Unfortunately there hasn't been any dedicated support for modeling the kind of requirements that real time and embedded software rises, especially modeling timing constraints and scheduling has only been supported through adaptation of models from other areas.

This thesis will look into an extension to the Unified Modeling Language (UML)[18] called Modeling and Analysis of Real-time and Embedded systems (MARTE)[15]. This extension is designed specifically for Real time and embedded systems modeling. While a number of other modeling standards exists for real time and embedded systems, the MARTE specification has been named by the Object Management Group to be the future standard for UML modeling of real time and embedded systems.
The MARTE profile is still in beta, and currently it is very lightly documented. The only source of information available is the official specification for MARTE[21] which describes the objects and elements of MARTE, but from a technical point of view, not a practical.

1.1 Thesis goal

The goal of this thesis is first of to determine the state of MARTE, how does it work, what kind of documentation exists and are there guide available. As MARTE is still in beta, some challenges are expected. A description of MARTE should be made that takes the information available in the MARTE specification and explains it in a way that is understandable for possible users.
To achieve this, it is necessary to first do a study of MARTE, determining how it is designed and how the different elements are connected. This will be done in chapter 3, where the MARTE specification is analyzed and MARTE is explained in a more informal manner.

Once this is done, and the workings of MARTE are understood. The possibility of using MARTE for educational purposes will be examined, here focus will be on the knowledge required to use MARTE, and both knowledge concerning modeling and real time and embedded systems will be considered. If MARTE is to be an industry standard, as it is hoped by the Object Management Group, then it should be possible for everyone working in the real time modeling field to use it, and not just those with in depth knowledge about it.

Finally the last interest of this thesis, is the ability to take the model of a

real time system and have its schedulability determined by an analysis tool, this indicates taking the UML model and using this as input to a real time analysis tool. While doing that will almost certainly include a transformation of the data, to fit the input requirements of an analysis tool, it should not require any additional information to be added to the system, leading to a situation where optimally the conversion should be of such a caliber that it can be automised.

These goals have some common grounds, they require the modeling concept and language to be close to, if not identical with the terms commonly used when describing real time systems. Especially if it is to be used in an educational environment, the naming convention should match what one would expect to be used in teaching, meaning that it would be possible to intuitively understand the objects of the solution. Likewise analysis tools, designed for a general purpose use, will also stick to the common concepts. The exception to this would of course be specialized analysis tools, designed for a narrower part of the real time and embedded software field.

In order to determine whether MARTE lives up to these goals, it will be compared to a general analysis tool, and notice will be made, as to how far it deviates from this. The analysis tool MAST has been chosen both as the receiver of the MARTE model if possible, and as a frame of reference. MAST is a opensource real time analysis system.[10]

The idea of the project, is to make it possible for someone without a specific context, nor an in dept knowledge about real time modeling (basic knowledge will be required), to create a model of a system using MARTE, and then have it analyzed in MAST. In order for this to be possible some kind of transformation has to be done on the MARTE model, so that it fits the specifications of MAST input. The result of the analysis can then be used to update and improve the model, leading to a new analysis, etc. This would make it possible to make conclusions regarding the time critical parts of a system early in its development phase and then as the project developed these conclusions could be refined and improved. The concept is illustrated in figure 1.1

1.2 Scope of the thesis

The MARTE profile is designed to handle all variations of real time and embedded systems modeling, this makes the profile so big, that it is beyond the scope of this thesis to examine it in its entirety. Instead limitations are done, making the thesis focus on the basic of real time and embedded modeling and schedulability analysis.

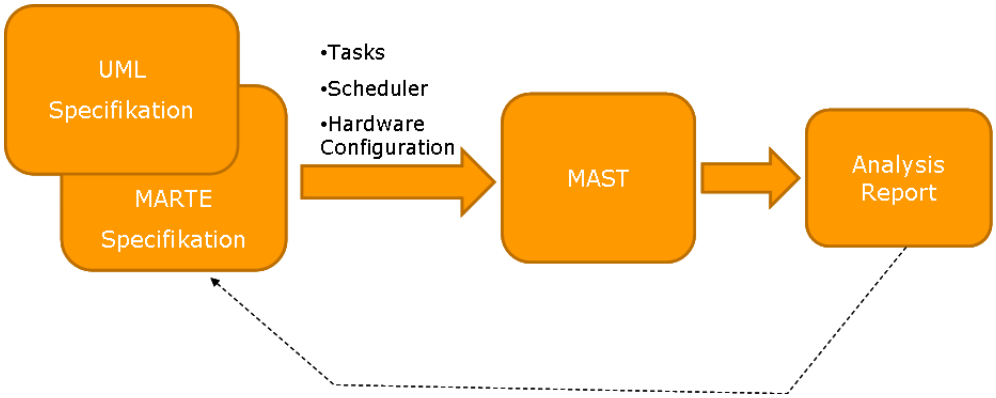

Figure 1.1: The outline of the thesis. A MARTE model is created extending a UML model of a software system, information from this model is given as input to an analysis tool which gives a analysis result. This result can then be used to improve the original model.

Furthermore this thesis focuses on software modeling, while hardware modeling is an important part of a real time system, it is beyond the scope of this thesis to do in depth research on this field also. Therefore attention is put on the software elements.

In the MAST analysis tool, elements concerning network communications are left out, as a result network elements aren't considered in MARTE either.

Being able to use a MARTE model as input to MAST is mainly a question of making sure that the modeling done with MARTE matches the modeling concepts expected in MAST, while this can be observed, the actual transformation of data from MARTE to MAST is not part of this thesis, an example is given proving that the transformation is possible, and the consideration that should be made when creating an automated software system to transform a MARTE model to MAST input is described. The actual implementation of an automated transformer isn't done though.

1.3 Chapter overview

Chapter 3 contains an analysis of the MARTE profile. This analysis will try and answer all the issues raised in this introduction, as well as look into how MARTE is used in its current form. Because the documentation for MARTE

is scarce, the analysis will also contain detailed explanations of the main layouts and elements, however the scarce information also means that a number of the explanations and conclusions reached in this chapter will be based on 'best estimates' based on general knowledge regarding software development, UML modeling, and real time and embedded systems.

In chapter 4 an analysis of MAST will be done, where it will also be examined what information a model should contain for MAST to be able to do an analysis based on it. The analysis will also look at a basic example supplied by MAST and what is required to make this example function

This will be followed in chapter 6 by the creation of a generic library, where, based on the result of the MARTE and MAST chapters, a generic library will be designed and implemented in Papyrus. This chapter will also contain a realization by means of the generic profile, of the basic example introduced in chapter 4.

Finally in chapter 8 the output files of MARTE UML models will be analyzed and an approach to convert a MARTE model into something that can be used as input to MAST and thereby analyzed will be suggested, but although the suggested process works, it is still recommended that it is implemented in an automatic software solution before put in production.

Chapter 2

UML 2

The Unified Modeling Language (UML) is a commonly used modeling language in software engineering. It is a very diverse language allowing for modeling within most fields of software engineering. The UML specifications are created and updated by the Object Management Group (OMG), who is also responsible for approving extensions and changes to the UML specification. UML is currently at version 2, and the change from UML 1 to UML 2 was a big one, bringing with it a lot of updates and changes, including the integration of some of the more well known and popular extensions made for UML, such as the UML profiles for System engineering (SysML), for Modeling QoS and Fault Tolerance Characteristics and Mechanisms or for Schedulability, Performance and Time.[18] While UML is used for modeling everything from software to business processes, it was originally designed for software systems, and it is especially well suited for modeling object oriented software.

The default UML 2 standard does not contain the capability for modeling real time driven events or tasks on a development level, although significant improvements has been made in UML with the release of the UML 2 specification in the real time field, it's still not enough. One of the major improvements that did come with UML 2 was the possibility to model task scheduling, which can now be done with the help of Sequence and Timing diagrams, as shown in figure 2.1 and figure 2.2. Here the task scheduling of a real time system with 3 tasks and a critical resource is modeled.[23] For more complex systems, it seems highly

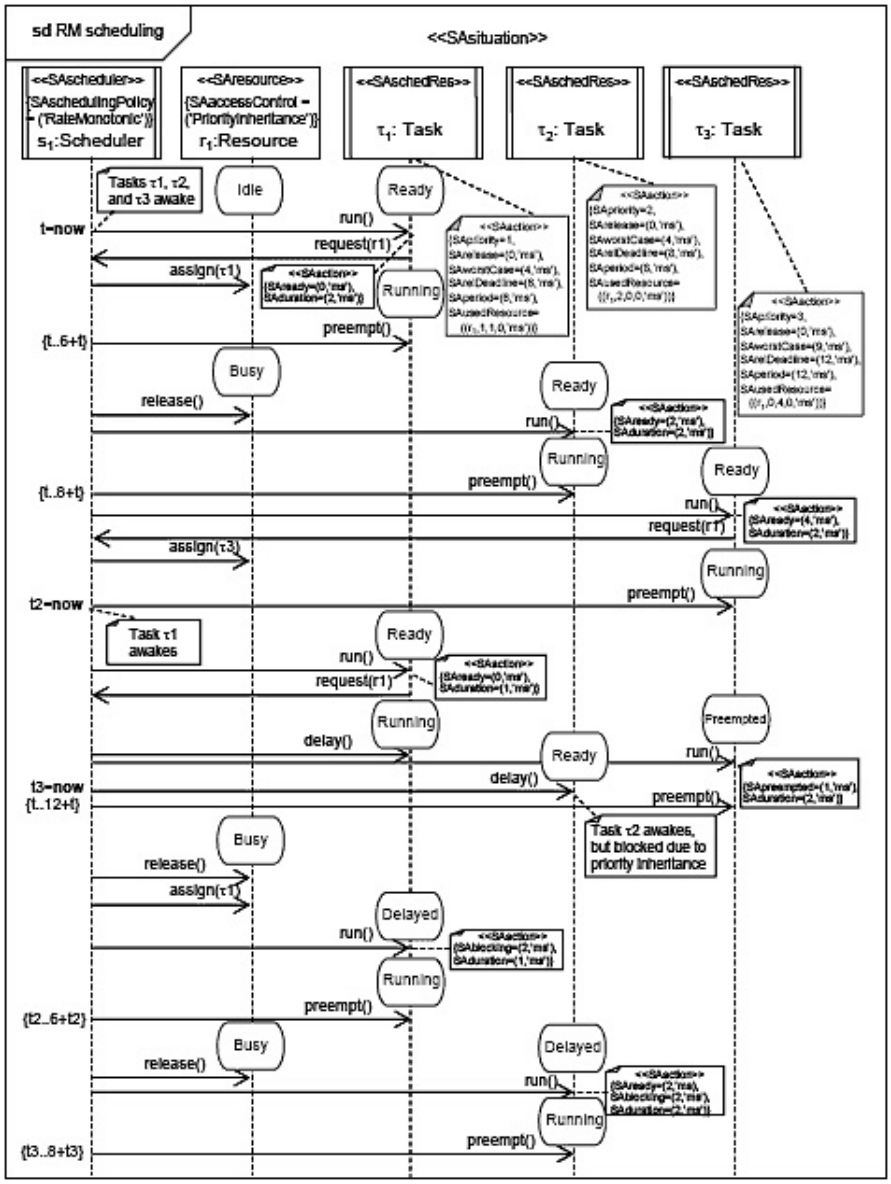

Figure 2.1: Sequence diagram showing a real time schedule

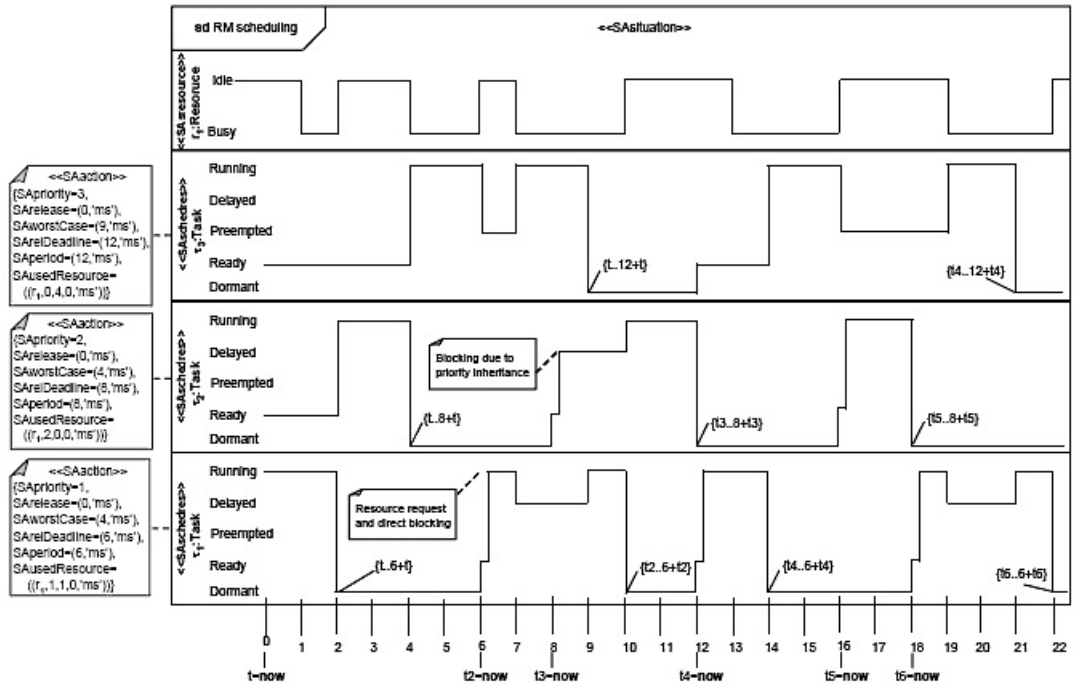

Figure 2.2: Timing diagram showing a real time schedule

unlikely that anyone will do the actual math behind the scheduling by hand, which means that these models would be used to show the result of work done by an analysis tool.

The two diagram types briefly require a bit more explanation to fully understand why they aren't by themselves sufficient. Neither of them were created specifically for real time modeling, instead they are adopted because their primary functions fits the parameters needed to illustrate a scheduled system.
The main purpose of the sequence diagram is to show interactions between objects. This diagram type is very versatile and can be used in both system development, communications or, any other situation where there are interactions taking place in a specific, sequential, order. Sequence diagrams are often used as the next step after the creation of use cases.[1]

The timing diagram is used to show the behavior of objects within specific time periods. There are two main ways of doing timing diagrams, a concise notation, and a robust notation, but because of the level of detail and precision necessary when doing real time modeling, only the robust notation is really us-

able when modeling within this field. Timing diagrams are especially effective in the design of embedded software and in fact real time systems.

What both of these diagram types have in common is that they are meant to show something that has already been scheduled. While it is true that the models can be made as a part of the scheduling process, they are mainly effective at showing a real time system, after it has been scheduled. Neither of them addresses how to model a real time system that isn't scheduled.
It is possible to create a model using basic UML features that includes all elements of a real time system, but it isn't possible, staying within the parameters of those objects, to add all the information needed in order to do scheduling of the system. If one is using sequence and timing diagrams these informations has to be added directly when doing the actual scheduling.

Furthermore, the models are both fairly complex. At first glance it doesn't show obviously what the tasks of the system are, nor the information connected to each of those tasks. If one imagines having a UML model and then creating sequence and timing diagrams to show the scheduling of it, the connections between these diagrams and the rest of the model, especially in a complex system, can be hard to grasp. What would be interesting is a system where the step from modeling to scheduling could be done simply, regardless of the complexity of the system, or automatic.

There has been several attempts during the development of UML at making models capable of making real time modeling possible, this is done through the creation of UML profiles.
An UML profile is an extension to the core UML features. Anyone can create an UML profile, but it does require advanced knowledge about UML and its syntax. The idea with these profiles is to allow anyone with a modeling idea the capabilities to make a modeling environment for it. The Systems Modeling Language (SysML) [17] profile is a good example of such a profile, it is also the profile often used today in the modeling of real-time and embedded software.

This ability to extend UML, making it useful in fields not covered by the core functionalities, is a very strong property, which helps keep the UML specifications up to date, regardless of what changes might occur in the field of software engineering. It is also this ability that this thesis will look further into as the MARTE profile is analyzed in chapter 3

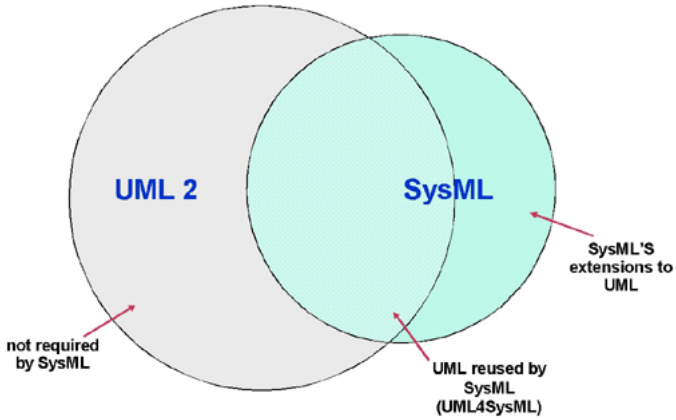

Figure 2.3: Relationship between SysML and UML[17]

2.1 SysML

The system Modeling Language (SysML) is an existing profile to the UML specification with the purpose of enabling modeling of complex systems containing both hardware and software. It takes its foundation in UML, but adds a number of new features to the language. The extensions added makes SysML satisfy the requirements of the UML for Systems Engineering Request For Proposal[11]. As shown in figure 2.3. SysML has been around for several years, the first draft for it was released in April 2006, and was last updated in December 2008 where version 1.1 was released. It is a widely known profile used in a number of different fields for modeling complex systems.[17]

SysML is currently also used for modeling real time and embedded systems, and there is a number of different modeling tools available designed for modeling real time and embedded systems that uses SysML as its modeling concept. A popular one being MagicDraw[6], which has also recently released a preliminary MARTE extension, enabling MARTE modeling. Another one being SysML-Companion[19] which is developed by RealTime-at-Work, and created specifically for modeling real time and embedded systems.
One has to keep in mind however, that SysML wasn't originally designed for modeling this kind of system, it has been adapted to the concept because there was a lack of alternatives for modeling real time and embedded systems in UML. This is one of the reasons for the creation of MARTE, in fact there is currently a MARTE project in the works covering SysML and the link between these two

profiles[15].

Because SysML is the current profile to use when one wants to model real time and embedded software in UML and because of the connection between MARTE and SysML it seems fair to assume that MARTE is inspired by SysML. Many of the features that SysML offers is also present in MARTE, for instance the capability of modeling complex systems and the ability to model both hardware and software. The main difference between the two is that MARTE is designed specifically for modeling real time systems and SysML was adopted for it, as such, MARTE has some modeling features that SysML is missing.

Chapter 3

MARTE Analysis

In this chapter the documentation available for MARTE will be analyzed, and from that the purpose and workings of MARTE will be derived. The chapter will be divided into sections looking at the packages in MARTE that directly affects real time modeling and schedulability analysis.

First in section 3.2 the Software Resource Modeling (SRM) Package will be examined, this package contains objects for representing all the software elements of a model.

This will be followed in section 3.3 by a quick look at the Hardware Resource Modeling (HRM) package, modeling hardware is beyond the scope of this thesis however, and it will only be covered briefly and just enough to cover the basic needs for a real time system.

In section 3.4 the Schedulability Analysis Modeling (SAM) package will be analyzed, but in order to understand this package, brief knowledge is required about a number of packages it inherits from, hence those packages are explained before the actual analysis of the Schedulability Analysis Modeling package is started. The Schedulability Analysis Modeling Package contains all the information necessary to do schedulability analysis, which has been classified as Non-functional, this term covers all information that doesn't directly affect the functionality of a model, the time it takes for a method to execute is considered a non-functional

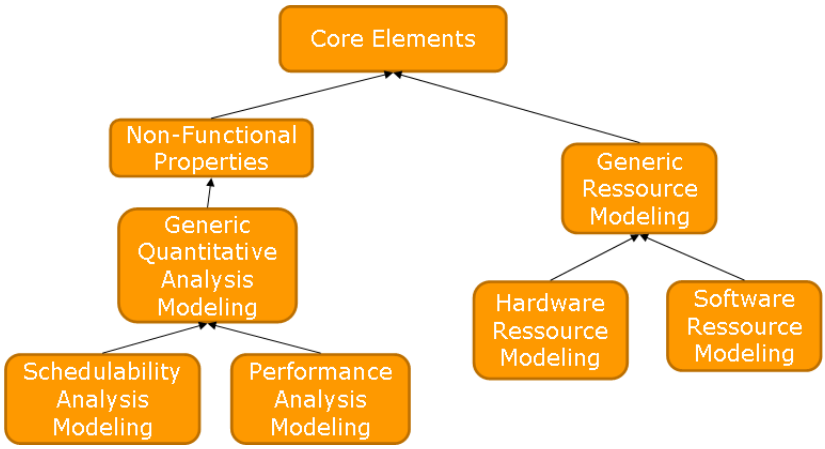

Figure 3.1: The layout and inheritance between the MARTE packages

property. It can roughly be described as the information that isn't directly relevant for the implementation of the model.

These three packages and their parents represents all the objects necessary for this thesis, and once it has been determined how they work and are connected, the chapter will change from trying to understand MARTE, and instead look at how it is used today and could be used in the future. Firstly the limitation or focus of the existing models will be touched upon, followed by a description of how MARTE is to be used. This however is still based on the specification and might therefore contain intuitive leaps at times. Special attention will be given to schedulability analysis and how this is done, since this isn't a part of the core MARTE functions.

Finally the possibilities of making a generic library for MARTE is examined, and a more practical look is taken on the existing libraries, seeing if inspiration can be found as to how one would go about making such a generic library, or what pitfalls might have been avoided during the creation of these libraries. Again because these libraries are undocumented, the observations and results reached will be based on analysis of the implementation of the model.

In figure 3.1 the layout of the different MARTE packages used in this thesis can be seen.

3.1 About MARTE

The only documentation available for MARTE is the official OMG specification[21], this specification defines each object in the MARTE profile, but it doesn't contain a description of how the MARTE objects interact besides from what can be deducted from the UML diagrams available. This means that in order for MARTE to be understood, it is necessary to go through the specification and determine how each object functions, and what objects are necessary in different situations. If one was to do this for the basic UML, one could simply look through one of the many explaining guides available, which explains every object and its connection in detail, but because of the lack of documentation available here this task becomes more tedious.

MARTE is an UML profile designed to add the ability for 'model driven development of Real Time and Embedded Systems' to the UML 2 standard[15]. It is developed by the Object Management Group (OMG), which is also the group responsible for the general UML standard[16].

Modeling a computerized system in MARTE is split into two parts, software and hardware modeling, while these two are obviously connected for any real time system, they are modeled separately and then connected. The idea is that one should be able to model the software and the hardware of a system independent of each other, and if needed, make changes to either part without directly affecting the other. The following sections will look into how software and hardware modeling is done in MARTE, with the main focus on software modeling.

The final step to the MARTE modeling process, after the software and hardware has been modeled, is to model the interaction between them, and thereby add the real time information which is needed for further analysis of the system, be it scheduling, performance, or another kind of analysis. In this thesis the focus will be on the scheduling analysis, and the parts of MARTE that makes this possible. Thus this will be looked into after the analysis of software and hardware. The schedulinganalysis modeling however, is an addition to the framework, and as such isn't necessary for modeling a system with MARTE. This is why it is treated separately.

This means that the first modeling layer done, containing software and hardware, should be complete enough to be viewable on its own.

3.2 Software Resource Modeling package

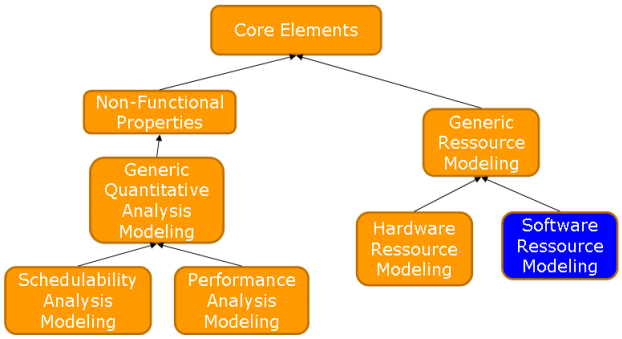

Figure 3.2: The Software Resource Modeling Package will be examined in this section

When designing real-time and embedded software today, there are two commonly accepted approaches, sequential and multi task. In the sequential-based approach, during which the order tasks are run in, is pre-calculated. The multi task based approach allows tasks to be executed concurrently, this approach requires an execution support that handles the real-time features. This is normally done by having it provide the needed resources and services through an API (Application Programming Interface).

In the development of MARTE, the conclusion was reached that it is the latter of these approaches that is the most commonly used approach today, and as such this is where the attention is focused[21]. Furthermore, it was deducted that a real-time operation system (RTOS) is almost always used as execution support.
Unfortunately there isn't one RTOS that can be said to be generally used, instead there are several standardized ones, each with different APIs and different fields of use, for example the OSEK/VDX RTOS which is used in automobile computers, or the ARINC which is used in the aviation industry.
Therefore the purpose of the Software Resource Modeling(SRM) Module is to create a platform that enables design regardless of the RTOS being used. This means that the modeling principals has to be so general that they have room for the flexibility needed, to accommodate all the different RTOS in use. In MARTE this is solved by making the elements of the package very vaguely defined. All the elements are present, but what their purpose is, and how they fulfill that purpose, isn't defined exactly, instead lose general terms are used. The MARTE profile contains all the parts needed to model a real time system, but they are underspecified. To use the framework in conjunction with a specific RTOS, it is necessary to create a model library, defining the API of that RTOS

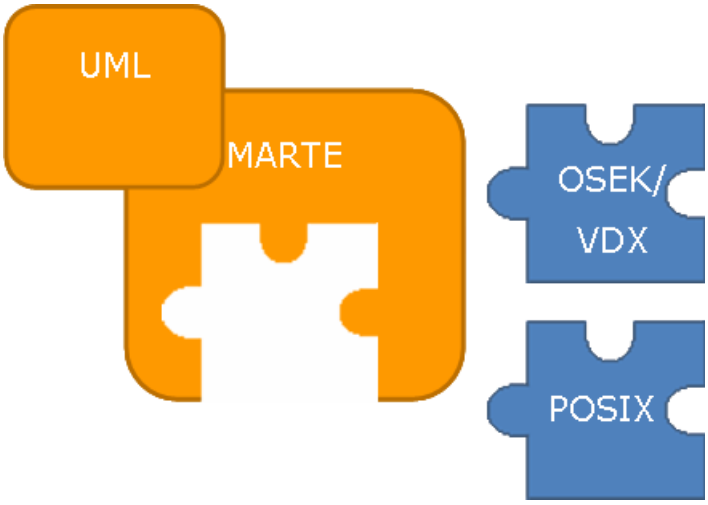

Figure 3.3: The MARTE profile in itself is incomplete, a Real Time Operating System implementation is needed for it to be complete

using MARTE objects, as illustrated in figure 3.3.

And example of this can be found in the official MARTE tutorial[20] where a library, defining the OSEK/VDX library, is defined and used.

While this approach allows MARTE to be used in conjunction with any existing or possible future RTOS (assuming the basics of real time modeling doesn't change significantly), it has the drawback of not working without a RTOS. It is not possible to model a system without having a support library attached, completing the underspecified nature of the MARTE profile. This seems to be a knowing sacrifice made during the development of MARTE. It can be seen reflected in, among other things, the official guide to MARTE where even simple examples are made with the OSEK/VDX library as support. The choice of OSEK/VDX is most likely connected to the fact that MARTE was initially developed in France by VDX.

An example of the underspecification lies in the MARTE object SwSchedulableResource, this component covers all schedulable items that can occur in a model, which basically means everything that is time dependent. Typical examples of SwSchedulableResource are the POSIX Thread, the ARINC-653 Process, and the OSEK/VDX Task.[21]. So in a POSIX implementation of MARTE an object named Thread would be created, which extended the SwSchedulableResource, and likewise for other implementations. As illustrated in figure 3.4.

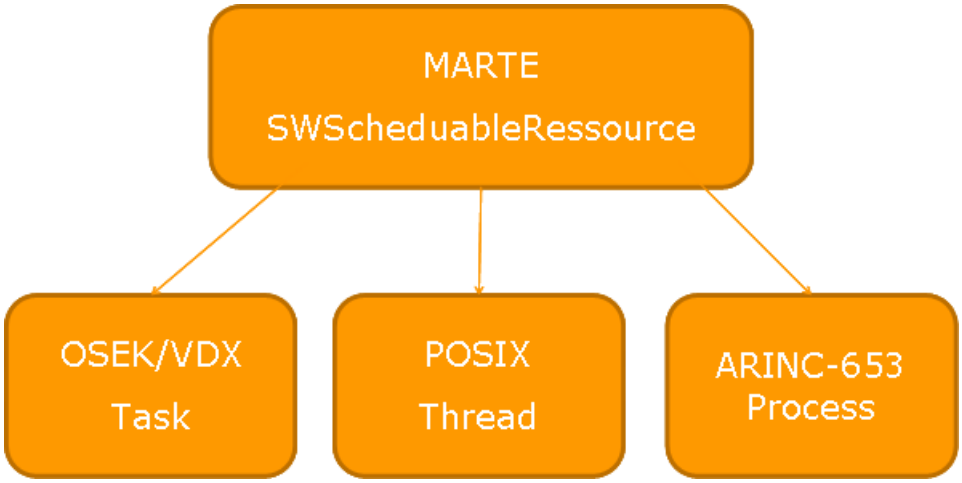

Figure 3.4: The SwScheduableResource as it is seen in OSEK/VDX, POSIX and ARINC-653

Looking further at the SwSchedulableResource object, the underspecification can be seen. All objects in MARTE has a number of properties connected to them, it is in these properties that one defines the parameters for the objects when instantiating them for a model. The SwSchedulableResource is a task, which means that an obvious property for it to have, is a 'Priority' property, since this is used by the very basic scheduling policy of 'highest priority first', which states that of all the tasks ready to execute, the one with the highest priority should always be allowed access to resources first.
This priority can, in theory, be defined as any kind of value that can be sorted, but in most situations it would simply be defined as integers, since this is a common way of sorting elements by rank. In the MARTE implementation however, instead of having a property called Priority that takes an integer as parameter, the property is defined as:

priorityElements: TypedElement [0..*]

This means that that the property 'priorityElements' can contain any number of TypedElements as parameter. A typed element is, according to eclipses UML API, defined as: "A representation of the model object 'Typed Element'. A typed element is a kind of named element that represents an element with a type. A typed element has a type ".[4], this means that anything can be a TypedElement, which is too wide for any actual implementation. Furthermore there seems to be no viable situation where being able to have several priorities

at once is needed, making the fact that the property can have any number of TypedElements as its parameter an obsolete feature.
So if its such a obsolete feature, why is it there? The answer to this lies in the generalization of MARTE, all properties that can be interpreted in more than one way is defined in such a broad way, regardless of their functionality, so even though a few of them could perhaps have been better specified without compromising the flexibility of MARTE, one way of defining flexible properties has been selected and is used for them all. This is what gives MARTE its extreme flexibility and allows it to be used in the diverse manner the developers aimed for with its creation.

3.2.1 Issues with the SRM package

All of this does rise a number of issues, the first one being, that in order to use MARTE, one has to get a hold of a RTOS library, which may or may not be publicly available. Recently a model library for ARINC653 in Papyrus has been released on the MARTE page [22], but this is only one library for one modeling tool (Papyrus). The OSEK/VDX library, which is the one used for the examples, was the first model available for Papyrus and can be now be found on the official Papyrus site[5]. If one wants to model in another tool than Papyrus there is no official libraries released from OMG, and one would have to either contact the support team of the tool, or create the needed libraries.

This leads directly to another issue, in order to model with MARTE, in dept knowledge is required about the RTOS to be used, and it has to be possible to express the model within the limitations of a specific RTOS. This kind of knowledge might be hard to find outside the fields that a specific RTOS is created for. While the car industry has RTOS/VDX and aviation has ARINC 653, smaller fields, or simply more general developments, won't have a specific RTOS tailored to their needs. Instead they would have to compromise, first finding the RTOS closest to what they are looking for. Then learning the API and design concepts of that RTOS, and adapt to that in their modeling.

As previously mentioned, each RTOS has its own naming convention for different objects, likewise, MARTE has its own naming convention, which sometimes seems to fit nicely with a more general text-book naming convention, but at some points differ widely. Again one can only assume that this was done in order to achieve the flexibility needed to support the different RTOS's around, by using a naming convention that doesn't lean on one specific RTOS, but is unique. Unfortunately this also means that using and developing for MARTE requires

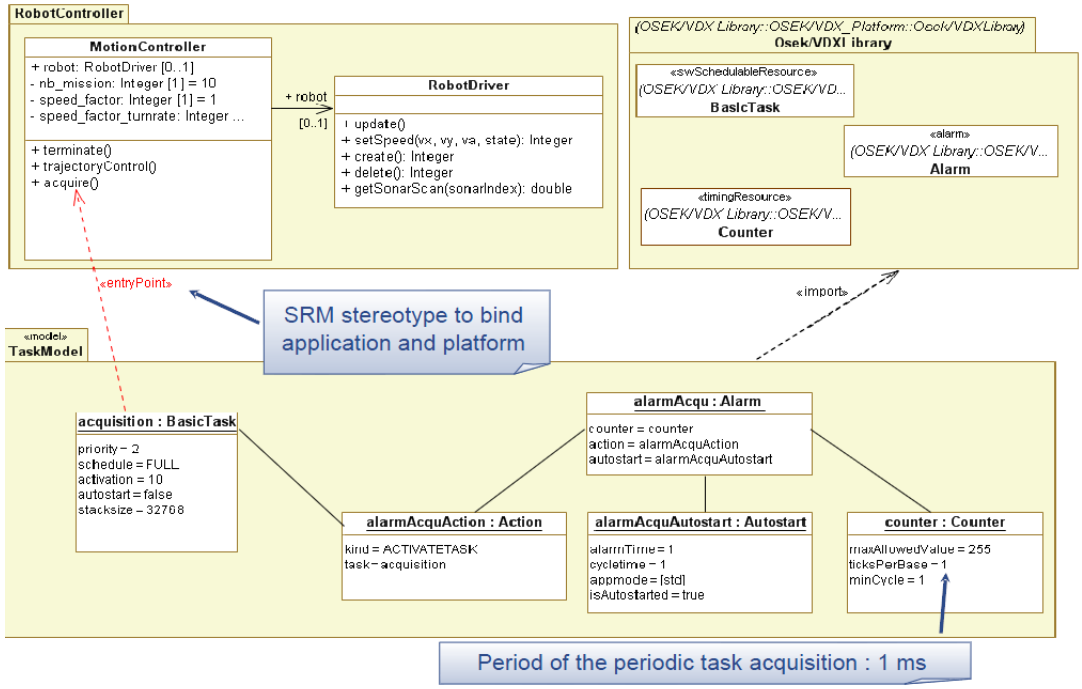

Figure 3.5: An Example modeling of a periodic task, using MARTE supported by the OSEK/VDX library[20]

some learning and understanding of how things are named and connected, and so far very little documentation has been released concerning MARTE, and the use of it. At the same time there is little to no introduction material, tutorials or examples available either.

There is also a question of complexity. Modeling for a specific RTOS requires one to adapt to the modeling conventions of that RTOS, and as a consequence the modeling will not always be obvious or straightforward. Here, one can look at the only modeling example to be found in the MARTE tutorial, which can be seen in figure 3.5. This rather complex model looks like it does, because it has to fit with the OSEK design pattern for a periodic resource, but it isn't intuitive, nor easy to read. This means that in order to use this model, one has to understand the design patterns of the RTOS being used. Not only the designers need this knowledge, but anyone wanting to view the model is required to have insight into the workings of OSEK/VDX, for the model to make sense.

All of this gives the impression, that while OMG sees MARTE as a future

replacement of the current Real-time modeling possibility of UML, it seems to be a product developed with a very specific user group in mind. The only people who can use MARTE off the shelf, is people working within one of the major real time and embedded software fields, and only those with a prior knowledge about one or more of the dominating RTOS's. If MARTE is to truly become a standard for real time modeling in UML, it seems fair to expect a more general usability, so that it can be used by people designing in non-specific environments, smaller fields of interest that doesn't have its own RTOS, or for educational purposes.

The idea of a more general model library, that isn't directed at a specific RTOS, is even touched upon in the MARTE specification, where an example shows how to create extensions to the SRM objects, and expand this into a general API concept. More information about this concept isn't available though, and it is only used in the SRM examples. This gives the impression that while the MARTE development team seems aware of a possible lack of usability for general purposes it was never seen as important enough to allow more than a simple example, showing that this lack could be addressed.

3.3 Hardware Resource Modeling package

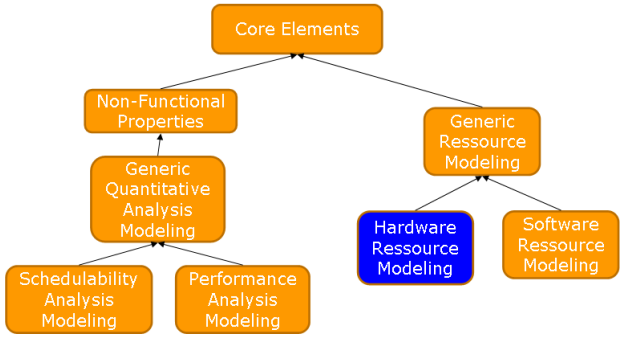

Figure 3.6: The Hardware Resource Modeling Package will be examined in this section

While hardware modeling is almost as big a part of a real time system as the software modeling, in this thesis it is only examined briefly.

The hardware resource modeling (HRM) profile, used by MARTE to express physical hardware, is more complete than the software packet, in the way, that it isn't underspecified, but complete. This means no support library is needed in order to use it. The reasoning behind this seems to be, that the differences be-

tween hardware, in the different fields where real-time and embedded software is being used, is small enough to express in an overall way. This means that none of the RTOS specific model libraries, used to support MARTE, contains any hardware information, therefore the hardware modeling doesn't change when using MARTE, regardless of what RTOS is being used to support the development. This means that it isn't affected by most of the main issues that the SRM has. There will still be a need for any designer to learn how hardware is expressed in MARTE though.

3.4 Schedulability Analysis in MARTE

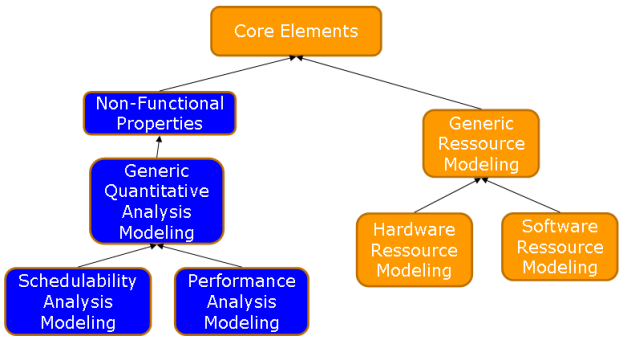

Figure 3.7: The parts of MARTE designed for analysis purposes

The Schedulability Analysis Modeling components of MARTE (SAM), is the elements that allows the modeler to add information that has no effect on how the software/hardware should be created, or implemented. Instead it is information like execution time, access speeds, and others such, soft values. In a software model one can roughly divide everything into one of two categories, functional properties or non-functional properties. Hardware and software elements would, in most cases, be considered functional properties, where as soft values would be considered non-functional properties. They have no effect on the functionality of the system, and give information which, while being important for analysis or similar concepts, is irrelevant for the implementation and behavior of a software system.

In order to represent this kind of information, MARTE uses a concept called NFP (Non-Functional Properties), which in turn is used by the Generic Quantitative Analysis Modeling domain (GQAM), and from this GQAM concept the SAM components are finally extracted. This mean that in order to understand how SAM works, one has to first understand NFP, and then GQAM. However,

even the MARTE description of the these different elements assume you know the concepts they inherit from, and as such only explain what is added on each layer, even though inherited elements is also used. Hence a description of each of these domains is needed.

3.4.1 Non-Functional Properties package

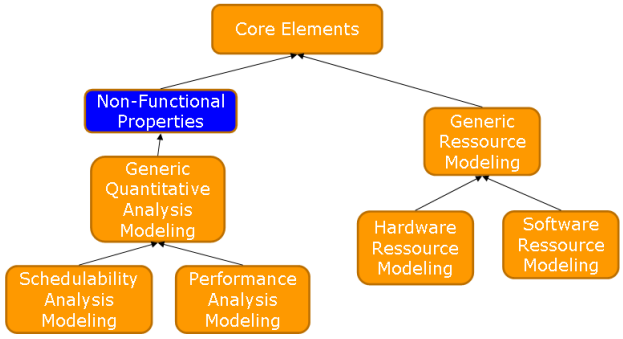

Figure 3.8: The Non-functional Properties package

The NFP concept isn't unique in UML modeling, and the idea of modeling this kind of 'soft values' has been touched upon before in another UML profile, the QoS&FT profile (Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms)[14]. However, it was felt that the NFP model would bring enough new concepts to the field to validate its creation, instead of using the QoS&FT profile.

The main argument is that the two step annotation process that is required by the QoS&FT profile when used is complex for the user, and might create unreadable models. Furthermore QoS&FT leaves out a number of attributes that is required by the MARTE domain. Examples of this is measurement sources, precisions and time expressions, all things that are necessary in order for MARTE to function, and therefore the NFP specification will contains these information[21].

Another UML profile available, which gives a simple approach to specifying 'a set of predefined stereotypes and tagged values'[21] is the profile for Schedulability, Performante and Time Specification (SPT)[12]. This profile meets some of the requirements that MARTE has for the NFP's per default. This profile however suffers from not being formal enough to allow it to be extended for other uses. As already described MARTE is defined in such a way that it is

very flexible and easy to extend, but this is not a rule when making UML profiles, and the SPT profile is defined in such a way that extending it isn't feasible. Both these profiles are worth mentioning however, because MARTEs NFP profile has derived many of its basic components and layout from them, combining their strengths, reducing their weaknesses, adding flexibility, formality, and simplicity to the area.

So what exactly is it that the NFP adds to MARTE, which is so essential? Basically the NFP adds the ability to define non-functional elements, and give them a non-functional type. This means creating elements that are completely independent of the other parts of the model. Additionally the ability to define NFP constrains is added, and finally the concept of a unit is added as an element. A unit is the value type defined to NFPs, since a non-functional property doesn't have units like functional properties which are defined as strings, integers, etc. Non-functional properties are instead defined with units like seconds, and can also be used to express less precise or stable values, like a severity rating defined from a numerical scale.

3.4.2 Generic Quantitative Analysis Modeling package

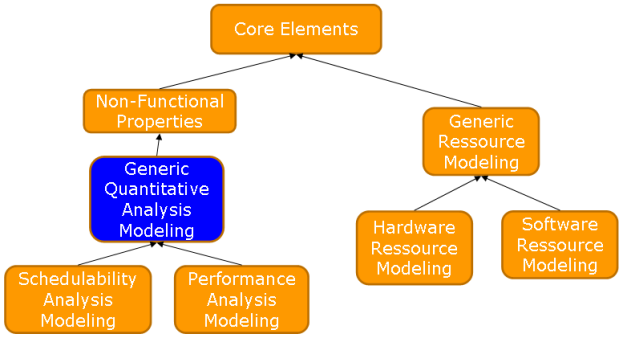

Figure 3.9: The Generic Quantitative Analysis Modeling package

In the same way that the SRM was designed to be a platform for the different RTOSs, to implement their specific features, allowing the flexibility and diversity that MARTE was designed to have, the GQAM is designed to be the platform for different kinds of software analysis. In order to ensure this, it uses many of the same approaches as the SRM did, it defines the core properties that is common for the different analysis, and then expects an extension to be made, and extending these core properties to the exact needs of specific analysis techniques.

The differences between the analysis techniques lies not only in terminology, but

also in the way they use the different NFPs. Situations where an NFP would be considered input in one analysis but output in another is not uncommon. This initially leads to a situation similar to the one experienced in the SRM, of having a modeling environment, which isn't usable in its basic form. To avoid this, MARTE is delivered with extensions to the GQAM that makes modeling possible in two different popular fields, schedulability analysis and performance analysis, supplied by respectively the SAM (schedulability analysis model) and PAM (performance analysis model) modules. This means that unlike software modeling, where MARTE requires an extension library to elaborate its definitions into fitting the API of some RTOS, it should be possible to model some Schedulability analysis parameters of a system, without adding anything to the system. The next chapter, which will look into the SAM module, will further investigate this concept.

3.4.3 Schedulability Analysis Modeling package

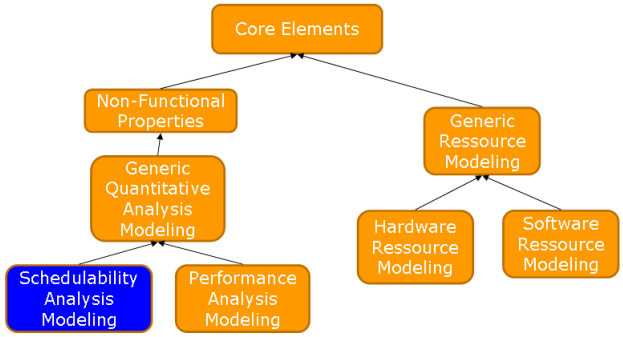

Figure 3.10: The Schedulability Analysis Modeling package

While MARTE provides modules for modeling both schedulability and performance analysis elements, this thesis focuses solely on the schedulability part. This means that while there is a performance analysis module in MARTE, it will not be looked into. Instead focus will be on SAM, the module which extends the GQAM which, as describer earlier, by it self isn't well enough defined to be used.

SAM is designed in such a way that it should allow schedulability information to be entered on the level of detail that best fit the designers wish. This generality should make it possible for users to use SAM in both early stages and on complete systems. In early stages of the design phase SAM can be used to do a high level description of the timing constraints on a system, fitting the knowledge available at that time of development. On finished systems the ability to

do non-invasive schedulability modeling makes it possible to see the results of changes to the system. Both approaches give the ability for developers to detect pitfalls in a system, or 'play' with the different parameters of the system, seeing the cost changes will have, for better or worse.

The MARTE developers have tried with SAM, to create an extension to GQAM that allows for schedulability analysis in a general way, allowing it to be used in as wide a range of situation as possible, both in terms of different system types, and different phases of a systems development. They are however aware, that the price of such generality is that the model becomes complex to use. In general, the more situations they try to accommodate, the higher the complexity of the model. Therefore they also encourage the development of further extensions to SAM to narrower fields, allowing for the creation of models that, while keeping within the parameters of the MARTE specification, is easier to use, or is more extensive in a particular area, but might not be as generally usable as SAM.[21]

The fact that they already in the introduction of SAM decide to point out that extensions might be necessary for it to be optimally suited for some situations, combined with the information found earlier in this thesis regarding MARTE and its intended users, gives an early indicator that schedulability analysis modeling with MARTE might be very complex, and require deep knowledge into the world of real time modeling. This would directly conflict with the goal of having a modeling language that is easy to use. In an attempt to determine the exact level of complexity and the knowledge required in order to model with the Schedulability analysis model, a close description of the main elements of such a model will be given.

It is also worth noticing that among the properties of the SAM elements there are several that refers to information that should be supplied by an analysis tool, so the SAM isn't directly meant to be a model with information that can be entered into a scheduler, but more a model containing information acquired from an analysis tool, and then entered into it. An example of this is the isSchedulable property assigned to an end-to-end flow object, which is a boolean variable, declaring whether the flow is schedulable with its current parameters. This is what would normally be the result of a scheduling analysis, likewise the schedulabilitySlack property, that defines the amount of slack the flow will have with its current parameters, is also an information normally supplied by an analyzer, not one given to it. So some modification might be necessary for the SAM module to be suited as an input to an analysis tool instead of being based on the output of one.

SAM domain modeling relies on two basic modeling concerns:

- WorkloadBehavior: represents a given load of processing flows triggered by external (e.g., environmental events) or internal (e.g., a timer) stimuli. The processing flows are modeled as a set of related steps that contend for use of processing resources and other shared resources.
- ResourcesPlatform: represents a concrete architecture and capacity of hardware and software processing resources, used in the context under consideration.[21]

To keep with the approach set forward, so far, by the MARTE concept of making everything as independent as possible, the goal of the profile is, that it should be possible to model the workload behavior and the resources platform independent of each other, in such a way that should it become necessary to change the architecture on which the system is running, this should be doable without any changes being done to the workload behavior, and vice versa.

Modeling the resource platform of the system, consists mainly of adding NFP values to the hardware and software, defined by the SRM and HRM modules. These modules already express the software and hardware that are relevant from a real time perspective, and thus they represent resources available for the scheduling.

Modeling the workload behavior is, or should be, the biggest part of the SAM model, this is where the flow of the system should be modeled, and it is here most of the parameters that is needed, for a schedule to be calculated, comes from. The WorkloadBehavior is the entirety of the system, in its current form. So the elements found in the WorkloadBehavior should represent all the necessary timing information for an analysis. If one wishes, one could then create several different WorkloadBehaviors, each representing a way to setup the system, and compare these. This would, for instance, be a good way to examine the consequences of changing the deadline of certain flows and similar. The workload model is divided into two parts, end-to-end flows and steps.

The first object in the SAM module to be examined closer is the end-to-end flow. The UML implementation of the end-to-end flow can be seen in figure 3.11. A WorkloadBehavior can contain one or more end-to-end flows, each end-to-end flow representing "'a unit of processing work in the analyzed system, which contend for use of the processing resources"'[21]. This description makes it fit with the concept of a task, a process that needs processing resources. It fits even better when one looks at the properties and objects linked to it, as can be see on figure 3.11. An end-to-end flow is started by one or more stimuli in the form of Workloadevents, these have an arrivalPattern that matches the different types of triggers one would expect tasks to have. An end-to-end flow also has a worst case runtime and a deadline, again things one associates with a task. It

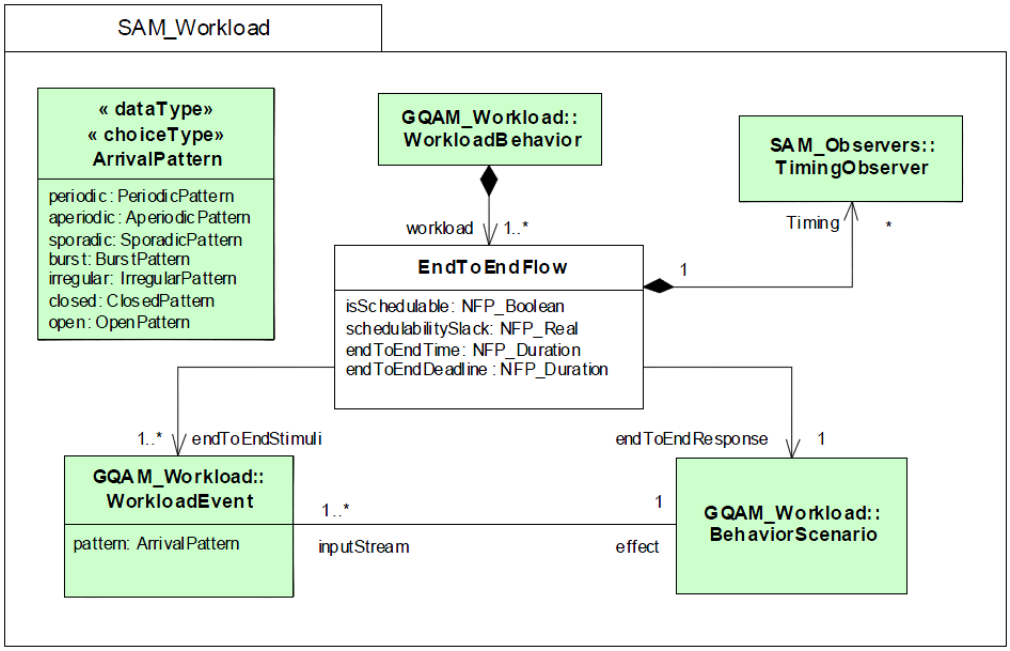

Figure 3.11: The SAM Workload domain model: EndToEndFlow[21]

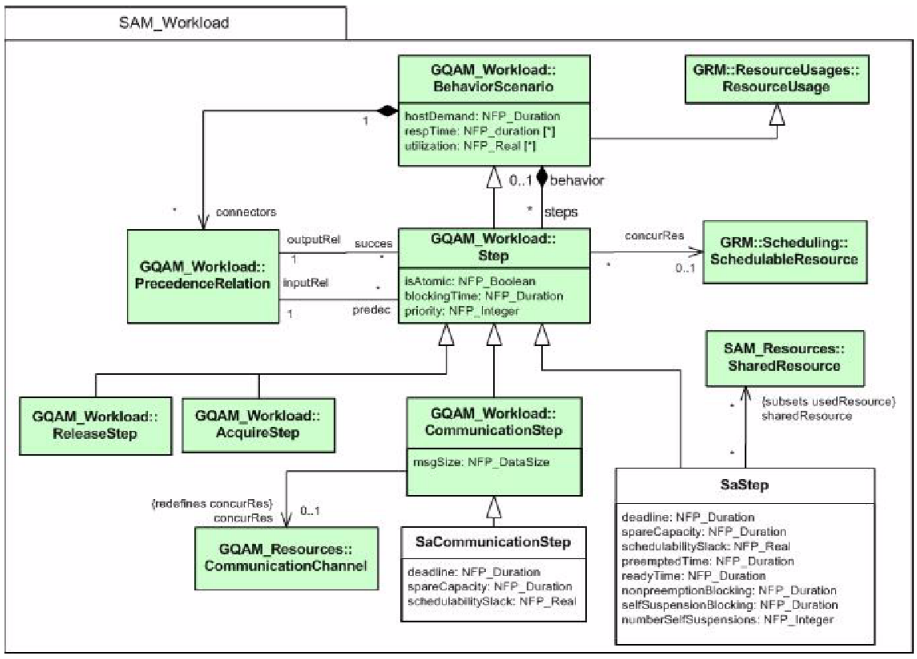

Figure 3.12: The SAM Workload domain model: BehaviorScenario[21]

is also on an end-to-end flow that it is noted if the situation is schedulable and if this is the case, how much slack there would be in the schedule for the flow. So the end-to-end flow contains most of the timing information that is connected to a task. It doesn't contain all the logic a task would contain, it only contains the top level of information, how long does the entire execution take, how often is it started and such. This however, is not enough information. For a scheduling to take place further information is required, like how long will the task require the different resources it uses to be locked, and when in its execution does it require this.

All this is instead defined in a BehaviorScenario. A BehaviorScenario is what the end-to-end flow starts when triggered. It contains the detailed information about what happens during the flow.

In figure 3.12 the BehaviorScenario can be seen. While an end-to-end flow can only trigger one scenario, the scenario can contain any number of steps. Steps are the detailed version of an execution.
Depending on the level of detail being modeled, a BehaviorScenario could simply contain just one step, which would then have execution time and deadline

matching the end-to-end flow that triggered it, or it could contain a number of connected steps, each representing their own little part of the overall code to be executed in the flow. Especially for flows that has to use one or more shared resources as part of its execution time, working on a level of detail that makes it possible to model steps consisting of just the time it needs to have the resource locked, will give a more realistic result of the scheduling analysis. Exactly what is done during the step is of little interest, since in this modeling aspect the only interest is the scheduling analysis. Instead the step needs to contain information like execution time and shared resources to use. To keep the flow of the steps in order a predecessor-successor relationship is used, but this doesn't restrict the execution to being linear, in fact a series of different flow method is available:

- branch (one predecessor Step, multiple successor Steps, each with a probability of selecting that branch)
- merge (multiple predecessor Steps, one successor triggered by any predecessor)
- fork (one predecessor Step, multiple successor Steps, indicating that all successors are executed logically in parallel)
- join (multiple predecessor Steps, one successor triggered by all predecessors completing)[21]

by combining these it should be possible to model all possible flow situation to be found inside an end-to-end flow.
As it could be seen from figure 3.12, the step is implemented as SaStep in the SAM package, but the SaStep inherits from the GQAM step object. One has to look at the GQAM Step object to get a complete overview of the objects available to SaStep. The most important parts have been included in figure 3.12. Elements in green come from the GQAM, and elements in white are created for SAM. Some confusion might arise from there being a reference from Step to SchedulableResource, since a SchedulableResource would normally be associated on the same abstraction level as end-to-end flows, but there has to be a connection between SchedulableResource and Step, in order for the step to have knowledge about processing resources, shared resources and other elements the SchedulableResource contains.

One can see the entire system, starting with an end-to-end flow ending with the steps as the scheduling analysis implementation of a task. The end-to-end flow contains the outer parameters of the task, the arrival pattern, the overall deadline and execution time, information concerning whether the task is schedulable and if so how much slack there is (the last two parameters are information

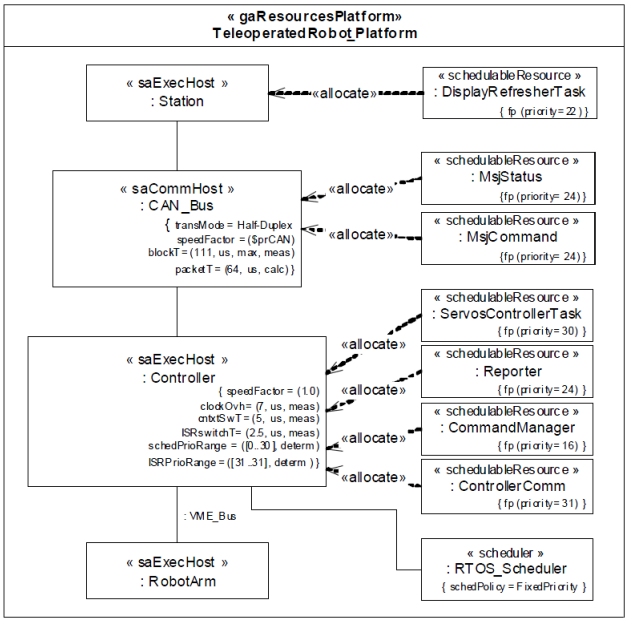

Figure 3.13: An example implementation of the SAM layer[21]

gained after a scheduling analysis has been done). The steps contain the detailed information about the task, what resources it uses and when, how much of its code can be executed in parallel if possible, saving overall run time, but requiring more processing power, etc.

It is also worth looking at an implementation example in the MARTE specification for SAM. In this example, which can be seen in figure 3.13, the processing resources are defined in the SAM layer, as 'saExecHost' objects, and has the tasks connected directly to them, with the scheduler being an input to the 'saExecHost' instead of following the layout in the SRM layer, where the scheduler lies between processing resources and tasks.
This underlines the fact that the modeling layers are independent and modeling can as such be done differently in the different layers.

The reason why its interesting to redefine the processing resources, as it is done in the example, lies again with the independence of the different elements. The information one would be interested in knowing, about a processing resource, will vary depending on what kind of analysis is being done on the system, as such the 'saExecHost' contains the specific information about processing resources that are of interest when doing schedulability analysis.

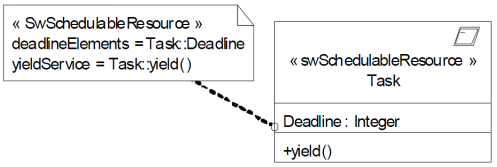

Figure 3.14: An example of how to extend the classes in the SRM, supplied in the MARTE specification[21]

3.4.4 Possible impact of the SAM package on the overall system

The addition of the SAM, NFP, and GQAM modules, has a big impact on the MARTE module. Many of the underspecified parameters in the SRM module, which is extended by the different libraries available, are also defined through SAM. An example of such a parameter could be the task priority property, which in the OSEK/VDX implementation is defined for the periodic task by further defining an underspecified property in the SWSchedulableResource called priorityElement.
Another example can be found in the examples that the MARTE specification supplies with the SRM description, which is shown in figure 3.14. Here it is shown how the classes of the SRM can be extended to fit specific needs. As can be seen in the example, the deadlineElements property of the SWSchedulableResource is further defined as the deadline property that takes an integer as its parameter, but the deadline of a task is a NFP value, and as such is defined on the End-to-end flow. This means that some properties are suddenly available twice. This however, means that the MARTE modeling profile might be usable without a big support library. How many of the underspecified properties that becomes redundant, when SAM is also used, is hard to say without doing a thorough analysis of exactly what parameters is used when doing simple modeling, and as such will be addressed later in this thesis.

This means that when modeling in conjunction with SAM the fact that MARTE is underspecified could be a significantly smaller issue than it is when just doing SRM modeling.

Because of the lack of documentation it is impossible to know exactly why this redundancy in information exists, but two plausible scenarios does seem realistic.

The first one being that the SRM module and these underspecified properties was defined before the SAM profile, and perhaps even the entire concept of NFP was designed. The SRM module seems to contain a number of NFP

values, defined as functional values instead of as NFPs. If the SRM and these properties was defined before the concept of using NFPs and an explicit model for defining schedulability analysis information was devised, it could explain the presence of information that now becomes redundant. This is contradicted however, by the SRM not having the properties defining other NFP values, like arrival type and similar.

The other possibility is that the NFP values are doubly defined because those properties have been determined too important to be left out. It isn't a requirement that a SAM layer is created for a MARTE model, this is just an addition if someone is interested in schedulability analysis. If one then determines that information like the deadline of a task is important in other circumstances than schedulability analysis, having it defined directly on the SRM becomes understandable. Regardless of the reasoning, it is still a non-functional value. Other circumstances where it could be of importance, other analysis situations for instance, should have it defined in the framework if it is of importance to them. It would lower both readability and flexibility of any analysis model, to have the execution time only defined in another model.

However, because of the scarce information available regarding the thoughts behind MARTEs development, this is just something that has to be accepted as is. Because the focus of this thesis is on schedulability analysis, it might be possible to use the redundancy to simplify some areas or at the very least save some modeling work by only using one of the two options. This is only valid if it is guaranteed that the SAM module is also applied to the situation. Because of a design wish for each layer of modeling to be as independent from each other as possible and for the SRM/HRM model layers to be viewable on their own, a certain amount of overlapping information will be necessary.

3.5 Limitations of Extension models for MARTE

The existing libraries live up to the requirements that one would have when wanting to model something to fit any of the current main RTOS's available. However, when looking upon the desired effect of a model, these systems seems to all have in common that they have no focus on modeling all the information that would be needed if one wanted to go directly from a model to an analysis. More precisely, the RTOS's analyzed in this thesis all ignore the time it takes for tasks to execute, be it the time accessing a shared resource takes, or simply the computation time needed by the task to do whatever it was designed for. While the RTOS libraries extend the SRM of MARTE, adding the features of its API, they have no requirements or specifications with regards to how you

define the SAM. In a RTOS, there are no programming parameters for defining these parameters, since they are of little interest to a programmer. Therefore these soft values are obviously not part of any RTOS's API.

Part of the purpose of this thesis was to examine MARTEs usefulness in an educational environment. In such an environment, and similar high level situations, the ability to analyze a system without having to do the actual implementation is very useful. But as discussed in the section about SAM, even though the SAM specification is complete, the method of how to use it, isn't intuitive and requires the designer to have knowledge concerning MARTE and the thoughts behind its design, something that it is currently only possible to learn by going through the specifications for MARTE[21]. Therefore, even though the SAM is complete, it might still be worth considering extending the SAM in such a way that it becomes more user friendly for people without a deep background in neither MARTE nor any of the main RTOS's currently in use.

In general, it should be kept in mind that the different existing libraries, doesn't add new features to the MARTE system, instead they show the features available in MARTE in such a way that it fits the specification they represent.

3.6 How to use MARTE

As already described, MARTE was created to enable the modeling of real time and embedded systems. While the previous sections explain the different elements of MARTE, it is also necessary to explain how these elements are used when modeling.
While it is possible to have a system consisting of a number of tasks and a scheduler, which one wishes to model, this would rarely be the case in actual modeling situations. Instead the real time modeling is part of a larger model, containing both real time critical elements and non-critical elements. More specifically, one can imagine a class consisting of several methods, some of these methods would be time critical, with a deadline, a period or similar, while other methods would be non-critical, simply having to finish when they are done, or having a soft deadline so big that compared to the scale of the system it was of no relevance.

Regardless of the reasoning, such a mix of critical and non-critical methods seems possible. Furthermore when modeling a software system, while the real time parts of this is important in ensuring that the overall systems can run within the restrictions put on it, it has little consequence on how the different parts of the system interact from a developers point of view. MARTE addresses this issue by seeing itself as an extension to a normal UML model of a system,

instead of an entire model in itself. If one was to again look at figure 3.5, one can see that in the top left corner there is a normal UML class diagram depicting the class 'MotionController'. This class has 3 methods, but only one of these are real time critical, the 'acquire' method. By using MARTE it is then possible to extend just this one method through the 'entryPoint' connection, and make a real time definition of it, explaining its period, deadline, execution time, required shared resources, etc. In real time terms, the method Acquire is a task, and should be modeled as a task, which can be done with MARTE, but from a software developers point of view it is a method. A more thorough look at the entryPoint stereotype and how UML models of software systems is linked with a MARTE layer is done in chapter 7 where the results of this thesis is used in a practical example, giving a MARTE layer to an UML model of a primitive system.

The ability to have a normal UML model of a system, and then add MARTE to the locations that are relevant from a real time perspective, is exactly what MARTE was aiming for. In a system consisting of several class diagrams, this would lead to a situation were tasks would be modeled near the relevant classes, spread out over the system. But as long as the entirety is only one system, having one hardware set to its disposal, though this hardware set could consist of several processors, all the different tasks will need to be connected to the same scheduler. This either through a number of secondary schedulers, which is then connected to the primary scheduler, or directly to the primary scheduler.

Normally UML makes connection between elements with associations, but because of the distributed nature of the MARTE elements, this wont always be feasible. Therefore, MARTE has adopted the approach of having the elements that are known to be distributed far from each other, defined as parameters instead. For instance, the SwSchedulableResource has a parameter called 'scheduler', the input for this parameter should then be the scheduler that the task is bound to. Likewise the MARTE scheduler has a parameter called Schedulableresources which should contain a list of all the tasks that it is responsible for. A similar property exists for shared resources which one could expect to be defined at some central location and not near the tasks that needs to access them. Initially this might seem like a complex solution, especially because it requires the modeler to manually update these parameters if changes are made to the system, and it has to be done both ways, but that complexity has to be compared to the gain of being able to spread the task definitions out in the model to the places were they are actually defined as methods. The schedulers and the shared resources can be modeled by them self as a whole since they have little to no effect on the underlying model. For when one works with simple models where distribution isn't an issue, associations is also defined between the objects, meaning one can use these instead of the properties to define the connections.

The important thing to understand about the MARTE profile and its use, is that it allows any system to have real time parameters added to it, and not only during the development. In theory, one could take an existing, complete UML systems model and add the MARTE profile to the parts of it that are real time relevant. Other tools that try to implement the ability to make UML profiles of real time situations often suffer from being invasive in the system being modeled. For instance the MAST Analysis tool has a UML feature[9] which allows users to design and model real time analysis using an adaptation of UML, but this requires the real time parts to be incorporated into the system which first of gives a need for extra knowledge to be present by all designers involved in the progress, and second it creates a model bound to be a real time analysis model.

If one uses the MARTE approach of having the extra elements added, in this case real time capabilities, with as little disruption to the original model as possible, one gets the advantage of being able to design the main system without any insight into MARTE or the capabilities of real time concepts, hence these can be added by someone with knowledge about it. Likewise, it becomes possible to show the system without this extra information when the focus is just the system, and it becomes possible to use the model for other kind of specifications, like the performance analysis extension also available in MARTE. If a UML designed system is to be implemented as software for instance, a software programmer would have little to gain from the real time parts of the model, instead the programmers focus would be entirely on the class, methods and variables of the system.
This approach requires more work when being implemented, because it requires everything to be modeled as their own elements instead of being added onto existing elements, the advantages of such a non-invasive modeling approach is big. A prime example being a situation where an existing real time system is to be changed or adapted to fit a different operating platform. While it might not have been possible when the system was modeled originally to add real time parameters and analysis elements, with the MARTE non-invasive approach, these elements can be added without any changes to the existing model, and the price of any changes can be determined.

Another thing to keep in mind when making this model layer, is what information it is meant to express. It has earlier been mentioned that each modeling layer should be viewable independently, meaning that this MARTE layer can't rely on information supplied by the SAM layer in order for it to be complete. Since it has already been determined that it doesn't contain schedulability analysis information this could be seen as a hard result to reach. Seen in the right light however, this isn't necesarily and issue. This is a modeling layer that identifies and specifies the key elements in a system, seen from a real time point of view, whether it is the SRM elements identifying the important software parts

of the system, or the HRM determining the key resources to be taken into consideration.
It was never the purpose of this layer to allow the viewer to learn everything there is to know about the system, but to identify the key elements, thus allowing for the usage of the SAM or PAM layers to be applied, based on the information supplied.

3.6.1 How to do Schedulability Analysis Modeling

Just like the previous section gave insight into the modeling approach one should use to incorporate the software and hardware elements of the MARTE specification, this section will briefly explain how the Schedulability Analysis Modeling module works. After the SRM and HRM aspects has been applied to a model, one can chose to further expand the overall model by adding schedulability analysis information to the system. This is done by extending the MARTE model the same way the MARTE model extends the original UML model. Modeling with SAM is also non-invasive, and the elements needing to be modeled has already been mentioned in the SAM analysis (chapter 3.4). Because of the way MARTE has been designed, wanting it to be possible to do several different analysis models on the same system without interfering, the SAM model can not be included with the normal MARTE model, but has to be done in its own iteration. This might seem confusing or overly formal, but it will first of all allow for exactly what its creators intended, complete separation between the system model and the analysis model, and second having the analysis elements in their own model gives increased freedom for tweaking features, or trying different changes, without worrying about messing up the original model. The drawback is of course that one has to keep in mind when making permanent changes to one layer of the model, the other two layers has to be updated manually to reflect this.

When creating the SAM layer, it has to be determined how much information needs to be redefined, since the SAM module contains elements defining, for instance, processing resources, which could be defined again here, even if they are also available through the HRM module.

3.7 Making a generic model

While many issues were brought up during the analysis of the MARTE profile, the one this thesis has its main focus on, is the fact that MARTE is created and

focused on a very specific user group, and not usable generally.

Since, as previously mentioned, MARTE is designed to be supplemented by a library, any solution will have to somehow be able to do this. This means either the creation of a new library, or adapting one of the RTOS libraries. The concept of adapting to another RTOS has been discussed already, and the conclusion reached is that it requires learning the RTOS, since one cant be sure it will be well suited for modeling ones system. This has a big chance of becoming overly complicated, like it is seen with the OSEK/VDX example, were a complex design pattern has to be used to live up to the OSEK/VDX system requirements.

So perhaps the creation of a new library is the best way to go? While creating libraries isn't overly complex, it requires knowledge about both UML and MARTE to do. This knowledge however, is only needed by the team creating it, the knowledge required to use the library is only dependent on the design and complexity of the library.

The advantage of creating a new library, would be that the library could be designed to address the issues found in MARTE, by filling in the gap currently there for general users.

A list of the main issues from the MARTE Analysis can be found below:

- Needing a RTOS implementation
- Needing knowledge about the selected RTOS
- Compromising to adapt to the parameters of the RTOS
- Knowledge about MARTE naming convention
- Many RTOS has overly complex modeling patterns to express simple situations

Seen from the perspective that the goal is to have MARTE used by as many people as possible, in as great a variety of design situation as possible, the biggest issues are those that requires the user to have specific knowledge about fields that isn't directly relevant for their work. Unless there is some other big gain, the preferred choice when modeling will be to pick a modeling method that one is familiar and experienced with. The knowledge needed to use MARTE today could therefore potentially 'scare' off possible users.

A possible solution would be to make a 'generic library', binding the many different MARTE expressions, classes and concepts to some non-ambiguous simplified

objects.

If it is possible to create a library with objects and parameters that resembles those used in general terms for real time and embedded systems, not only will general modeling become a possibility, but it will also make the benefits of MARTE modeling available for people without insight into a RTOS. Furthermore, with a general library that stays within the main concepts of real time modeling, it would be possible to use models as input to analysis tools for real time systems available today, and thus have models verified through these.

In this thesis, the analysis tool MAST will be used, as described in Chapter 4, while it might not seem like the optimal solution to lock onto a specific tool, since that implies a need for knowledge about it. MAST is chosen because it covers most, if not all, of the most common analysis methods, and does this while keeping with a general real time naming convention, thus making it a tool that doesn't require extra knowledge about real-time and embedded systems.

Any support library for MARTE can be seen as an attempt to bridge two fields. In the case of the OSEK/VDX library, a bridge is created between the MARTE language and the OSEK/VDX API, enabling something defined in OSEK/VDX to be expressed by MARTE. Making a generic library would be the same, one would have to make a library, in such a way that it is possible to express general models in MARTE. In order to do this, it is first necessary to know what can be defined in general modeling, to ensure that most of the objects used in general modeling is included. Inspiration to what these objects are, will be taken from MAST, since it is possible to analyse most general models with MAST, it seems logical that it must contain all the objects needed for it. The analysis of these objects will therefore be done in Chapter 4.

Once all the elements used in general modeling has been found through the analysis of the MAST tool, the next step in making the generic library is to map these elements onto the MARTE specification. To do so, one has to find the closest corresponding MARTE element for everything one wants mapped. This will be looked into in chapter 6

While this gives a theoretical solution to the generic library, an actual implementation is needed in order to use it. A common tool for UML modeling that is supported by MARTE is the Papyrus tool [5], which has not only the MARTE library implemented, but also the OSEK/VDX and the ARINC 653 libraries. When the analysis of both MARTE and MAST is completed, it is with this tool that the actual implementation will be done.

3.8 MARTE model library

It was determined in the last section, that the best way to realize the goals of this thesis, would be through the creation of a new UML library, this rises the question of 'how does one make UML libraries for MARTE'?
There is no documentation available from the official MARTE page, nor from the OMG group, with regards to how one would go about creating a library, nor what it should contain.

Because of the lack of documentation, an alternative approach has to be considered. Such an alternative approach could be to look at the existing libraries available today and see how they have gone about doing the implementation. No documentation comes with these libraries however and it becomes necessary to instead look directly at the implementations and reverse engineer from that how they are supposed to function. However since they are forced to stay within the UML modeling parameters, and has to use MARTE objects as their source, it should be possible with UML and MARTE knowledge to determine how they work.

Currently there are two known libraries extending MARTE, the ARINC library, which is publicly available on the MARTE webpage[15], and the OSEK/VDX library which isn't publicly available on the MARTE page, but can now be found on the official Papyrus webpage[5]. Both these libraries are created based on the concept that one models the API of the RTOS in question with whatever formalism one wants. The OSEK/VDX library is Object oriented, while the ARINC library is component based, through interfaces. When one has modeled the API completely, the SRM profile from MARTE is used to annotate the resulting model elements, and the semantics from the OS is attached to the elements denoted in the model. Finally the implementations available of both libraries are made in the Papyrus tool.

3.8.1 The ARINC Library

The approach used in the ARINC library is, as mentioned, a component based approach, where all elements of the ARINC API is modeled as interfaces. For example, the Process implementation is shown in figure 3.15, a process is ARINCs version of a task, here one can see that the the component matches the API-interface called Process_management_Service, and has a number of operations at its disposal that matches the operations in the API. At the same time as being an interface however, it is also a swScheduableResource, which means that it also inherits all the parameters and features of the MARTE version of

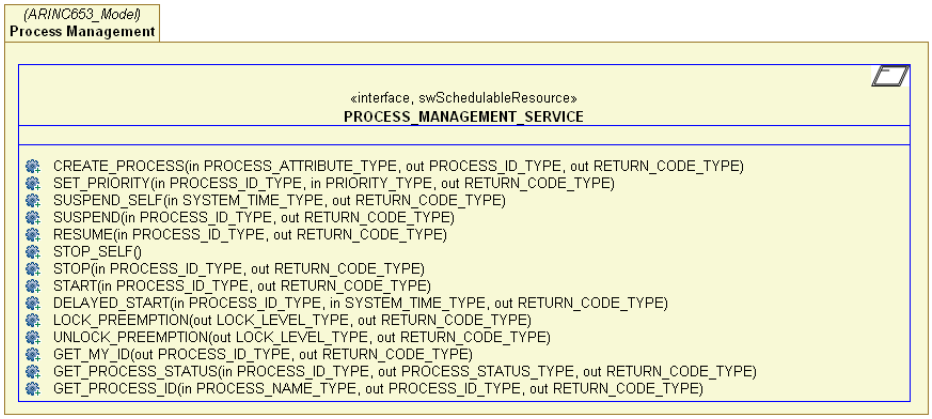

Figure 3.15: The component implementation of the ARINC process in its MARTE library

a task, but the parameters will be of little use to it, since it is an interface and cant have parameters defined as so, instead it will rely purely on the operations that is implemented.
The input for the operations are in many events undefined elements, for instance the implementation gives no indication of what a 'PROCESS_ATTRIBUTE_TYPE' is, and it must be assumed that this is an element that is so simple that it requires no further definition to someone familiar with the ARINC API.

If one instead looks at the ARINC implementation of mutually exclusive resources, which is called semaphores, one gets what can be seen in figure 3.16, just like with the process, the main component is both an interface and a swMutualExclusionResource, meaning that it inherits from both of these classes. Besides these it has a number of operations that matches those of the API for the Semaphore. What makes this example interesting, compared to the process, is the addition of dataTypes, several non-default datatypes are defined which matches some of the input to the Semaphores operations. Such datatypes indicates that the input or datavalues of those parameters isn't available in the default MARTE implementation. MARTE and UML, accepts a wide variety of default input values, covering most common elements like integers, strings, booleans, etc, but if the datatype differs from these it is necessary to define it, which is what can be seen done here.

Unfortunately, there is no documentation available to support the ARINC profile for MARTE, not even a how to use, or explanation, which makes it hard to give practical examples, and at the time of writing, no public description of the ARINC API has been found either, making it hard to verify the assump-

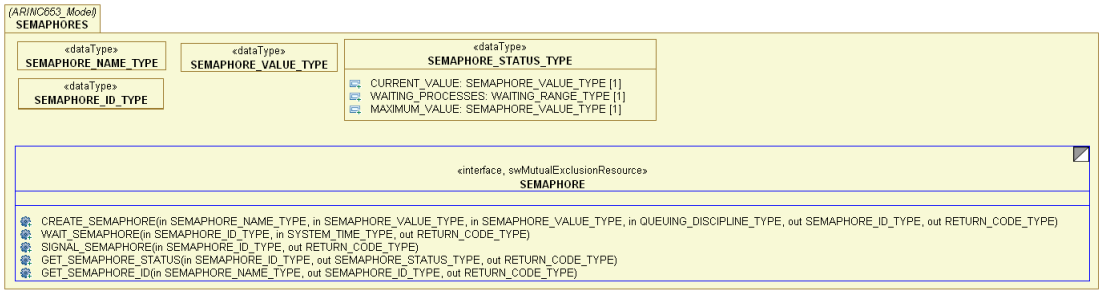

Figure 3.16: The component implementation of the ARINC Semaphore in its MARTE library

tions made. Instead the analysis of the ARINC model is based completely on the assumption that the model library created for MARTE is correct and confines itself to the restrictions placed upon it by MARTE. The ARINC implementation also seems to be without any semantic definitions, but without further insight into ARINC as a system, no conclusions can be reached concerning this.

3.8.2 The OSEK/VDX profile

The other profile that has been available during this thesis work is the OSEK/VDX library, this library doesn't seem to currently be publicly available and as it was the case with the ARINC library, no documentation is available to support the library.

The OSEK library is implemented as an object oriented model, which makes it differ from the ARINC library, but the choice of making it object oriented makes it fit better with UML, which is very well suited for object oriented modeling. As it became evident in the analysis of MARTE, the OSEK library has a number of areas where it is rather complex, these complexities is also evident in the profile.

The first thing to be examined in the OSEK/VDX library is the implementation of a task, in OSEK this element is called a task, and its implementation can be seen in figure 3.17. Here one can see that a task is defined as a class, with a number of parameters, this class then has two children, BasicTask and ExtendedTask, which both inherits not only the Task, but also stereotypes the swScheduableResource stereotype from the MARTE profile. The difference between the BasicTask and the ExtendedTask is merely that the ExtendedTask can spawn one or more events were as a BasicTask can simply execute. In order to match with the OSEK API there is also an interface containing the API

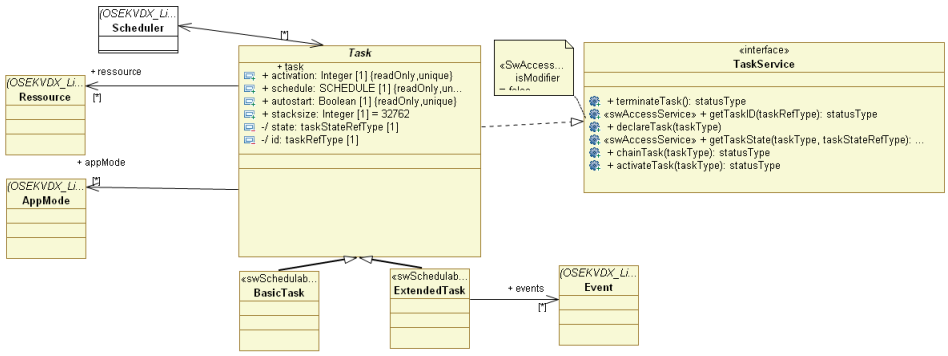

Figure 3.17: The implementation of the OSEK task class

from OSEK concerning tasks, called TaskService, this contains the operations available through the API.
The advantage of this implementation over the pure interface implementation is that while this is more complex to create, it is easier to use. If one wishes to use this model, one could create a BasicTask and it would have all the necessary features available to it.

During the MARTE analysis it was shown that defining a periodic task in OSEK/VDX requires a number of objects to be defined, a task, an action, an alarm and a counter. This setup also has to be defined in the library, in order to enforce that consistency, how this is done can be seen in figure 3.18 where that setup is defined. Here it is defined that any swSchedulableresource of the type 'periodic' has to live up to the PeriodicOSEKTask layout.

While there is no point of going over the entire OSEK library, since most of it is defined just like the task, it does have one last feature worth mentioning. Even though it is relatively simple, the model uses a state machine to define the behavior of a BasicTask, this state machine, which can be seen in figure 3.19, shows a simple way to define behavior, and limit the possibilities of a task. The fact that a task can be suspended allows the task to be interrupted by higher priority tasks that needs the processing resources or similar.

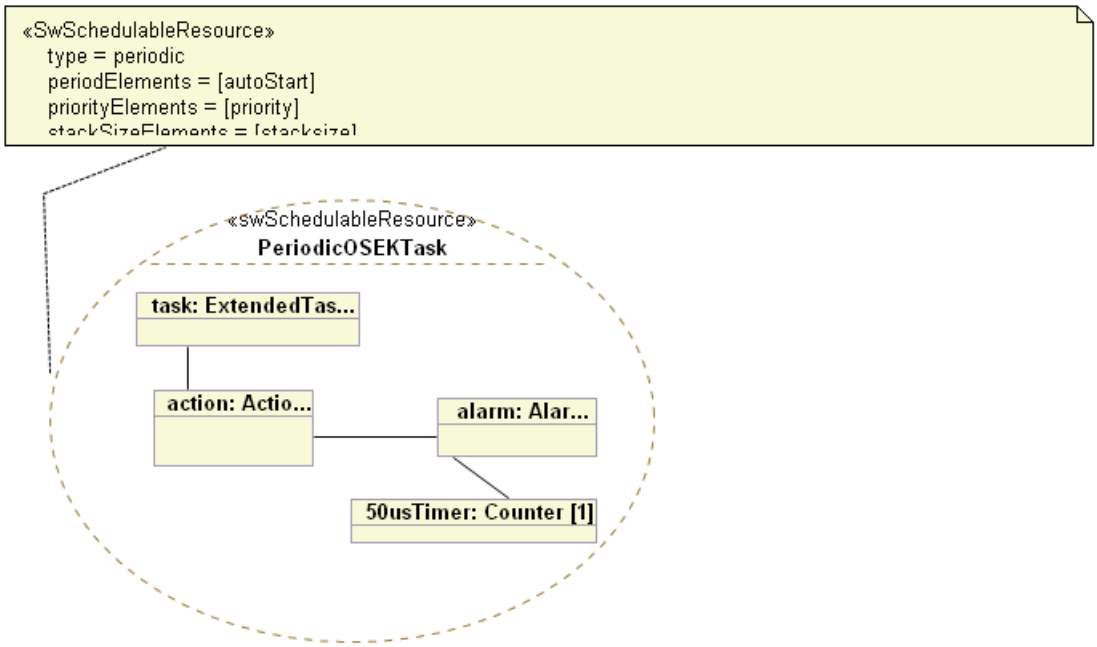

Figure 3.18: The pattern of a periodic task in OSEK

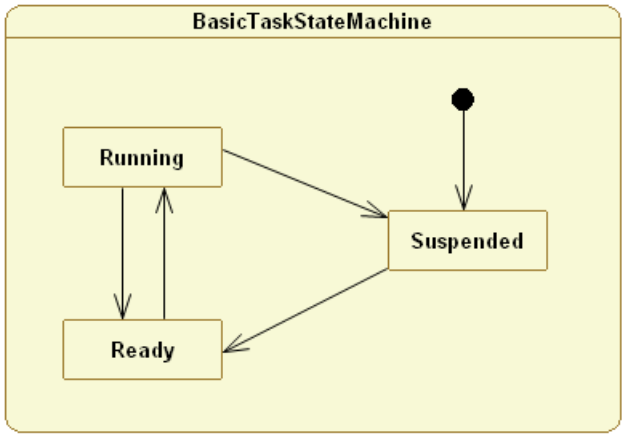

Figure 3.19: A state machine defining the behavior of the OSEK BasicTask

3.8.3 Generic Library

The conclusion reached in chapter 3.7, is that the creation of a new library would be the best way to proceed. Now with the information gathered from looking into the two available MARTE libraries some decisions can be made concerning this generic library.

First off, because there wont actually be an API to model from, the focus of the generic library should be less on interfaces. Already here it becomes clear that the ARINC approach won't be optimal, since it is completely interface based. Furthermore, the ARINC approach is also the least user friendly when it comes to actually using the library for modeling, and since one of the major points of the generic library was that it should require as little new knowledge as possible, the solution that is easiest usable for modeling should be chosen. Therefore the object oriented approach used by the OSEK/VDX library seems to be the best choice, and will make the library more complex to develop but easier to use.

Another thing apparent from both libraries is, that while they inherit the MARTE classes for the different objects, they seem to use very few, if any, of the parameters and settings defined for those classes, making it appear as if the inheritance is mainly done in order to get the objects to be of the right type, and not because there is anything useful in the classes they inherit. In the case of the OSEK/VDX library however, this might just be appearance because it's impossible to say for certain whether it uses the MARTE features for schedulers and similar, since the complete lack of examples and documentation prevents this. Since the generic profile doesn't have a strict syntax it needs to follow, and the MARTE objects already contain a high number of useful parameters, simply adapting and using their parameters wherever possible gives not only the least amount of work, it also insures the best compatibility with MARTE. It also seems redundant to spent time on inventing something if it has already been done, still it might be necessary to rename parameters and object so that they fit with the general naming conventions for real time and embedded systems.

3.9 Summary

In this chapter the MARTE specification has been examined and the information provided by it has been gathered in a reader friendly way. The main packages relevant for both basic real time modeling and schedulability analysis has been identified and the connections between these packages have been defined.

MARTE has been determined to be a modeling profile designed by and for users doing real time modeling within one of the real time fields that uses a specific RTOS for execution. This coupled with the lack of documentation makes it difficult for users that doesn't work within one of these fields or are experienced with real time and embedded modeling to use it.

As a solution to this the creation of a generic library is suggested that simplifies the MARTE specification, this generic library should only contain the elements necessary for doing basic real time modeling, and should be easy to understand and use.

The generic library should be based on the MAST analysis tool to determine what information should be provided by a basic real time system for it to be schedulable. The requirements of MAST will be examined in the next chapter.

Chapter 4

MAST

This chapter will take a closer look at the MAST analysis tool.
First a description is given of MAST in section 4.1. In this section the major issues in MAST will also be addressed.

Next, the different elements of MAST are listed and their purpose is described. This will be the elements later used in creating the generic library. This is done by going over the official list of MAST elements supplied by the MAST documentation[3], adding brief explanations to each part.

Once all the elements of MAST has been described, an example supplied by MAST[8], to demonstrate the basic concept of modeling, is used to determine which MAST elements is necessary to do a basic example, and how they are connected. This will later be used to create the core functionality in the generic library.

The elements not used in the basic example and as such not covered in that section is then described, and their viability for the generic library is determined, keeping the focus of the thesis in mind.

4.1 What is MAST

MAST (Modeling and Analysis Suite for Real-Time Applications) is described by its authors as "'an open source set of tools that enables modeling real-time applications and performing timing analysis of those applications"'[10].

While the advantages of MAST, and why it was chosen in combination with this thesis, has been touched upon in previous chapters, mainly the fact that it accepts the most common real time models, and has the most used analysis methods at its disposal is an important factor here. While it isn't necessary to bind one self to a specific analysis tool, by using MAST as the foundation for the generic profile, it can be guaranteed that at the very least, the most common models will be available in the library. The idea is that in the final product, the user will freely be able to use another analysis tool and the profile should still cover it, MAST is just to be used as inspiration and verification for the model.

While MAST is chosen for inspiration, and one of the arguments was its textbook naming style, there is one rather big exception that should be addressed, namely the concept of tasks. In MAST a task is called a scheduling server, which initially seems far from the logical choice for it. In order to understand this name, a deeper insight into MAST is required, especially into the modeling principles of earlier versions. In version 1.2 and before, MAST models didn't contain a scheduler as an element on its own, instead the scheduling policy was defined partly on each task and partly on the processing units, and tasks were then connected to the processing units that was to execute them. This model can be seen in figure 4.1.

In later versions, this method was found to not be satisfactory, and instead a new model was derived, which has one or more schedulers handling all the scheduling and communication with the processing resources, and the tasks connected to this scheduler. This was done, among other things to allow for hierarchical schedulers in the model[7]. This new model can be seen in 4.2, where the hierarchical structure can be seen, and the scheduling servers has changed to 'simply' just being tasks. As will be looked into later, even the MAST description of the scheduling server sees them as "tasks, processes, threads". However the MAST development team decided to keep the name Scheduling server on these elements. This could have been done so as to not confuse existing users, or perhaps the tool still allows for models made using the old setup to be analyzed. This isn't entirely clear from the documentation and any further look into the reasons for keeping the name would be plain speculation, and since the reason isn't important for this thesis it is simply accepted.

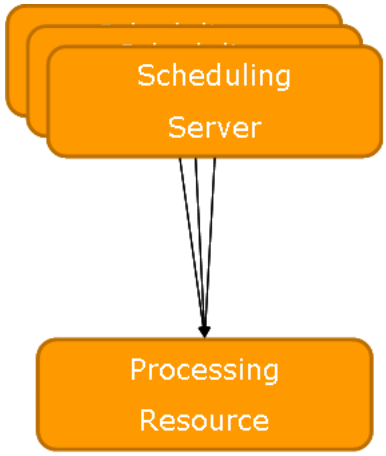

Figure 4.1: Before the update to MAST, the schedulingservers (tasks) connected directly to the processing resources

4.2 MAST Elements

In this section, the elements of MAST will be analyzed, and their significance looked into, with regards to the generic profile. The MAST documentation provides a very complete list of all elements and their attributes, these can be found in [3]. Below the categories can be seen accompanied by a short description of the elements they contain, and an attempt to relate the importance of these elements to MARTE.

4.2.1 Processing Resources

While this list contains every element in all aspects of MAST modeling, there are some parts of it that isn't relevant here. The first category, Processing Resources, is focused on hardware, this is needed to model a real time system, but as explained in the description of MARTE, hardware modeling is unaffected by the support profile in use. Therefore, it isn't necessary to model hardware in the generic profile, and that entire category can be ignored, seen from a MARTE perspective.

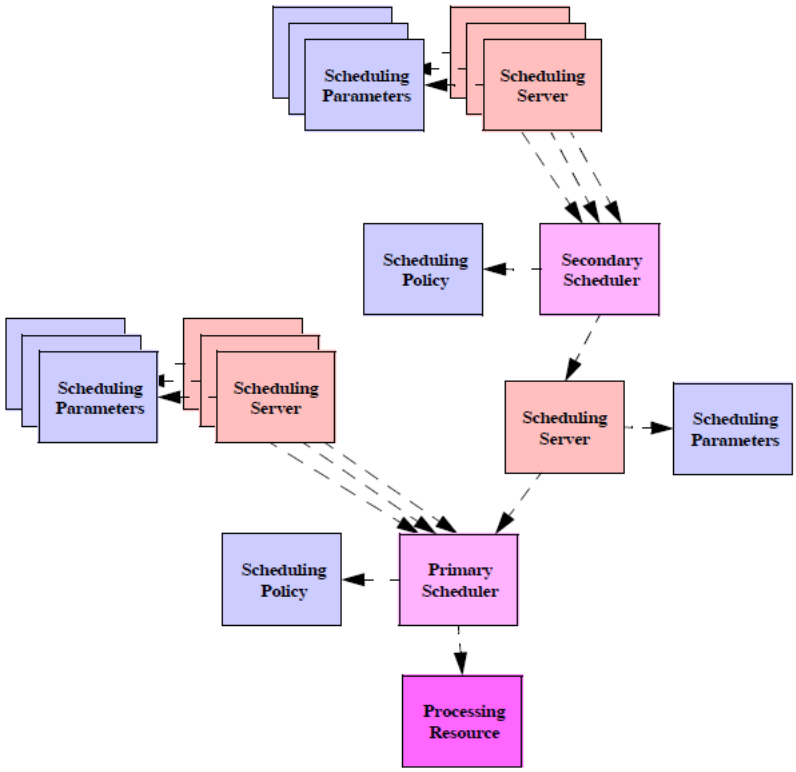

Figure 4.2: MASTS scheduling structure [3]

4.2.2 System Timers

The MAST description for system timers are "They represent the different overhead models associated with the way the system handles timed events"[3], this means that system timers are elements made to define the overhead/delay between an event is supposed to start, and when it actually starts. This is especially interesting if one is dealing with a system using a 'ticker' clock. A ticker is a clock type that works with some predefined period, and every time that period is up all events that has reached their activation time since the last tick is activated. This can obviously cause a delay, especially for big tick periods, and it can also force several events to be executed together, which can cause further delay if they later need to wait for each other to finish, etc.

4.2.3 Network Drivers

Network drivers are defined as "They represent operations executed in a processor as a consequence of the transmission or reception of a message or a message packet through a network."[3]. Basically, they define tasks received through a network, which means tasks received from another system, or at least another physical location. One of the reasons for having these elements is that when something comes through a network, instead of being in the system, additional overhead situations arise, depending on how network transmissions are handled. For this thesis however, the network drivers are not going to be a focus, because they are roughly speaking just an extension to tasks, if at a later point they are found to be needed, they can be implemented as such.

4.2.4 Schedulers (primary scheduler, secondary schedulers,....)

This category contains the description of both the primary and the secondary schedulers, the only real difference between these two elements is that the primary scheduler is connected to a processing unit, and the secondary scheduler is connected to a scheduling server as its executer.

4.2.5 Scheduling Policies (fixed priorities, EDF,...)

This is the different Scheduling policies available, scheduling policy objects are a required input to a scheduler. MAST supports a number of different policies each with their own parameters.

4.2.6 Scheduling parameters (priorities, deadlines,...)

The scheduling parameters are input to a scheduling server, but as described earlier they are used by the scheduler. They are placed on the scheduling server because some scheduling policies allows for several different scheduling behavior to coexist, and the only way to represent this in a system is to have every scheduling server define these parameters on their own. If no parameters is available, the scheduler uses the default settings for whatever scheduling policy it is using.

4.2.7 Synchronization parameters (preemption levels,...)

The synchronization parameters are also a class that is given to the Scheduling server as input, this class defines how access to shared resources should be performed. It is only necessary to define these parameters whenever there is a situation were the Scheduling policy and parameters isn't by themselves enough to do this.

4.2.8 Scheduling Servers (tasks, processes, threads,...)

Scheduling servers, the class's miss leading name was covered earlier. Currently one can only make Scheduling server classes of the type 'regular' as this is the only type viable in new models, but to preserve backwards compatibility older types can still be used, they just wont work with a model conforming to the current MAST model concept. Basically scheduling servers are tasks in whatever forms they might appear, all schedulable entities in a system should be defined as a scheduling server.

4.2.9 Shared resources (for mutually exclusive access)

As the name indicates, Shared resources are elements of the model that needs to be accessed, but are bound by mutually exclusive access. In MAST there are three different protocols for handling this, the Immediate Ceiling Resource protocol, the Priority Inheritance Resource and the Stack based resource protocol (SRP). This is pretty much the only information a shared resource contains, since the actual operations done on it and their runtime is defined in operations.

4.2.10 Operations (procedures, functions, messages,...)

Operations define the concept of doing something, reading from a data storage would be an operation, and the execution times one would associate with that operation would then be the execution time of the data storage. One has to keep in mind that data storage doesn't necessarily mean a hardware module, it could also just be a data server that one contacts and requests some data from. Seen from the operations point of view one is still doing a reading of data, even if what really happens is that one simply requests data from a server, which then proceeds to access the physical data storage, pull the data, and send it. These different levels of abstractions are allowed, and should be used so that one keeps focus on the level of detail currently being worked on.
There are several types of operations available, the example given above was a simple operation, furthermore one can make operations that consists of several other operations added together either as simply a sequence of operations, or merely as part of an operations execution. Finally one can also make message operations which handle the concept of sending messages over a network.

4.2.11 Events

Events are elements used in transactions, there are internal and external events, while there is only one type of internal event (regular) there is several types of external events. The external events represent the way tasks can arrive, for instance as periodic, singular or sporadic, combined with the timing for which this happens. While the internal events represents the limitations of a task after it has been started, for instance a deadline that needs to be kept, these requirements are given to it in form of a timing requirement.

4.2.12 Timing Requirements

Timing requirements are given as input to internal events, they are used to define the different requirements that an internal event is bound by, the different kind of deadlines that an internal event must keep is an example of a timing requirement. Other timing requirements are the definition of the 'max output jitter' or the 'max miss ratio'.

4.2.13 Event Handlers

Event handlers are the actions taken when an event is triggered, and most of the time they result in the generation of one or more new events. There are a number of different event handler types, the most common one being an activity which executes some operation.

4.2.14 Transactions

Finally, MAST uses 'Transactions' to describe the event flow in the system. A transaction is a graph, it shows the external events in the systems and the event handlers that these trigger, this leads to the events created by the event handlers, etc. The transaction doesn't in itself contain any information, its purpose is instead to bind together the events and event handlers so the overall system can be seen, hence the concept of flow is introduced.

4.3 Basic Example

As can be seen from the description of the MAST elements, there are several elements to consider and not all of them are core functionalities, so instead of trying from the first iteration to include all elements in the model, it seems viable to use a different start approach. MAST has a 'basic example' [8] which it uses, among other things, to describe the basis input system of MAST. An approach where an early iteration of the generic profile is created that simply fulfills the requirements of this simple example, and can then later be expanded, seems viable. This example, which demonstrates a basic system, consists of:

- 8 operations
- 4 tasks
- 2 shared resources
- 1 cpu

This is a simple setup of a basic real time situation, but it lacks a scheduler. This lack of a scheduler is a result of the example being created for an older version of MAST, and not being updated when the changes where made to the MAST structure. Unfortunately none of the examples available from MAST is

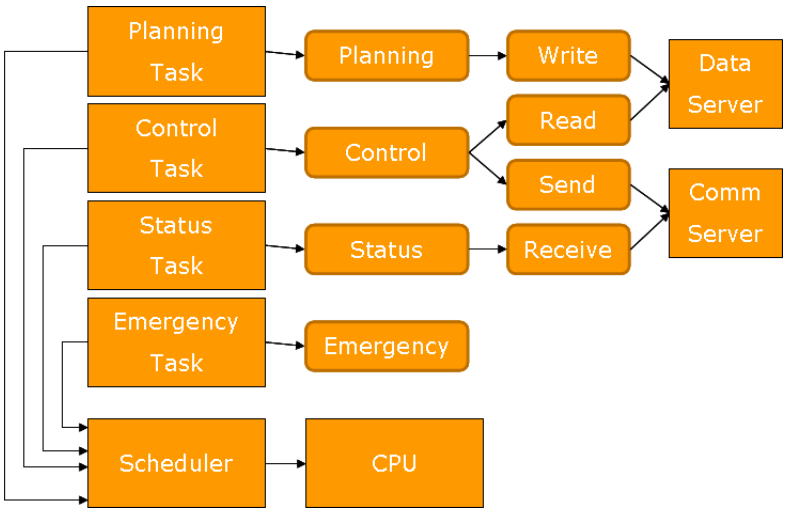

Figure 4.3: The elements in the basic example from MAST and the connections between them, when it is updated to reflect the current MAST modeling concept

updated to reflect its current modeling concept of having a scheduler defined as an element and not as parameters on the scheduling servers.
In order to compensate for this, the example is updated so it reflects the current MAST version. A scheduler is added to the system and the scheduling servers are connected to the scheduler which is then connected with the processing resource, instead of there being a direct connection between the processing resource and the scheduling servers. Likewise the scheduling policy is defined on the scheduler, and the scheduling servers only contain information unique for each task that is required to properly enforce the chosen scheduling policy. In figure 4.3 the updated system can be seen as the current version of MAST sees it, here one can see the tasks being bound by a scheduling server instead of standing on their own. Furthermore one can see that the operations are stand alone objects and not, as one might have expected, parameters of the shared resource that uses them. The example shows the elements needed in order to model simple systems, therefore a list can be derived from it, with the elements and information needed in the generic profile for the system to work.

As said previously, in general the CPU isn't interesting from the perspective of the generic profile, it would be needed in a MARTE model for it to be complete, but by being a hardware element it's covered by the HRM functionality, and as such the generic profile can ignore it.

While one can argue that a shared resource can also be a hardware module,

in case of storage devices of different kinds, in this case they are to be seen as software elements, such as variables, databases or even other systems. In the example given by MAST, the 2 shared resources are a data server and a communications server, while this is rather simplified it serves the purpose of the example nicely. The data server contains data that the tasks can either read or write, the communications server represents an element the tasks can either receive information from or send information to. While these operations are from the tasks point of view basically the same (read and receive is both receiving data and likewise write and send is sending data) they are here to illustrate the different kind of elements that can be covered by a shared resource.

There are two sets of operations in this example, the first set defines the concept of doing something to the shared resources and consists of read, write, send and receive, while these operations are rather simple, and only affects one resource each, MAST does support more complex operations that might use more than one shared resource or similar. The main purpose of these operations is to define the worst case run time for operating on the resource in question, and to define the protected operations, which is operations that works on a shared resource. The second set of operations defines what each task needs to execute, these operations first of all binds together the protected resources that are used by each task, and also defines any runtime that might be needed by the task but isn't connected to the usage of a shared resource, since the first set of operations only defined the worst case execution time it would take to perform an operation on one of the shared resources. In figure 4.3 the operations and how they are connected can be seen. It can also be seen that since this example is primitive there is very little overlap on the shared resources.

Each task is, as explained earlier, called a Scheduling server. Besides from having a name, they contain the parameters needed for the scheduler, if any extra parameters are needed, and the parameters needed to perform synchronization correct, the synchronization parameters are only present when more information is needed for the scheduler than what can be derived from its own information and the scheduling parameters. In this case, a fixed priority is used, which means that each task must define their priority in their scheduling parameters.

After updating the example so that it fits the current version of MAST a scheduler has been added to the system. It contains a scheduling policy and providing a link between tasks and processing resources. Since the example uses a fixed priority policy there is no need to give extra information to the scheduler.

In this example there are 3 periodic tasks and one sporadic, each only executing one activity, this means that 4 transactions are needed, these can be seen in figure 4.4. Because of the simplicity of the example the information

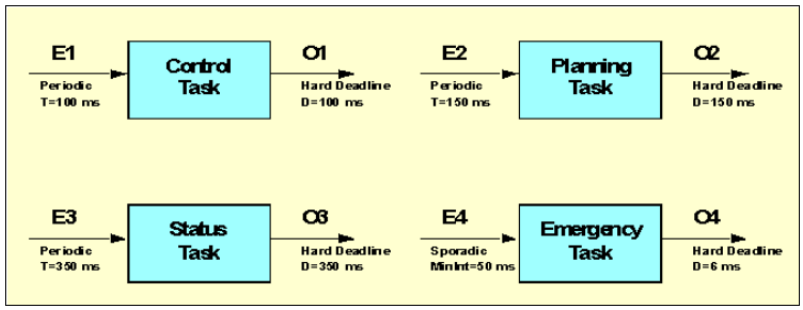

Figure 4.4: The event flow of the basic example, showing the 3 periodic and the sporadic task. E1-4 are External events, O1-4 Internal

supplied by the transactions are:

- The period or minimum arrival time of the tasks
- The deadline type and time
- The operation to be executed

4.4 Advanced features

The previous section went over all the information needed for the MAST analyzer to analyze a simple system. This chapter will look into which other features MAST offers, that should be implemented in the MARTE profile. Initially it is assumed, that every element in MAST to be implemented, so this section will focus on why it might not be feasible or interesting to implement some features, features that isn't mentioned in this chapter, but are present in MAST, should be made available in the model. The reason for not going through every element is that most of them have already been touched upon in the description of the different elements available in MAST. This section reaches the conclusion that a feature should be implemented or left out regardless of what is already available in MARTE. The choices are made based on their relevance for the goal of the thesis, this means that when it becomes time to implement the features, some of them might be implemented as easily as using an existing MARTE object directly.

When adding features to the model, it is important to keep in mind that the purpose of this model was to make an easy to use, easy to understand modeling

concept that can be used by people with little knowledge about real time and embedded modeling and UML. One of the drawbacks of the MARTE profile is that it was created to encompass too many features, the creators of MARTE wanted to satisfy all the different fields of Real time modeling which leads to the very complicated nature of MARTE and the need for these extension library in order for it to be used for anything.

The processing units are still ignored, since they are hardware features, MAST only supports some very basic features on its processing unit, which is well within the capabilities of the MARTE HRM. The more advanced features available in MAST aren't through physical hardware, but network resources, this thesis has already determined that the concept of modeling network situations is beyond its scope, as thus these are not included. For the same reason, network drivers are also excluded from further analysis and won't be supported in the solution. The network drivers in MAST, while adding the ability to model more advanced features, technically only adds a different kind of tasks to the system that can take up resources, but because of the many different ways that network drivers can arrive, they would add unneeded complexity to the system.

The MAST specification contains two different timer types, the alarm clock and the ticker. The alarm clock is a timer that simply activates as close as possible to the earliest event in the system. Basically this means that every event will be started as close to its start time as the operating system in use can measure, this clock type gives the lowest delay.
The ticker has a period, every time that period is up, all events with a starting time that has passed since the last tick will be started. This clock type gives an obvious delay.
While having more than one clock type adds diversity to the system, the ticker clocks seems to have limited use, and while if possible adding it to the system wouldn't hurt, it is hardly worth adding too much to the model for. In worst case situations, having the ticker clock available could confuse more than it helps.

In matter of schedulers, the ability to make hierarchical schedulers is a big part of both MAST and MARTE, and having a secondary scheduler has very little difference from having a primary scheduler. In fact, the only real difference there, lies in the fact that the secondary scheduler gets its processing capacity from a task instead of directly from a processing resource.
Hierarchical scheduling is used especially in systems with different levels of importance on different sets of tasks, so that if some tasks fail or in some way fail to meet their requirements, they will only affect the tasks they share scheduler with. By then having critical tasks grouped on their own scheduler, even if other tasks fail, the critical tasks will get the processing and data resources they need in order to continue to function.
This kind of advanced modeling allows for the design of more complex systems,

while adding very little complexity to the model. Therefore, it is worth including in the profile.

When it comes to scheduling policies however, MAST only supports variations of Fixed Priority Scheduling and Earliest Deadline First scheduling. These are two very basic scheduling policies, MARTE contains a number of other policies that will automatically become available with the implementation of one them. However, depending on how much extra information has to be entered in order to use the other policies available through MARTE, these might not be fully supported.

MAST has three protocols for protecting shared resources, together they cover the most common approaches to resource protection, and should be included for diversity.

The basic example that was examined in the previous section, uses almost all the different operation types available in MAST, the only one not used is the Message_transmission operation which handles messages transmitted through a network, as with all other network elements this will not be implemented in this iteration of the model.

Of more interest are the events available in MAST, since these define the different possible arrival patterns. In the basic example events representing sporadic and periodic arrivals are defined. In addition to those MAST supports singular, unbound and bursty events. Arrival pattern gives flexibility and complexity to a system, since they allow different kind of events to take place, while singular events are primitive elements that brings little complexity to anything. The ability to make bursty tasks, or their extended version unbounded tasks, drastically improve the modeling possibilities.

The last thing to look into in MAST is the different ways to design the execution flow, in the basic example a primitive flow was used, one external event triggering one event handler, which in turn generated one internal event. This is fine for making basic flows, but MAST supports much more advanced flow possibilities, the main ones being the Barrier and Multicast event handlers which gives the ability to make join and forks respectively.
There are many other event handlers which aren't as important but still worth mentioning, the Delivery server and Query server handlers both generate one event even though they have several outputs, the difference lies in the method they use to decide which output to use.
The Delay and Offset event handlers generate an output event a certain time after receiving an input event, again the difference lies in how the delay is determined.
The Concentrator and the Rate Divisor generate output events after receiving

a certain amount of input events, one in case of the concentrator and for the rate divisor it is defined for each instantiation.
The last event handler available is the System Timed Activity, which shares many of the parameters of the activity, but instead of being triggered by an external event, it is triggered by a system timer.

What all the advanced event handlers have in common is that they cant contain operations, only the activity and the system timed activity can be connected to operations, the advanced event handlers are designed only to create an elaborate flow, making the branches needed for the systems run time behavior.

All features of the MAST analysis tool has now been examined and the necessity for them to be implemented in the generic library and thereby available when doing MARTE modeling with the generic library has been determined all information required to do the implementation has been gathered. Meaning that the generic library can be created.
This implementation can be found in chapter 6 but before it is done it is necessary to take a quick look at the Modeling tool in which the library will be implemented, this is done in the next chapter.

Chapter 5

The Papyrus Modeling tool

It was decided early on in the project that the generic library should be implemented using the Papyrus modeling tool. This decision was based mainly on the fact that while a number of tools is referenced by MARTE as usable[15], the Papyrus tool is the one having the strongest backing. Two official support libraries are released for it, and the developers of Papyrus are also involved in the MARTE profile creation. There is a number of issues one should be aware of when using the Papyrus tool, and this chapter will look into those issues.
The biggest issue encountered will be explained in section 5.1, along with the explanation of the problem, the impact it will have on library creation will be covered and a method to overcome it will be devised.

5.1 Missing inheritance

MARTE relies heavily on inheritance between the different objects. A good example of this can be found by looking at the scheduler object. The scheduler is defined in the GRM package, but it is connected to elements throughout the MARTE specification, swSchedulableResource, saExecHost, swMutualExclusiveResource, and the saSharedResource all has a connection to the scheduler just to mention a few key examples.

This is possible because these stereotypes are all children of elements in the GRM package, and the Scheduler is defined to function with their respective parents. It is done this way because the following definition applies in UML with regards to generalization (inheritance):

"'Generalization means that objects of the child may be used anywhere the parent may appear, but not the reverse. In other words, generalization means that the child is substitutable for the parent."'[2]

In this case this means that in a situation where a schedulableResource could be used, swSchedulableResource can be used instead, since it is a child of schedulableResource, thus making it possible to define the tasks associated with a scheduler to be of the stereotype swSchedulableResources. It seems however that Papyrus doesn't accommodate this concept, while Papyrus will accept a schedulableResource as parameter to the scheduler, the swSchedulableResource can't be used.

An implementation example of this can be found when looking at the OSEK/VDX library, and trying to model with this. In this library a new scheduler is defined to fit the needs of OSEK/VDX, this scheduler accepts objects of the type 'task' as parameters, the same way the default MARTE scheduler accepts schedulableResources.
In OSEK/VDX, Task has two children, BasicTask and ExtendedTask which are the objects to be used for creating tasks, the 'Task' class is defined to contain the common parameters for these two classes. If one creates an instance of BasicTask, this instance will not be considered valid input for the scheduler, where a Task instance will be. Since BasicTask is a direct child of Task this contradicts the UML specification for generalization, and the fact that it has been implemented like this in the OSEK library, even though it doesn't work, indicates that the authors of the library also assumed it would work like that.

To ensure that this isn't an error in the MARTE profile for Papyrus, a simple test example was created to verify this fault. In Appendix C a simple Papyrus project can be seen consisting of 3 classes, A, B, and C. B inherits from A, and C takes an object of A as parameter to its only property.
Then creating instantiations of all the classes, it can be seen that while the instance of C will accept the instance of A as its parameter, it wont accept the instance of B which it should according to the inheritance rules of UML. A question has been raised to the Papyrus developers as to whether this is indeed a fault in the tool or not.

Regardless of whether this is a fault in Papyrus, or purposely done like this, it means that for every class created in the generic library, all other classes using the parent of the class has to be redefined and use this new class instead.

Objects like the scheduler, which is already in MARTE well defined and contains the properties necessary to comply with the requirements of the generic library has to be defined in the generic library in order for it to accept the elements being defined in the generic library.

This means a limitation to the amount of 'free' features available with the library, because everything has to be redefined before it can be used.
An advantage of this need for redefining is, that when redefining it is possible to only include the properties that are needed for the modeling requirements of this specific library. This way an element like the swMutualExclusiveResource, which contains 18 different properties of varying degrees of complexity, can be reduced to only contain the ones necessary for the level of modeling being done. It is always, and easily possible to extend the library later to contain more complex properties should the usage of it become more complex and as such require it. This way the solution can be tailored to match the complication needs of any given situation.

5.2 Missing Elements

There are also a few objects from the MARTE profile that seems to be missing or is implemented in a way that doesn't make it possible to use them as stereotypes for classes. While there is no explanation for why a few of these elements are implemented differently, it will be pointed out when the need for using them arises.

For this thesis the only part of the papyrus implementation of MARTE that has been looked at, is the parts necessary for the generic library, whether or not the remaining parts of the implementation is complete is therefore not guaranteed.

5.3 Summary

While there are a number of issues with the Papyrus tool, they are all mainly annoyances. Adapting to overcome them will cause extra work and add complications to the library, but it can at the same time be used to make the library simpler for the end user who models with it.

Chapter 6

Generic Library

The purpose of this chapter is first to use the knowledge gained from the MAST analysis, and turn this into requirements for a MARTE library, more precisely the MAST analysis found the information necessary in a model in order for it to be complete enough for a schedulability analysis. This conversion will follow the layout of the MAST chapter in the sense that it will first look at the basic example from section 4.3 and the conversion of this, once this conversion is done successfully it will move on and expand the model with the advanced features found in section 4.4, converting these to MARTE. Once an appropriate conversion method has been chosen for each element, it will be implemented as part of a UML package in Papyrus.

As seen from the MARTE analysis, in MARTE there is a difference between modeling a real time system, and modeling schedulability analysis information. This is divided into two different models, one consisting of the SRM and HRM modules, the other one consisting of the SAM module. The model being created here should respect this division. This means that when the adaptation of MARTE is done in to make the generic library support the requirements of MAST, the process has to consider not only which MARTE elements it would be best to extend for specific elements, but also in which model layer the information belongs. To do the adaptation with least confusion, it is done in two iterations. The first one will identify and convert all the elements relevant for real time modeling in such a way that the resulting model has enough informa-

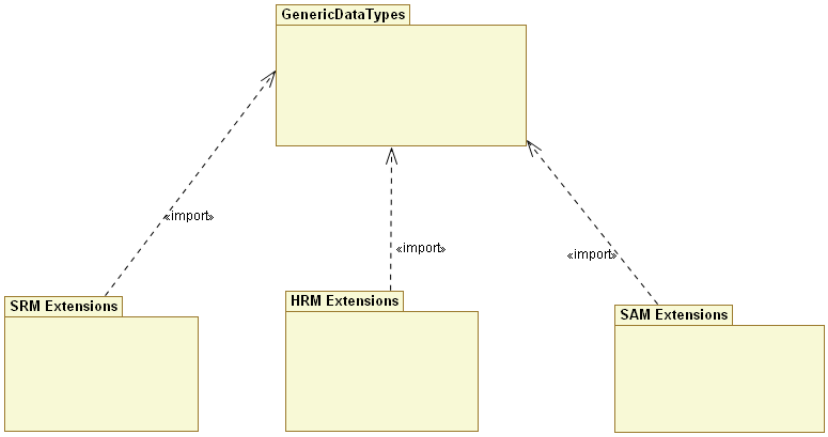

Figure 6.1: The layout of the generic library, the 3 extension packages all import the datatype package to gain access to the common datatypes and enumerations

tion to be viewed on its own and make sense. The second will focus on the NFP values necessary for schedulability analysis and convert these to the appropriate SAM elements.

The generic library, entitled 'GenericAPI', will, to respect the earlier mentioned division, be split up into a number of sub packages.
The 'SRM extension' package will contain the elements that either stereotypes elements from the MARTE SRM profile package or elements determined to fit best into this sub package of all the available packages.
The 'HRM extension' package will contain the elements that stereotype elements from the MARTE HRM profile package. It is meant to be used in conjunction with the SRM extension package to create the real time layer, representing and defining the key elements in a real time and embedded system.
The 'SAM extension' package, like the other packages, contains the elements that stereotype elements from the MARTE SAM package, the classes made in this package is meant to be used as their own model, creating the Schedulability Analysis layer, enabling schedulability analysis of the system.
Finally the library will contain the 'Generic Datatypes' package, this package will contain the enumerations, datatypes, and possible shared types necessary across the packages. The library layout is so that the the three extension packages all import the Datatype package, making the objects defined there available to all three. This concept can be seen in figure 6.1

The library model is an extension of MARTE, therefore it should strive to keep the layout of MARTE. This is ensured by dividing the library into three

packages, each extending a MARTE package, the SRM Extension handles the elements of the MARTE SRM module, the HRM Extension handles the MARTE HRM module and finally the SAM Extension will handle the SAM module. While this has little effect when using the library, it makes it possible to easily see where the different objects of the library comes from, which will make it easier for someone to make changes to the library should such a need arise.

When using the completed Library implementation to create models, the nature of importing means that not only will the generic library classes be available to the user, the many stereotypes of MARTE will also be available. This means that the list of available objects one has to chose from when using Papyrus to generate instantiations of objects becomes overwhelming. To make it as simple as possible for the user to find the relevant classes, those of the generic library, they will all be given a 'Ge' (Generic) prefix.

6.1 Basic Example

The basic example as it is defined in section 4.3 is the first thing to be implemented with MARTE, in order to do so, it is necessary to find all the information the example provides, determine which MARTE objects these information relates to the most and finally decide whether this information can be supplied by existing parameters of that object, or a new parameter needs to be added to the object. In some cases it might even be necessary to create a new object all together.

6.1.1 Real Time model layer

Converting elements to fit the real time model layer is the first iteration of the conversion process, the main focus of it is all the functional values: Tasks, schedulers, hardware and the likes. It will contain all the elements that fits with the MARTE SRM and HRM modules. It will contain the possibility to add information that will also be available in the Schedulability layer, this is a result of it having to be viewable on its own and some information is interesting to have available even when the overall viewpoint isn't schedulability analysis. Which NFP information that it would be prudent to have available in the Real Time layer is chosen based on a number of factors which includes the way it is defined in MARTE and MAST respectively. Since the concept of MARTE is so that every modeling layer is viewable on its own, the information that was found relevant for a layer during the development of MARTE, should be given

consideration.

6.1.1.1 Processing resource

The first element one encounters when looking into the example is the processing unit, while little focus has been put on hardware elements in this thesis, it is still necessary to consider the processing units of the example because of the inheritance problems in MAST, if elements defined in the generic library is to connect to a processing resource, it needs to be defined as an element in the generic library. Hence a class should be created that is stereotyped by the MARTE processingResource, this object would belong to the HRM Extension of the generic profile, since it represent a hardware resource. In the basic example the only parameter defined for the processing resource in use is the worst context switch time, but this is a NFP value, and as such doesn't belong on this layer of the model, but instead in the SAM extension. So in order to accommodate the basic example, no information needs to be added to the processingResource. This makes it a very trivial class at this point in the development.

6.1.1.2 Scheduler

Both MAST and MARTE supports a hierarchical structure of schedulers, this gives a common ground to begin from. Just as MAST requires a secondary scheduler to get its processing unit capacity from a scheduling server, MARTE requires their secondary schedulers to get processing time from a schedulable resource.
Knowing that both systems use the same structure for modeling scheduling systems, means that little has to be done. The basic example only uses a primary scheduler, and as such the secondary scheduler concept doesn't have to be implemented at this level, but ensuring that the approach chosen is compatible with possible extension is important. Since the hierarchical structuring was already supplied, attention is turned to the other information the scheduler provides the system within MAST. A class is to be made stereotyping the MARTE scheduler, since the scheduler in MARTE is defined in the GRM package, it doesn't actually belong in neither the SRM nor the HRM extension, but a layer above. However its ties are strongest with the SRM package since it connects with most elements in that package, therefore this is where it is placed.

In the basic example the scheduler is responsible for the following information:

- The scheduling policy

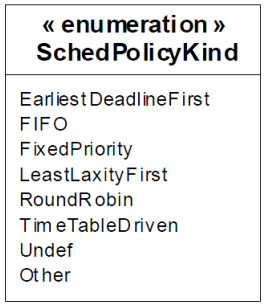

Figure 6.2: The SchedPolicyKind enumeration, showing the different kind of scheduling policies available in MARTE[21]

- A connection to the processing resource
- A connection to the shared resources
- A connection to the tasks using the scheduler

In MARTE the scheduling policies is a property of the scheduler, they are defined in the SchedPolicyKind Enumeration which can be seen in figure 6.2
From the figure it can be seen that the default scheduling policies in MARTE includes the fixed priority policy that is used in the example, therefore no changes is needed in order for this information to be provided. A property is added to the scheduler class matching the 'schedPolicy' property of the GRM scheduler, since there is no upper limit to the length of property names, the new property is called SchedulingPolicy for completeness and will require one parameter of the type schedpolicykind.

In MARTE the connection to the processing resource is also created by the scheduler, through the 'processingUnit' property in which one define the processing resource the scheduler works with. Here again no changes is needed for the MARTE scheduler to live up to the MAST schedulers requirements, which means the property can be directly reused, with its parameter altered to fit with the generic profiles processingResource instead of the original MARTE one.

The shared resources are handled in MARTE by the 'protectedSharedResources' property which takes a list of 'MutualExclusionResource'-elements as parameter, as with the processingUnit property these parameters has to be altered to fit the generic profiles version of MutualExclusionResources instead of the original MARTE ones, and since the scheduler only considers shared resources that are protected, the property can be simplified to 'SharedResources' without any information loss.

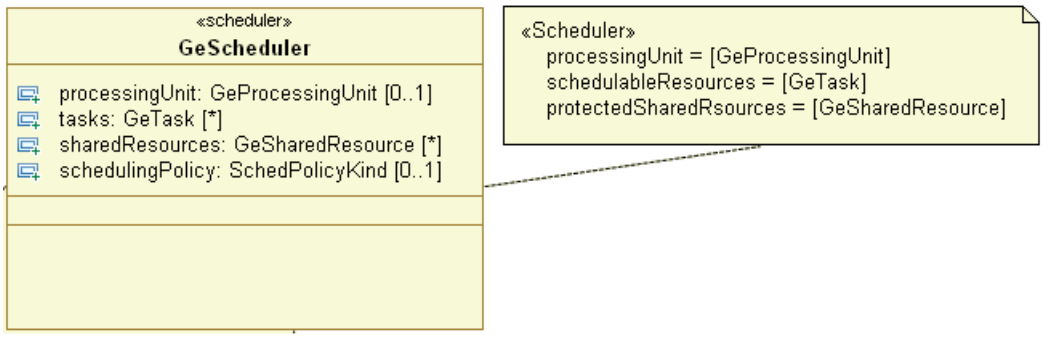

Figure 6.3: The Generic Scheduler as it looks to accommodate the basic example

The last information handled by the scheduler is the tasks, this has already been touched upon earlier, because of the distributed nature of the MARTE implementation tasks and scheduler has to be connected to each other from both sides. On the scheduler side this is handled by the 'schedulableResources' parameter, which takes a list of 'SchedulableResource'-elements as parameters. The generic version of this property is called Tasks and takes a list of 'GeTask' elements as input, GeTask being the class that should be implemented in the generic profile to represent tasks.

The resulting scheduler object can be seen in figure 6.3 with its parameters.

6.1.1.3 Shared Resources

In the MAST example there are 2 shared resources, 'Dataserver' and 'Commserver' respectively, the information attached to each of these are the protocol to use for for the sharing, in case of multiple sources trying to access the resource at once. In the example, the protocol in use is the Immediate priority ceiling resource protocol. Beyond that though there is no information attached to these objects, information like read time or other timed events are defined as operations and thus not actually part of the resource. In MARTE shared resources are presented as a MutualExclusionResource, these objects define the access protocol in the concurrentAccessProtocol property, which takes a protocol from the ConcurrentAccessPolicyKind. The Immediate Priority ceiling resource protocol is defined in the concurrentAcessPolicyKind as 'PCP' (Priority Ceiling protocol).

While nothing else is defined in MAST about these resources, it is implicitly defined that only one element can have access to the shared resource at a time, even though a shared resource doesn't necessarily have a limit of concurrent access of one. A shared resource could be one that can handle any limited amount of concurrent connections, in MARTE this is defined through the 'mechanism' property, in which the method for gaining access to the resource is defined. The one matching the situation here is in MARTE called 'BooleanSemaphore' sharing mechanism, this is also the default value given to the MutualExclusionResource objects in MARTE, it is a very basic mechanism with a boolean flag that tells whether the resource is available or not. Since this is a given in MAST, and the only available method for working with them, a lot of the more advanced capabilities of MARTEs MutualExclusionResources aren't needed.
Besides from the ability to define different mechanisms for access sharing, MARTE also has a number of different Queuing policies implemented if a concurrent access policy is used that requires a queuing concept to be defined. For the policies available in MAST however this is not an issue.

The Generic version of the MutualExclusionResource is called GeSharedResource and stereotypes the swMutualExclusionResource. The property handling the concurrentAccessPolicy is called accessPolicy, and still takes an element from the concurrentAccessPolicyKind Enumeration as its parameter.

6.1.1.4 Tasks

There is 4 scheduling servers in the MAST example, each representing a task. Because of the simplicity of the scheduling policy in use, the only information needed for the scheduler is the priority of each task, which are all well-defined. Had other scheduling policies been in use, like earliest deadline first, the actual deadline wouldn't be defined on the scheduling server but would instead be defined on an event which in turn would be part of a transaction. As it was with the shared resource, no timing information is connected to the scheduling server.

In MARTE tasks are defined as SchedulableResources, and come with a variety of parameter possibilities.
The SWSchedulableResource has a property that defines a priority, but because it is defined to accommodate as many different systems as possible, instead of having the priority defined as anything definite, its defined as a priorityElements, of the type 'TypedElement', basically meaning it can be specified to be anything, based on what the implementing system needs. This means that in order to use it in a case like this with fixed priorities, a definition needs to be made allowing the priorityElements to be defined as simple integers.
An example solution that uses this approach, can be found in the implemen-

tation of the OSEK/VDX model where the priorityElements is refactored by further defining it, the result is a property named 'priority' which takes an integer as its parameter. While it might have been just as simple to add a new priority, name it priority and define its input parameter to be an integer, as so totally ignore the priorityElements property, having these guidelines properties that is then tailored to meet the exact needs of the implementation is a good guideline for what properties an object should have.

The generic version of the schedulableResource is called GeTask, and it stereotypes both the swSchedulableResource and the schedulableResource, this is necessary because some of the properties of schedulableResource isn't available in swSchedulableResource, even though it is a generalization of it.
The properties that needs to be on the task is:
Its connection to a scheduler, which is achieved through the 'scheduler' property that takes an instance of the generic GeScheduler as parameter.
A list of the shared resources it needs to access during its execution, which is defined in the 'sharedResources' property that takes a list of generic GeSharedResources as parameter
The earlier defined priority, that takes an integer as parameter.

Beyond that it has to be decided what other information should be made available, the above lists the information crucial for the task to be able to fulfill its job, but it has to be decided what other properties should be defined on the task in order for the model to be viewable on its own.
That the task is the point with the biggest overlap in terms of NFP values that could fit in both this layer and the Schedulability Analysis Layer is also reflected in the MARTE implementation, by the swSchedulableResource having several properties that can be extended. Information like arrival Pattern and deadline are both NFP values, but at the same time important information for any real time system. An important factor to consider is the availability of the information at the time of modeling, for instance the complete execution time of a task is doubtfully known before the task has been divided into steps defining the actions it has to take, but the arrival pattern of a task is often known when defining the pattern, will it happen periodic? or is it a sporadic event?
Likewise the deadline of a task will in most situation be known very early in the development, a classic example, which is also mentioned in the introduction, is the fuel injection of a car, where the deadline for the fuel calculation can be derived based on information that is independent of the software, meaning that it can be found before any implementation or even considerations with regards to implementation is done. So at least having this information available seems reasonable in order for it to provide a more complete model. This is done through two properties, deadline and arrivalPattern, deadline taking an NFP_duration as parameter and arrivalPattern taking an instance of the ArrivalPattern object from the MARTE 'basic types' package.

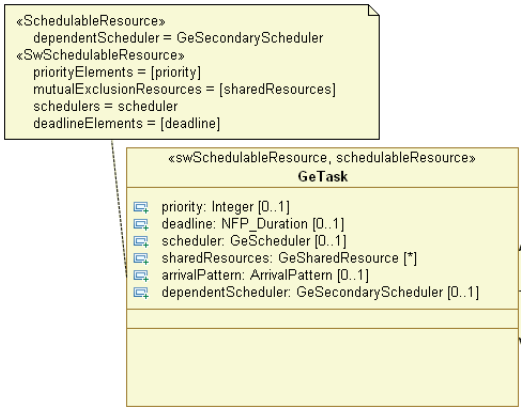

Figure 6.4: The generic version of the swschedulableresource

The resulting class can be seen in figure 6.4.

The four classes now defined in the generic profile is enough to create the first layer of the model for the basic example. The generic library as it looks in completeness and a model of the basic example can be found in Appendix A and Appendix B respectively. The Real time layer consists of all the elements of the basic example, but without the NFP information necessary to do the full schedulability analysis. These information will be gained through the SAM layer which will be looked upon in the next section.

6.1.2 Schedulability Analysis Layer

This section looks upon the expansion of the generic library, which will enable users to create a schedulability analysis model of a system, containing all the information required to do a schedulability analysis of the system, and once that is done represent the results of the analysis in the model. Like the previous section it will go through the relevant elements of the basic example and convert these to fitting classes derived from the MARTE SAM module.

6.1.2.1 Processing Resource

While the processing resource was defined as a hardware element in the real time model layer, it also needs to be defined as a schedulability analysis element, because there is a number of NFP values connected to processing resources. In the basic example the processing unit has a worst context switching of 0.25, which might not be vital to the scheduling of the tasks in the example, but will be for systems with less slack. This NFP value should be defined as part of the schedulability analysis model, in the SAM module this kind of information is meant to be represented by a SaExecutionHost. The SaExecutionHost was created to represent any kind of processing resource, furthermore it contains a property 'ISRswitchTime' (context switch time of ISR (Interrupt Service Routines) interruptions) which can be used to represent the worst context switching time from the basic example.

A new class is created in the SAM extension, called ExecutionHost stereotyping the SaExecHost stereotype. A property is added to it called contextSwitchTime representing the worst context switching time. It's parameter is a NFP_duration like the ISRswitchTimes, which it is based on.

6.1.2.2 Operations

The operations of the example, of which there are 8, contains all information about execution time for the different parts of the system, the time it takes to access shared resources for different purposes and the processing time that tasks needs which isn't connected to a shared resources but is simply computation time. As discussed earlier in the analysis of MARTE, these information doesn't belong in the SRM modeling structure of MARTE, but is part of the SAM module. Therefore it seems fair to assume that all the information gained from operations should be represented in the SAM Extension.

MAST defines 3 types of operations in order to achieve all the different combinations, simple, composite and enclosing. the simple operation is the concept of doing one thing, is has an execution time, and can have a shared resource attached to it. this is the kind of operations one would use to define a read operation from a data storage for instance. Composite and enclosing both define the concept of doing several things, they are both meant to be made up of one or more other operations, a composite operation consists solely of other operations where an enclosing operation consists of other operations but can also have its own execution time that should be added with the other operations to define the final operation, this concept is illustrated in figure 6.5

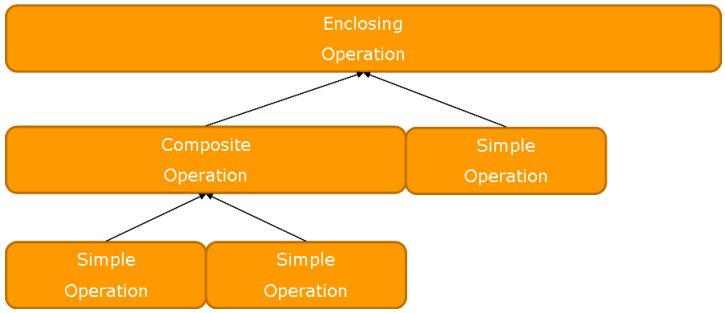

Figure 6.5: The 3 different operation types illustrated. Simple consists only of its own execution, composite contains one or more other operations, enclosing contains one or more other operations and it has its own execution also

The operations can roughly be compared to the steps in SAM. They have quite many things in common, they are both the smallest pieces of the puzzle. The simple operation matches the basic definition of a step, defining the execution of a small amount of code, which could for instance be a read operation. A step however does require more information than an operation, where operations needs a list of shared resources they access and their execution time, steps also needs to contain information about the task it is connected to, this is a result of it being part of the SAM module which is its own model and not a part of the model that initially defines the tasks and the processing resources. Therefore it is necessary to tell the system what schedulable resource the step is connected to. In general because the MARTE modeling is divided into two models, it will be necessary to define some information in MARTE that isn't necessary in MAST, in order to link the two MARTE models together.

Like the composite operations in MAST it is possible to define a step as a WorkloadBehavior in itself, and have it contain a number of steps, creating the same concept as the composite operation. It is however not possible to create an exact replica of an enclosing operation, if one wants to create an enclosing operation with the currently available methods in SAM, one would have to combine the implementation of a composite and a simple operation.
However to understand the implication of this, or lack of same, one has to look at a key difference between modeling in MARTE and in MAST. The reason why creating composite and enclosing operations in MAST is possible, and worth doing is first of that it makes transactions easier to define, but second it allows for reuse of operations. The basic example unfortunately doesn't show this feature, but if a task was added which, as part of its execution, performed any of the things also being performed by existing tasks, like reading from the data server,

an operation could be defined that contained the Read operation, instead of redefining the parameter.
In SAM however, because of the need to connect a step to a specific WorkloadBehavior and Schedulable resource, it isn't possible to reuse the steps. If several different tasks has to do the same thing, it will be necessary to define a step for each of them doing it, even if that means the steps will look almost alike. Likewise the need to simplify in order to make transactions easier to define isn't necessary since the steps are already ordered under a WorkloadBehavior, which could be seen as a composite operation on its own, so regardless of the amount of steps a task is divided into, there will always be a WorkloadBehavior above it connecting them all.
All of this means that the enclosing operation isn't as relevant in MARTE as it is in MAST, and the fact that it isn't easily modeled in SAM isn't a loss worth giving more attention.

Two classes are needed to model the different operations if one is to keep with the MARTE modeling layout. A GeStep class representing the basic steps, and a GeBehaviorScenario class representing the GaScenario.

For the Step most of the information that needs to be supplied through the GeStep class is available already in the SaStep and as such only needs to be extended. The class GeStep is created in the SAM Extension, stereotyping SaStep, the information that it is necessary to supply the GeStep with is:

- Execution Time
- Shared Resource
- Task
- predecessor Steps
- successor Steps

These information are provided partly through the Steps properties, and partly through associations with other objects.
The Execution time is defined in SaStep on the property 'hostDemand' for clarification it will be called 'executionTime' on the generic GeStep however, it takes an NFP_Duration as its parameter.
The connection to the shared resources that a Step locks in its execution is defined through the 'sharedResources' property, which matches the 'sharedRes' property of the SaStep.
as it was described in the SAM analysis, a Step needs to have a connection to the schedulable resource it belongs to, this connection is created by the 'task'

property, which matches the 'concurRes' of the SaStep.

In SAM the predecessor and successor Steps of a Step, is defined through a PrecedenceRelation object which works as a connector between the different Steps. This is implemented as its own class, having a connectorType property defining what kind of connection each instantiation represents. The possibilities are defined in a connectorKind Enumeration. While the basic example only contains operations connected in sequence, and advanced connectorKinds aren't necessary for it to function, they are defined explicitly still to allow for adding more complex types should the need arise.
Unfortunatly the connectorKind and PrecedenceRelation isn't part of the Papyrus implementation of MARTE, whether this is because they seem to be very vaguely defined in the MARTE specification, and as such might not be complete, or simply has been forgotten from the Papyrus implementation, is impossible to say. Regardless of the reasons, these classes has to be added to the generic library without stereotyping the elements they extend, in order for the SAM layer to be complete.

The GeBehaviorScenario is triggered by an end-to-end flow, and contains one or more Steps, in comparison with MAST it is a composite operation. It should contain an execution time, which should represent the sum of the execution time of all its Steps, and it should contain a property defining the first of its Steps to be executed upon triggered. the Steps connected to it is realized through associations. The advantage of the SAM layer over the real time layer is that associations can be used freely, since all elements are defined at a central location and are easily connectable.

6.1.2.3 Transactions

The final component of the example is the transactions, they represent the bindings between the tasks and the elements they need to interact with in order to live up to their functionality.

Just like the operations were comparable to steps, the transactions are roughly comparable to end-to-end flows. Transactions contains external events that triggers the execution of the operation connected to it, the operation when finished then trigger an internal event.
The external event contains all information regarding arrival times for the transaction, and the internal event contains information about when the transaction should finish, these are all information that the end-to-end flow was designed to handle.

Beginning with the internal events, these define both the deadline and the deadline type of the transaction. As earlier mentioned deadlines values are defined on the task as an informational feature, but in the context of a schedulability analysis it is necessary to connect more information to it. Just knowing what the deadline is, isn't enough, it is also necessary to know the type of the deadline, and because the SAM layer should be viewable as an independent model, the deadline should also be present on the end-to-end flow, by means of a NFP_duration type. In the SaEndtoEndflow, this deadline is represented through the property 'end2EndD'.

In SAM, the observer package contains an object called a SchedulingObserver, which inherits the 'laxity' property from the TimingObserver object. The possible input for this property is defined in the LaxityKind Enumeration, and defines the different deadline types available, in the default SAM implementation this is:

- Hard
- Soft
- Undefined
- Other

In the basic example all the transactions use the global hard deadline, which matches the normal hard deadline available in SAM, so using this enumeration would be sufficient.
This means that it should be possible to connect a SchedulingObserver to the generic version of the end-to-end flow containing this laxity information. Unfortunately the implementation of the schedulingObservation in the MARTE profile is done in such a way that it can't be stereotyped, instead a class GeSchedObs is created which doesn't stereotype anything, but is still inspired by the schedulingObservation object from MARTE.

Another event type that the transaction contains, defines the arrival times and type for the tasks. In the basic example this resulted in having periods for the three periodic tasks and the minimum inter arrival time for the sporadic task. In SAM this kind of information is bound to the end-to-end flow and defined in the WorkloadEvent which contains a Pattern property taking input from the ArrivalPattern datatype, which can be seen in figure 6.6.
The possible arrival patterns available through SAM are more than adequate to accommodate the requirements of the basic example. So they are implemented by adding the 'GeArrival' class to the framework which is associated with the GeEnd-to-end flow and has a property 'arrival' that takes an arrival

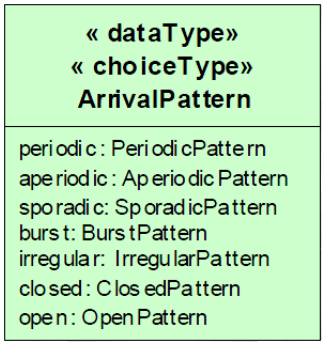

Figure 6.6: The Arrival Patterns available through the SAM implementation[21]

pattern as input. The GeArrival class, like the GeSchedObs class doesn't have any stereotype connected to it, just like the SchedulingObservation stereotype, the WordloadEvent stereotype isn't usable as a stereotype.

Both the Real time layer and the SAM layer of the basic example can be found in appendix B. At first glance it can seem rather complex, which wouldn't fit with the concept of wanting to make a simple modeling method, unfortunately the lack of functional inheritance in Papyrus and the need to stay with the MARTE layout means that the result can't be any simpler than what is seen. However because of the way the generic library, and MARTE, is designed, as the complexity of the system modeled increases, the complexity of the model doesn't increase by nearly as much, many of the elements defined in the basic example are 'overhead' elements, that are necessary regardless of whether one is modeling simple or complex systems. This is especially evident in the SAM layer, where the flow and behavior is only defined once pr flow, regardless of its complexity. A more complex flow would require more steps and relations but nothing else.
Another matter worth noting is that some of the elements can seem a bit redundant, the Arrival class contains little more than the arrival pattern, which one could be tempted to define as a property directly on the end-to-end flow and that way cut away a class. However the layout follows the MARTE layout which allows for several arrival patterns to be connected to one end-to-end flow, which is something that cant be modeled by a property, but requires an independent element to be understandable.
Finally when one looks at the SAM layer one has to keep in mind that this layer is only necessary when wanting to do schedulability analysis, in fact if one was to model a system purely for viewing from a design perspective it isn't recommendable to do use the SAM layer, instead one should stick to the Real time layer which contains all the high level information.

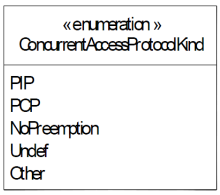

Figure 6.7: The ConcurrentAccessProtcolKind enumeration in MARTE, showing the available concurrent access protocols in the default implementation of MARTE[21]

6.2 Advanced Features

When converting the basic example from MAST to MARTE many of the MARTE objects used, contained the possibility for much more than the features used in the example, deadline types, arrival patterns, scheduling policies and the likes. This means that most of the advanced features are already available once the basic features have been implemented.

First the shared resource protection protocols available in MAST are looked into, as described in chapter 4 there are 3 such protocols available.

- Priority ceiling resource protocol
- Priority inheritance protocol
- Stack Resource Protocol

In MARTE the available protocols can be seen in the ConcurrentAccessProtocolKind enumeration shown in figure 6.7, and when comparing these it can be seen that MARTE contains the Priority ceiling resource protocol (PCP) and the Priority inheritance protocol (PIP). It doesn't however contain the Stack Resource protocol, and if this is to be available it needs to be added.

It isn't directly possible to add new elements to an enumeration defined in the MARTE profile, this leaves two possible ways to solve the situation, the first one would be defining a new enumeration in the generic library, containing all the elements from the concurrentAccessProtocolKind plus the Stack resource protocol. If the classes using the concurrentAccessProtcolKind was then changed to instead use this new enumeration, this would be a quick way to solve the problem, however this approach suffers from breaking with the MARTE profile, the classes defined in the Generic library are all stereotyping MARTE, and as

such is to be seen as extensions to the MARTE profile, and had it not been for the inheritance issues in Papyrus, a number of the classes wouldn't even have been necessary. Creating a new enumeration however would mean completely ignoring the MARTE enumeration, which also means that the generic library wouldn't be compatible should an updated version of MARTE be released which for instance included new protocols in the enumeration.
The alternative approach is to use the build in flexibility of MARTE, that results in all enumerations having an 'other' option that can be chosen, the MutualExclusionResource in MARTE has the property 'otherProtectionProtocol' in which a string can be entered containing the name of the protection protocol to be used which isn't available in the enumeration that MARTE supplies. This approach is a bit more complicated to use, because it requires the user to know that the protocol has to be defined this way, but at the same time it is the approach that stays closest to the MARTE profile. The work needed to implement the two approaches is very similar.
Since MARTE is still a young profile, chances are that there will be many updates and corrections to it still, therefore it seems prudent to stay compatible with any future releases, with a minimum of changes. This leads to the second approach being chosen and a new property is added to the SharedResource named 'otherProtectionProtocol' where users will be able to define any protocol they want to use, even beyond just the stack resource protocol.

The next feature to look into is the arrival patterns, in the basic example only periodic and sporadic arrival patterns were used, but it was found that MAST supports a number of other arrival patterns, as listed below.

- Singular
- Sporadic
- Unbounded
- Bursty

Again this is to be compared to the patterns available in MARTE, these have earlier been illustrated in figure 6.6. The Bursty pattern is directly available in MARTE in form of the 'burst' arrival pattern, the difference between unbounded and sporadic arrival patterns is that sporadic has an average and a minimum interarrival time where the unbounded only has an average interarrival time, meaning that no lower bound is defined for the unbounded arrival pattern. The sporadic arrival pattern can be modeled by the sporadic pattern in MARTE, and the unbounded pattern matches the aperiodic pattern. Both bursty and sporadic arrival patterns are extensions of the aperiodic pattern.
This leaves the singular, the MAST singular 'pattern' isn't actually a pattern,

it is an event that only happens once. While the developers of SAM describes the ArrivalPattern as "'This is a ChoiceType that contains the different kinds of parameters that are necessary to specify the most common arrival patterns of events."'[21] it seems they didn't find a singular event type to not fit within these limitations. Alternatively they singular arrival type might have been found too simple for requiring its one field among the arrival pattern, but there seems to be no way to input the information connected with a singular event type elsewhere (it is necessary to input when it is triggered).
It could be advantageous to have the Singular arrival type added to the list of possible arrival patterns. Unfortunately the arrivalpattern datatype doesn't contain an 'other' option, meaning that the solution used to add protection protocols to the system can't be used here. The only way to add a new element to the arrivalpattern datatype would be to duplicate the arrivalpattern from MARTE in a new datatype defined in the generic library. The consequences of such an approach, would be that the generic library would become detached from the MARTE profile. Should MARTE be updated with new arrival patterns the generic library wouldn't recognize this because it would have its own version. Creating duplicate versions of elements present in MARTE should only be done when it is necessary as a result of the original version not being implemented correctly in the papyrus implementation of the MARTE profile. Unfortunately there is no alternative way of adding the new arrival pattern to the framework that would make sense to the users. Instead if is unfortunately concluded that including the singular arrival pattern in the framework is too complex and should be left out. The best way to get it included would be to request it to be added to the official list of arrival patterns in the MARTE profile with its next update.

In the basic example section, it was determined that steps are somewhat comparable to operations, and while this is true for the information they supply, it is not so for the definition of flows, but because of the simplicity of the flows in the basic example the difference between the approaches to flow handling didn't become obvious. In MAST more complicated flows are defined through event handlers, but advanced flows can only be modeled for the interior runnings of a task, the flow of the tasks themselves is the responsibility of the scheduler and not the modeler. So while the transactions in MAST and the End-to-end flows in SAM are the elements responsible for defining the arrival patterns and the execution of the tasks, the internal flow of these elements are defined in MAST by the event handlers and in SAM by the steps. This means that the steps handle both the information of the operations, and the flow specifications of the event handlers. In figure 6.8 the modeling of a task and its internal flow can be seen as it is done in MAST, and in figure 6.9 it can be seen as it would be modeled in SAM.
While this might seem like a minor difference, it is still worth looking into. In SAM steps are connected directly to each other through the PrecedenceRela-

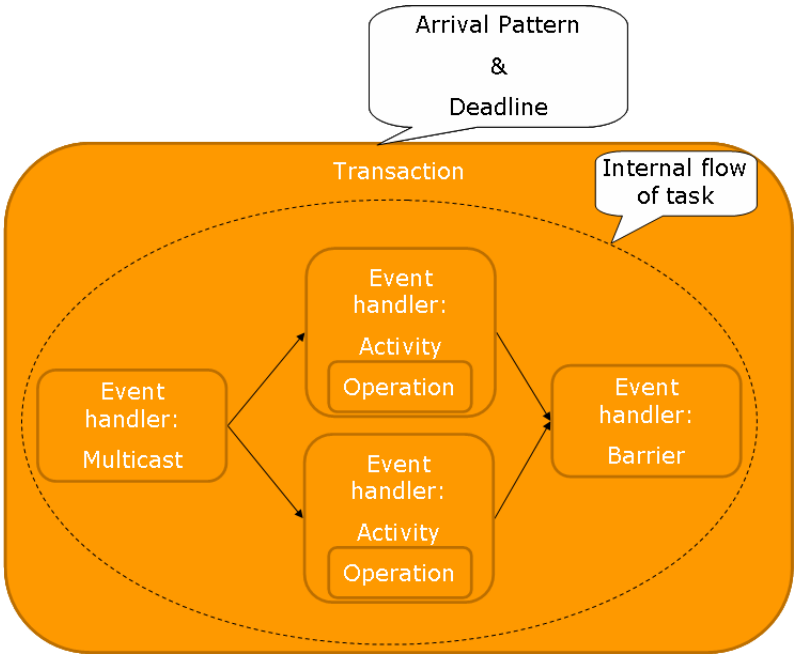

Figure 6.8: The internal flow of a task, a fork followed by a merge, as it is defined in MAST

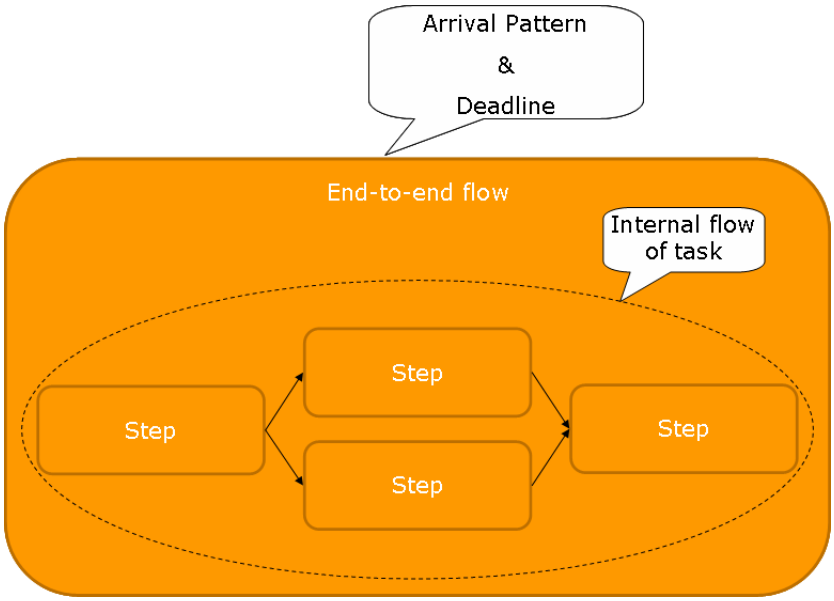

Figure 6.9: The internal flow of a task, a fork followed by a merge, as it is defined in SAM

tion, which directly determines the steps a given step is dependent on. This way for every step one can see which other steps has to finish before it can execute. While this is divided into two parts in MAST. This ties directly into the earlier mentioned fact that operations can be reused for many different tasks, because their location in the execution flow is determined through event handlers. An operation doesn't contain any specific information about when or where it executes or in combination with what, since it has already been determined that steps can't be reused for many reasons, having the information from the event handlers connected directly to the step seems like a fully acceptable approach, nothing would be gained from separating these elements in SAM.
All of this is directly relevant to the conversion of the advanced MAST features, as the event handlers define a number of advanced flow combinations, which should be compared to the SAM capabilities. The possibilities in SAM can be seen in figure 6.10, these different methods allows for the creation of more complex flows than what was seen in the basic example.

The sequence type is the flow pattern seen in the basic example, forks are when one step leads to several others, merges are the opposite, several step becoming one. Finally the branch connector is a connector type where, like the fork, one step leads to several others, but instead of activating all the successor steps, in a branch each successor step has a probability of being selected.

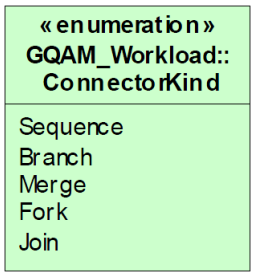

Figure 6.10: The different possibilities for defining flow types in SAM

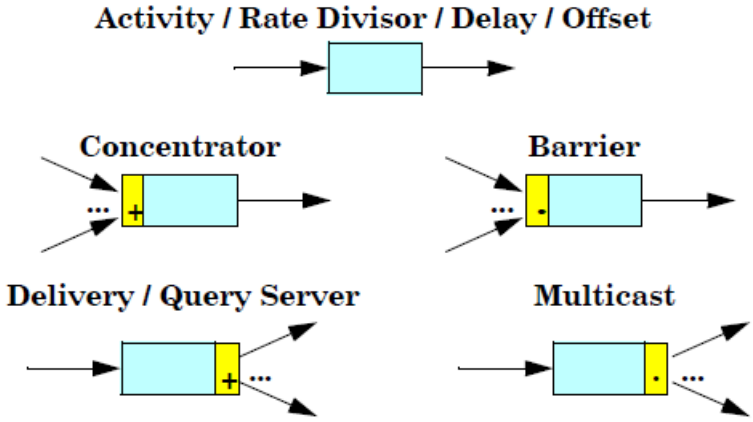

Figure 6.11: The event handlers available in MAST

In MAST the possible event handlers can be seen in figure 6.11. Some of these event handlers can be directly converted to one of the SAM connectors, the barrier matches the merge and the multicast matches the fork. While pattern wise the Activity/Rate divisor/Delay/Offset all matches the sequence connector, only the activity is a direct translation of it, the others all have advanced features connected to them, because the step contains both the information from the operations and the event handlers however, these advanced features shouldn't be handled through the connector types, but instead by the parameters defined in the steps.
The rate divisor doesn't generate an output event before a set number of input events has arrived.
The Delay waits after receiving an input with generation the output. This can be achieved through the selfDelay property defined in SaStep, which is extended

to be available in the generic library
The Offset like the Delay waits with generating its output, the difference is that the delay is started from some previously arrived input event, and not the newest one.
A concentrator generates its output event when it receives input on any of its input events, this matches a merge connector.

Both the delivery server and the query server are special cases of the branch connector, they only generate output on one of their output events whenever triggered, and then use different algorithms and parameters to determine which output event to trigger. Currently MARTE doesn't contain the possibility for defining how the probability of a branch connector looks, all that is known is that it is controlled by some probability. In order to use the Query and Delivery server which has more specific probability paths it must be possible to define the pattern being used.
The delivery server can use either a scan parameter in which the output event is chosen in a cyclic fashion, or it can be random, in which the delivery method is chosen completely randomly.
In the Query server the possibilities are a scan approach like the delivery server, highest priority, FIFO or LIFO.

Since the connectorKind enumeration isn't defined in the Papyrus implementation, and had to be defined in the Generic library, there isn't the same concern with expanding an enumeration as there was with the protection protocols on the shared resources, therefore the smartest approach to the many new eventhandlers is to add them to the connectorKind enumeration, and then add the necessary properties to the precedenceRelation for it to be possible to define the extra information some of those event handlers requires.

Finally as part of the advanced feature section, an element should be mentioned which isn't necessary for the MAST implementation, but is used to make the connection between MARTE models and ordinary system models in UML, this is the entry point association which is used to mark the entry point of a schedulable resource. The entry point association could be seen used in the OSEK/VDX example given in chapter 3 and can further be seen used in a number of small examples in the MARTE specification[21].
No changes are made to this association, but it still deserves mentioning since it is a vital part if one wants to use MARTE to model the real time layer of an existing standard UML system model, for models that illustrate pure real time systems, like the basic example looked upon in this thesis, the entrypoint association has no use, and neither does it from a schedulability analysis point of view, its sole purpose is to bind schedulable resources with their associated methods in an UML model.

Chapter 7

Connecting UML models with MARTE

In the MARTE analysis in chapter 3, a description was given explaining how MARTE is intended to be linked with an UML model of a software system by means of the entryPoint Stereotype.

This chapter is intended to further show how UML models can be linked with MARTE, by means of a practical example.
This wasn't done during the MARTE analysis because the elements needed for creating such an example hadn't yet been created.
Now that the generic library is created and tested however, all the material needed for linking an UML system model is available.

In the MARTE specification it is explained that the connection is done by making a dependency between the task and the method which has to be executed in the context of that schedulable resource.[21]

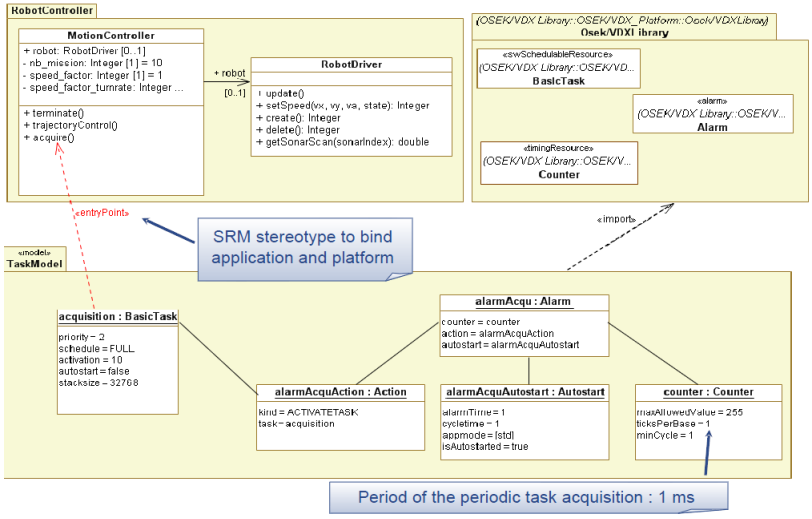

Figure 7.1: An Example modeling of a periodic task, using MARTE supported by the OSEK/VDX library[20]

7.1 MARTE Tutorial Example

In the MARTE tutorial there is an example model made in UML that shows how the connection between MARTE and a software system is created, figure 3.5 illustrated this, this figure is reposted in figure 7.1 to ease the reading.

Since the OSEK/VDX Library that is used in the example is available, and the example is created in Papyrus, duplicating it is possible. However when doing this one encounters a problem.
In figure 7.2 the example is tried implemented. Only the elements of direct relevance to the entryPoint concept is added. On this figure an instance of a BasicTask is created, and a class containing the operation 'acquire' is added.
Creating a dependency from the instance to the operation is no problem. However, when attempting to apply a stereotype to the dependency, Papyrus shows that no stereotypes can be applied. One must assume that if additional imports where done to the TaskModel to allow the entryPoint stereotype to be bound to the dependency it would be illustrated in the example.
If one adds the entire MARTE profile to both the RobotController package and the TaskModel model, a number of stereotypes becomes available that one can add to the dependency, not the entryPoint though. The same result is received if the MARTE profile is applied to the entire project.
So even when creating a copy of an official example, using the official OSEK/VDX

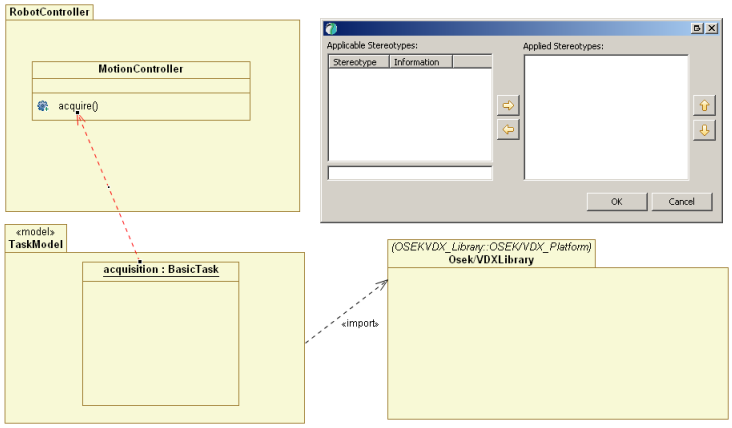

Figure 7.2: An attempt at duplicating the example from the MARTE tutorial, It is not possible to add the entryPoint stereotype to the dependency

library and MARTE profile, the results received in the example can't be reached. It is obviously possible, since it has been done by the creators of the MARTE tutorial[20]. How it was done, is however unknown, and attempts at contacting the people behind both the tutorial and the Papyrus project has been fruitless throughout this project.

7.2 Example linking of MARTE and a software model

The fact that the entryPoint stereotype isn't working, can be seen as a minor issue. It is obviously possible to use it, and even if not, it is an issue with Papyrus and not MARTE or the generic library. Therefore an example can be created. Inspired by the Basic Example which was implemented in chapter 6, an UML model is created of a fictive software system. This system contains two classes 'EmergencyHandler' and 'StatusUpdate' and can be seen in figure 7.3.

The EmergencyHandler class is a class which is triggered when an outside trigger sets the system in an emergency mode, when this happens the initiateEmergency method is called which shuts down all operations and forces a system reset.

The StatusUpdate class has a periodic trigger, when the trigger calls, the updateStatus method is started. UpdateStatus then connects to a communication

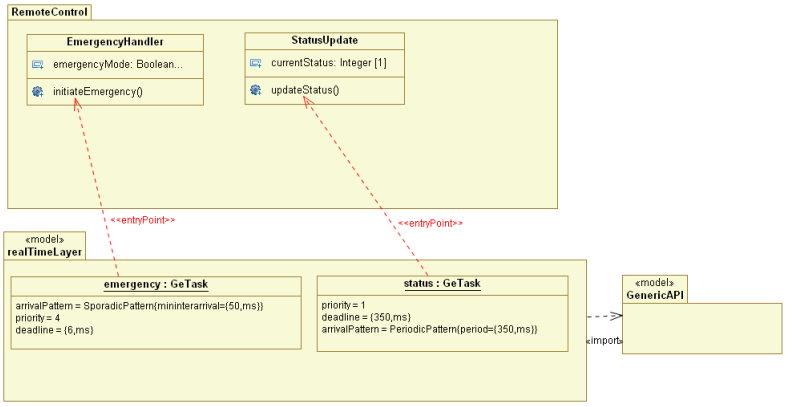

Figure 7.3: An example of how an UML model of a software system can be extended with a MARTE layer depicting its real time critical operations

server and receives the newest update to the status, and displays this. The status is defined as an integer and is seen as a flag, numbers representing different status states. Certain flag values can even trigger the systems emergency mode.

Now both of these methods are real time critical in our example. To handle this, the tasks emergency and status are created. Each of them is then bound to the operation they represent by means of a dependency, which would stereotype the entryPoint stereotype from the MARTE profile if this was possible, since it isn't currently, the dependencies are instead named '¡¡entryPoint¿¿ and their names are displayed, this gives a model that gives the correct picture of the situation.
With regards to the transformation from MARTE to MAST, the fact that the dependencies don't correctly stereotype the entryPoint is of no consequence, MAST only observes the MARTE layer of the model and as such ignores the UML system.

Chapter 8

MARTE to MAST

This chapter will look into the possibility of converting a MARTE model to a MAST input file, enabling the possibility of a schedulability analysis of a MARTE model directly.

While this transformation will be done manually in this chapter, the idea is for it to be an automated process achieved through software capable of doing the conversion. However creating such software has proved itself to be beyond the scope of this thesis and will have to be looked upon at a later stage.

While there has been found several similarities between MARTE and MAST and a number of the generic libraries elements are heavily inspired by MAST, the transformation isn't a straight over process, and a number of things has to be kept in mind when doing it.

In a papyrus project, any model is represented by two files, a .uml and a .di2 file.
The .uml file contains all the UML information, it defines what an object is called, what it inherits, what it stereotypes, its properties, parameters, etc.
The .di2 is a Diagram Interchange 2 standard file[13], containing the information necessary for creating the graphical representation of the .uml file. Both files are written in XML.

For the type of transformation that is to be done here, the graphics are of little interest, since the file format being aimed for isn't graphical. The MAST input format is defined in the mast description[3] though looking at the examples also available for MAST might be necessary for complete understanding.

Regardless of the graphical view, because models created by means of the generic library allows for the usage of associations to define connections between classes, it is necessary to include the di2 file in the transformation. The uml file only contains name and type of the associations, not the objects it links together, these link information are necessary to determine the instances bound by the associations however, and therefore needs to be known.

Since there isn't a requirement for names to be unique in UML, the files can't rely on names to uniquely identify objects, instead every element is assigned a xmi:id which can be used to identify it. This is also how the objects of the generic library are identified. This means that in order to recognize what object type an instance is, the xmi:id of its type has to be translated into an generic library element, this information can only be found in the .uml file connected to the implementation of the generic library.
Because the generic library uses a number of enumerations from the MARTE profile, in order to translate instance specifications that has a slot referencing an MARTE enumeration the MARTE library file is also necessary, though in a lesser capacity.

This all leads to four files being necessary to translate any given project from MARTE to MAST, the di2 and uml file of the model to translate, the uml file of the generic library and the MARTE library file. While one could be tempted to store the translation of the generic library locally and thereby spare the need for having it available every time a transformation is to be done, this solution isn't recommended since even a small change to the generic library would render the stored information obsolete. The MARTE library file is available in Papyrus, though one could make a copy of it for use if one remembers to update it when the MARTE profile is updated. Unlike the generic library, one would imagine that updates to the official MARTE profile happens less frequent, making it more acceptable to use a cached version.

8.1 Basic Example translation

While converting the entire basic example manually would serve little practical purpose this section will demonstrate how it is possible to translate a couple of elements, and highlight some of the pitfalls there is between MARTE and

MAST.
The Basic example can be seen modeled in Appendix B, while in an automated solution an initial approach of going through the uml file of the specific project and identifying all objects and determining their type before continuing with the transformation, since this is only a demonstration a jump is made directly to the GeScheduler object. Furthermore for the sake of readability instead of showing pure XML code, print outs from the XML files while have been through a XML-reader first, stripping away tags and leaving readable text, all information given can be found as XML code if one wishes.
In the GenericAPI.uml file, which contains the information about the generic library, an entry can be found concerning the GeScheduler:

xmi:type="uml:Class"
xmi:id="_Feh90HYhEd6zKowG9H-O7A"
name="GeScheduler"

This shows that the generic profile contains an element, which is an UML Class, its called GeScheduler and it has the ID _Feh90HYhEd6zKowG9H-O7A. It is this ID that should be used to identify GeScheduler elements in models.

In the BasicExample.uml file, which is the uml file connected to the basic example model it is now possible to find all the instances that are of the type GeScheduler. A search for the GeScheduler ID through the files yields the following information:

xmi:type="uml:InstanceSpecification"
xmi:id="_ar8_YHtnEd6XfJPJMTYLQQ"
name="Main Scheduler"

¡classifier
xmi:type="uml:Class"
href="../GenericAPI/GenericAPI.uml#_Feh90HYhEd6zKowG9H-O7A"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
/¿

An object with the name Main Scheduler was found, it has the ID _ar8_YHtnEd6XfJPJMTYLQQ and is an instanceSpecification, and this object has is classified as being an element of the type "'/GenericAPI/GenericAPI.uml#_Feh90HYhEd6zKowG9H-O7A"' which is the ID determined for the GeScheduler.
Now the instantiation of the scheduler has been found, the information attached to it can then be determined, in this case there are 2 slots attached to the Main scheduler, which defines the properties and parameters defined on the instance:

slot

xmi:id="_w5n0EHtqEd6XfJPJMTYLQQ"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"

definingFeature
xmi:type="uml:Property"
href="../GenericAPI/GenericAPI.uml#_WoJKIHYhEd6zKowG9H-O7A"

value
xmi:type="uml:InstanceValue"
xmi:id="_zsvpsHtvEd6eEq_gtEeQQA"
instance="_gVdlkHtnEd6XfJPJMTYLQQ"

slot
xmi:id="_zEfwMHtqEd6XfJPJMTYLQQ"

definingFeature
xmi:type="uml:Property"
href="../GenericAPI/GenericAPI.uml#_W4_CgHYhEd6zKowG9H-O7A"

value
xmi:type="uml:InstanceValue"
xmi:id="_1PYTYHtqEd6XfJPJMTYLQQ"

instance
xmi:type="uml:EnumerationLiteral"
href="pathmap://Papyrus_PROFILES/MARTE_Library.library.uml#_6VPi8BFaEdyUJeMeN__D-A"

Each slot has a definingFeature and a value attached to it, the definingFeature tells the name of the of property being defined, and value tells about the parameter entered into the system.
It can be seen that the names of the properties are also to be found in the GenericAPI.uml file, the value of the first slot is another instance specification, and as such can be found in the BasicExample.uml file, finally the second slot has an instance attached to it, this is because it contains an instanceValue as its value, in this case this is an EnumerationLiteral which is to be found in the MARTE profile. The same way the ID of the GeScheduler was found, the name of the instance value and the name of the different properties can be found, through simple lookups in the appropriate files.

Now the scheduler has been identified and the defined properties and values has been determined, the remaining information about the scheduler is the associations it has determining the tasks and shared resources connected to it. MAST, unlike MARTE, doesn't attach this information to the scheduler, and

for the point of translating those informations aren't necessary. A last bit of information that it is possible to define in MAST, without knowing whether it is necessary or not since the information is also defined on the processing resource, is the context switch time. In the generic library this information is attached to the GeExecHost, and through lookups an instance of this can be found and the context switch time can be determined to be 0.25 ms. Now enough information has been gathered to write the scheduler in a way that lives up to the MAST input file specifications:

Scheduler (Type =¿ Primary_Scheduler, Name =¿ Main Scheduler, Policy =¿ (Type =¿ Fixed_Priority, Worst_Context_Switch =¿ 0.25);, Host =¿ CPU);

This is a rather complex translation and the result of it was that one element has been written in MAST, a simple method even. But it proves that all the information necessary is available, and the example shows how to interact with the different files supplied by MARTE. The entire process could be made trivial through the development of a software translator however. While the process might seem complicated, there is a pattern to it which is identifiable and automatable.

When doing the transformation from MARTE to MAST, because of the lack of re-use of steps in MARTE, it will often be found that the resulting file could have been written simpler in pure MAST, but the resulting system being represented will always be the same. Should this lack of simplicity result in a noticeable decrease in performance, optimization can be done on the operations when these are created from the steps to enable re-use and the re-introduction of enclosing operation types.

Chapter 9

Conclusion

In this thesis, several different areas has been examined, and a number of conclusion has been reached. This is reflected in this chapter through a number of sections each concluding on a specific area of the thesis. The first areas to be covered in this chapter are the goals of the thesis.

The primary goal of this thesis was to determine how MARTE functions, and whether it is possible to use MARTE in an educational environment to introduce users to the field of real time and embedded modeling through UML. MARTE as a system will be the first subject discussed in this chapter.
This will be followed by a section focused on the usability of MARTE, considering both intended users and possible users.
The next goal was to enable a transition from MARTE model to schedulability analysis in MAST.

The conclusions concerning the thesis goals will be followed by a section which looks at the pros and cons of using Papyrus.

Finally an overall evaluation of the thesis as a whole is done.

9.1 The MARTE profile

The MARTE profile has been thoroughly examined in this thesis. The idea behind the MARTE profile was to create an UML extension that would encompass all real time and embedded modeling needs, and the MARTE profile seems to cover all fields and areas relevant for real time and embedded modeling, this is however an assumption based on the experience gathered during the thesis work, to guarantee that the MARTE profile covers all fields sufficiently, would require consultation with experts from all of these fields. Regardless the target audience of MARTE leaves much to be desired. MARTE was designed by professional real time modelers and in its current form it is usable only by professional.

The steps taken in this thesis might help address this, but until an official support library is released to address the potential users not working within a specific industry field of real time modeling the issue will remain.

Even though the MARTE profile is still in beta, the lack of documentation is an issue that should be addressed as soon as possible. Expecting anyone to work with a profile that requires weeks, if not months of studying to understand and use is unrealistic. The tutorial available for MARTE is a slideshow presentation which lacks the words intended to go with it. Getting support from the developers is also a difficult task, several attempts at contacting where ignored, and it wasn't until a request was made by an official DTU professor that any response was received.

While the current form of the MARTE profile and its documentation might be sufficient for people working within the specific real time fields, it isn't ready for general availability yet.

Despite the lack of documentation and available support, the MARTE profile is a thorough profile which, should the issues found be corrected, could become a very useful tool for real time and embedded modeling.

9.2 Using MARTE for educational purposes

As it was discussed, MARTE suffers from a lack of user friendliness. This isn't the only concern that was considered when evaluating MARTE as a tool to be used for introducing new users to real time and embedded modeling. The initial impression of MARTE, when going over the documentation available, is that it is a very complex framework that introduces a number of new terms to ensure

that they are unique. From an educational point of view the interest is to use as general terms as possible so they are valid in as many fields as possible. However when going deeper into the structure of the MARTE profile it is found that it is mainly a number of key elements that has been named in unconventional ways, but because they are the main elements of any real time model they still become the focus, the SwSchedulableResource is the prime example of this.
The concept of support libraries that the MARTE profiles relies on allows these deviations from the general terms to be overcome, and as can be seen from the generic library defined in this thesis, it is possible to create a MARTE driven framework that is easy to understand and use.

9.3 From MARTE to MAST

The idea of having a real time model, created in UML, and be able to get it analyzed for schedulability was another concept that this thesis considered. The MAST analyzing tool that was chosen uses the same overall structure as the MARTE, hierarchical schedulers with attached tasks, task execution flow, etc. When MARTE is used through the generic library, the modeling allowed by the framework available is of such a kind that it can be fully translated, from MARTE model to MAST input. Because of the way the Papyrus tool represents elements and especially associations, doing the translation by hand isn't recommended. However creating software that can do the translation effectively is doable, this was however, beyond the time scope of this thesis.

The generic library, supported by a software translator would make it possible to do schedulability analysis directly from a MARTE UML model of a real time system, but would this be better, smarter or more efficient than entering the model directly into MAST?
The disadvantages would be that the generic library requires more elements to be defined than is needed by the MAST, and it uses two layers instead of only a schedulability analysis layer. Because a model created in the MARTE framework represents more than just the schedulability analysis, creating will require more work than modeling directly in the MAST framework.

The advantages of modeling a system in the MARTE framework would be that one has a system that is within the boundaries of a recognized real time modeling framework, and the model can extend UML models of software systems in a non invasive way. Furthermore it is not a requirement of the generic library that the schedulability analysis tool used is MAST. While MAST has been the tool used in this thesis, any tool can in theory be used, and with a minimum of changes to the generic library. A software translator would have to be written

for each analysis tool to be used, but this is something that only has to be done the first time a tool is used.

There are scenarios where modeling directly into MAST is the most efficient way to go. If it is a one time event that schedulability analysis is required, it would be simpler to just use MAST, but in situations where the model was to be manipulated with in iterations, or where a link has to be created between the system represented by the real time model, using the generic library and MARTE would be very effective.

9.4 The Papyrus Modeling tool

Once the study of the MARTE profile was completed, and the functionality and concepts understood, most of the problems with developing a suitable solution can be attributed to the Papyrus Modeling tool.
The Papyrus tool was chosen for this thesis, as a result of it being the be most commonly used in combination with the MARTE profile, it is the only tool to which support libraries has been developed and made available. Furthermore since it builds on the eclipse platform, it was assumed that using it would be familiar to anyone with experience using eclipse. Unfortunately a number of issues was encountered while modeling with the tool.

The implementation of the MARTE profile is missing objects and contains misleading typos, it bears the marks of an untested environment. The Papyrus tool itself is documented, but the documentation is based on 'first use' situations of Papyrus, and little support is available for more advanced troubleshooting. Attempts at contacting the Papyrus developing team, both with regards to possible bugs in the tool, and for general support has gone unanswered.

This caused the generic library to contain a number of 'fixes' that doesn't fit exactly with the optimal solution, but was necessary to create a working framework. Many of these could be avoided if the Papyrus tool is updated to fix the bugs it contains.

9.5 Thesis Conclusion

Overall this thesis paper defines a number of goals, and reaches them. The generic library created became more complicated than what was initially hoped

for, this was partly a result of having to redefine a number of elements from the MARTE framework as a result of the inheritance issues encountered, but it is functional.
The complexity of the generic library is on a level, that should be understandable for anyone with basic knowledge within the fields of UML modeling and real time and embedded systems. A bit of studying might be necessary to fully understand the possibilities of modeling with the generic library, but this should be a minor amount, comparable to the time it would take a user to adapt to any new tool.

The models created with the generic library can contain all the information required for doing a schedulability analysis, and translating a model into the input format of any analysis tool is doable.

Bibliography

[1] Donald Bell. Uml's sequence diagram. http://www.ibm.com/developerworks/rational/library/3101.html, 2004.

[2] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The UML User Guide*. Addison Wesley, 1998.

[3] José Drake and et al. *Description of the MAST Model*, 2008.

[4] Eclipse. *Eclipse Documentation*, 2007.

[5] Sébastien Gérard. Open source tool for graphical uml 2 modelling. http://www.papyrusuml.org, 2009.

[6] Inc No Magic. Magicdraw - architecture made simple. http://www.magicdraw.com, 2009.

[7] University of Cantabria. *MAST Status File*, 2008.

[8] University of Cantabria. Modeling and analysis suite for real-time applications. http://mast.unican.es/mast-getting-started.html, 2008.

[9] University of Cantabria. Visual modeling and analysis suite for real-time applications with uml. http://mast.unican.es/umlmast/, 2008.

[10] University of Cantabria. Mast. http://mast.unican.es/, 2009.

[11] OMG. Uml for systems engineering rfp. http://syseng.omg.org/UML_for_SE_RFP.htm, 2003.

[12] OMG. *UML Profile for Schedulability,Performance, and Time Specification*, 2005.

[13] OMG. *Diagram Interchange*, 2006.

[14] OMG. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms Specification*, 2008.

[15] OMG. Marte. http://omgmarte.org/, 2009.

[16] OMG. The object management group (omg). http://omg.org/, 2009.

[17] OMG. Omg systems modeling language. http://www.omgsysml.org/, 2009.

[18] OMG. Unified modeling language. http://uml.org, 2009.

[19] RTaW. Realtime-at-work - better technical solutions for real-time and embedded systems. http://www.realtimeatwork.com, 2009.

[20] Thales. *MARTE Tutorial*, 2007.

[21] THALES. *A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2*, 2008.

[22] unkown. *ARINC653 library*, 2009.

[23] Maria C. Valiente, Gonzalo Genova, and Jesus Carrtero. Uml 2.0 notation for modeling real time task scheduling. *Journal of Object Technology*, 5(4):91–105, 2005.

Appendix A

Generic Library Implementation

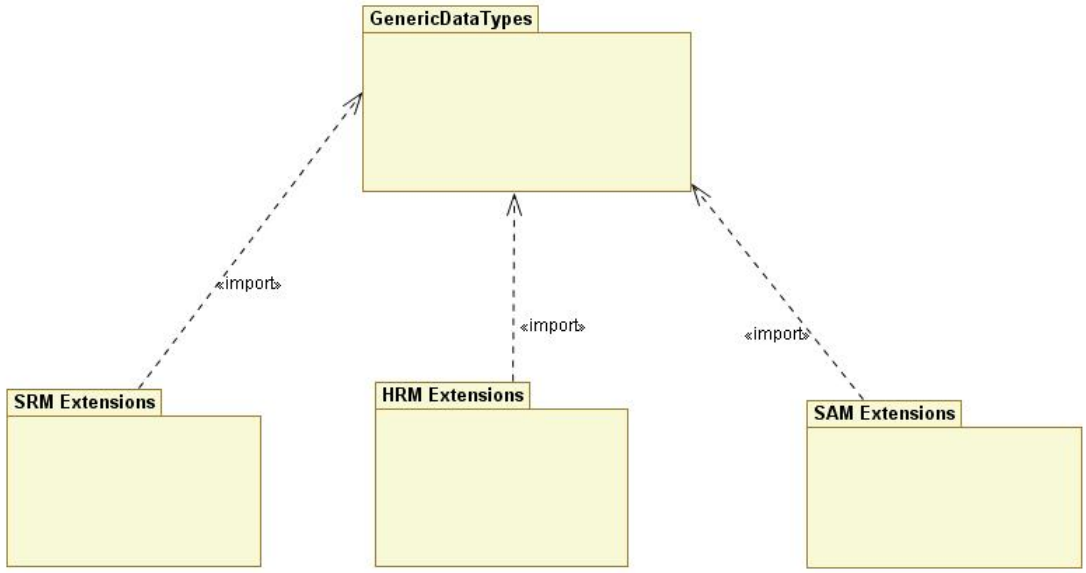

Figure A.1: The basic structure of the generic library, 3 extension packages each representing their branch all connected to a common data type package containing the enumerations, datatypes, and similar available.

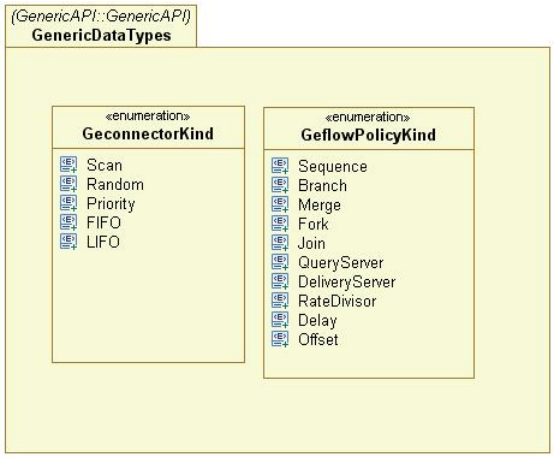

Figure A.2: The datatype package, containing the datatypes, enumerations, and similar available needed by elements in the generic library.

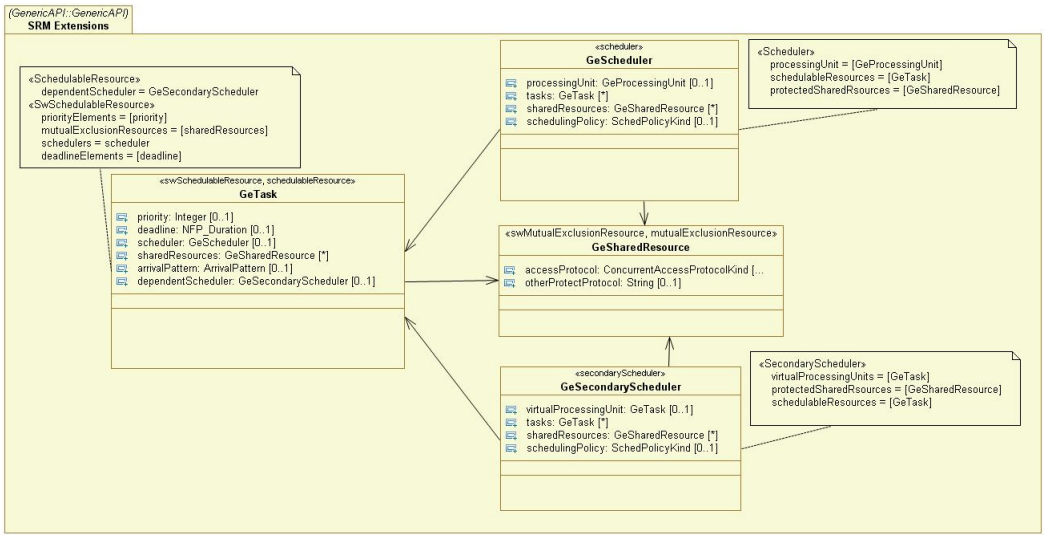

Figure A.3: The SRM extension package of the generic library, containing the elements that are extensions of the MARTE SRM package.

Figure A.4: The HRM extension package of the generic library, this package is the smallest, currently only containing the processing resource.

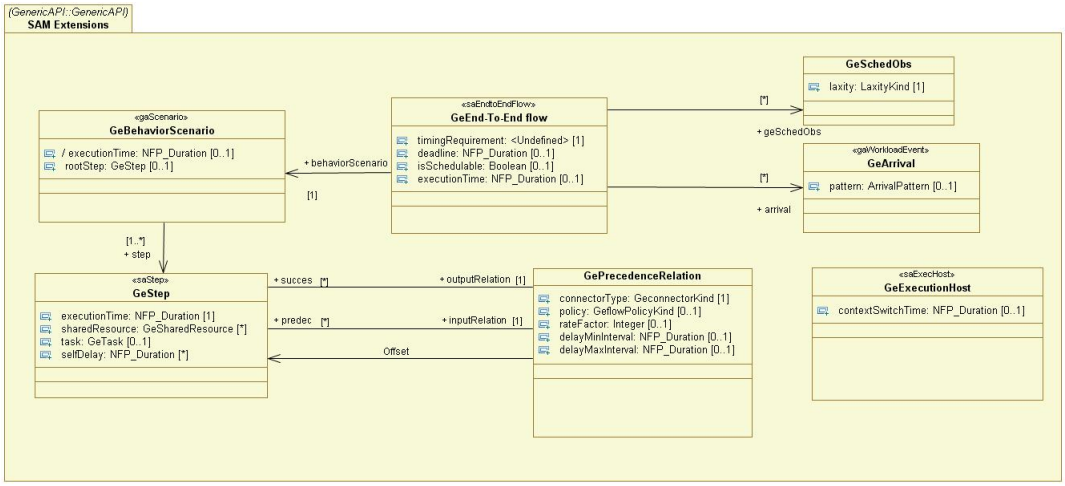

Figure A.5: The SAM extension package of the generic library, here all extensions of the SAM module can be found, this package includes classes that doesn't stereotype a MARTE stereotype because of the stereotypes either not being implemented in the Papyrus model of MARTE or being implemented in a matter that doesn't extend the metaclass concept

Appendix B

Basic Example Implementation

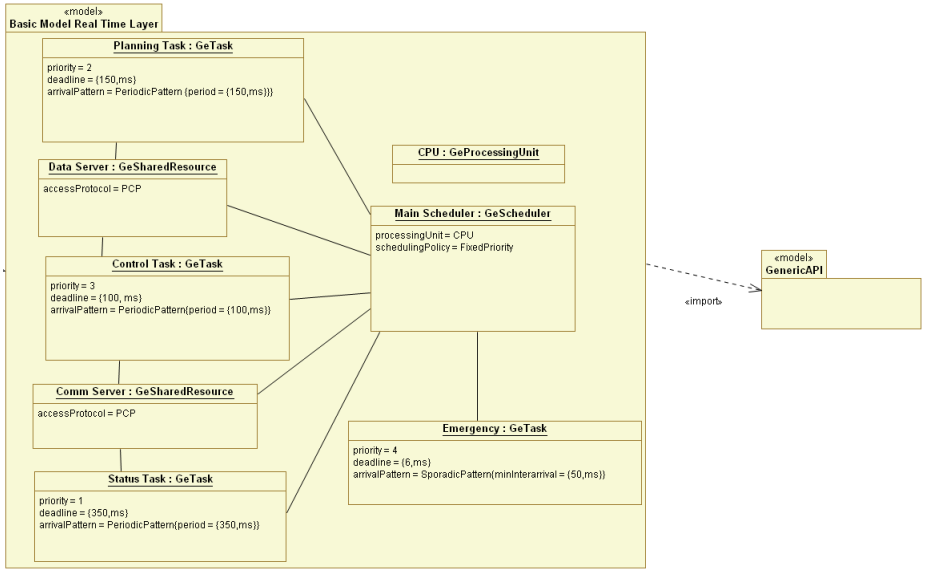

Figure B.1: First half of the basic example, the real time layer of the model, this layer shows all the key elements of the system and its key values

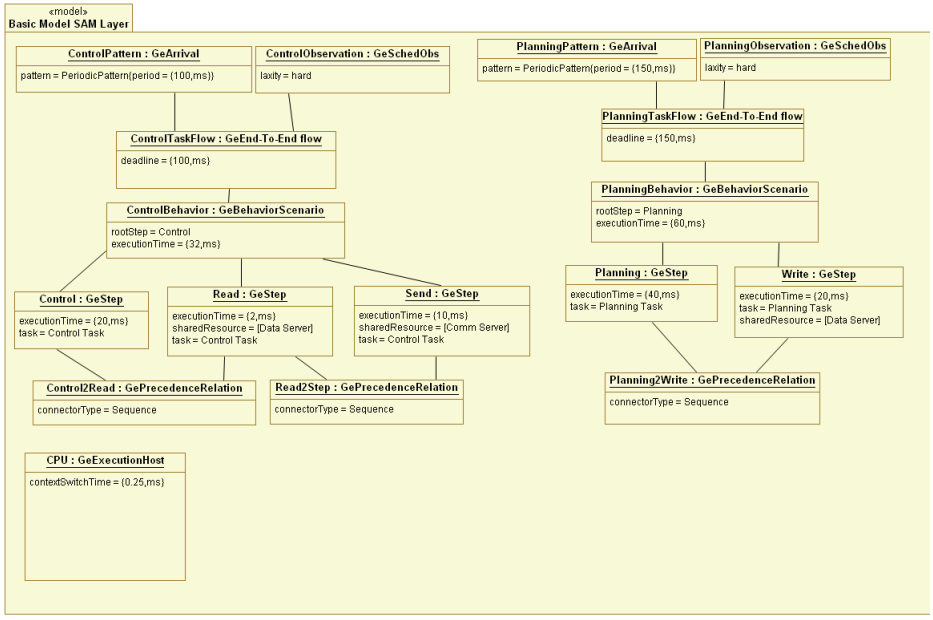

Figure B.2: The second part of the basic example is split into two for easier viewing, part one showing the control task and planning parts and their associated event flows

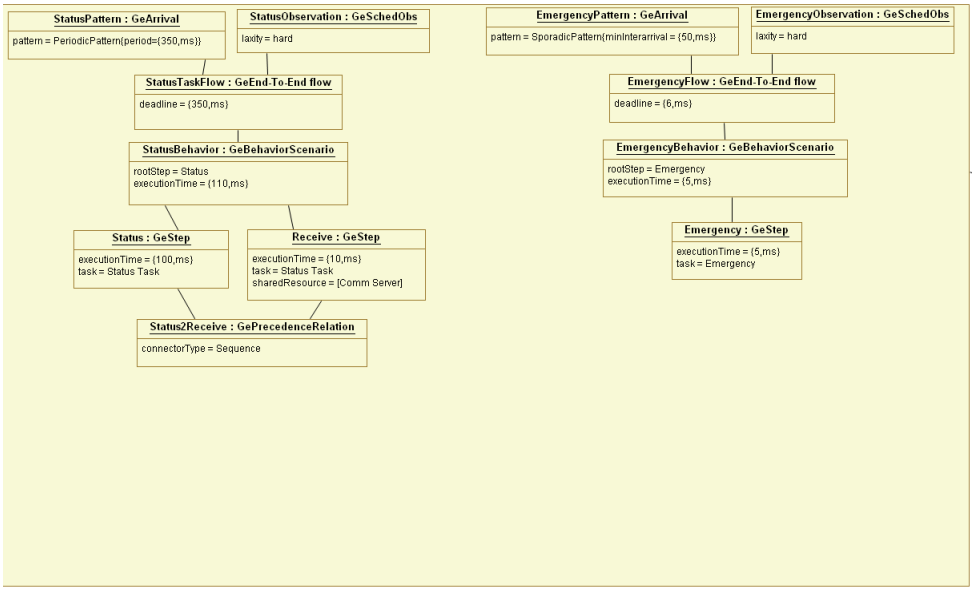

Figure B.3: Second half of the second pat of the basic example, part two shows the status task and the emergency task and their associated event flow

Appendix C

Papyrus Generalization bug

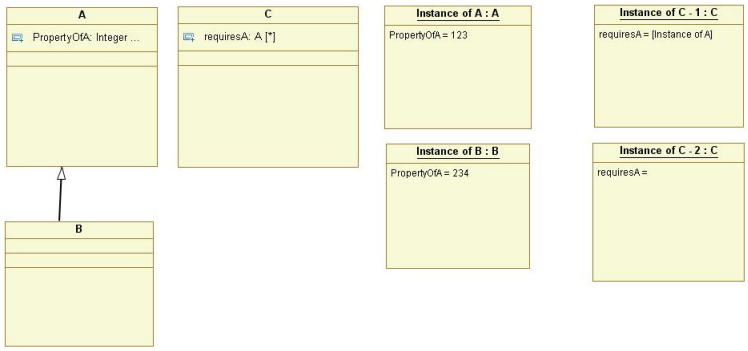

Figure C.1: An example project in Papyrus demonstrating the generalization bug. 3 classes are defined A, B, and C, B inherits from A. 4 Instance specifications are then defined, one representing the A and B class respectively and 2 instantiations of the C class, the first one having the A class defined as its parameter, the second should have the B class in its parameter, but this fails in Papyrus