We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

10

A Methodology for Scheduling Analysis Based on UML Development Models

Matthias Hagner and Ursula Goltz
Institute for Programming and Reactive Systems
TU Braunschweig
Germany

1. Introduction

The complexity of embedded systems and their safety requirements have risen significantly in the last years. The model based development approach helps to handle the complexity. However, the support for analysis of non-functional properties based on development models, and consequently the integration of these analyses in a development process exist only sporadically, in particular concerning scheduling analysis. There is no methodology that covers all aspects of doing a scheduling analysis, including process steps concerning the questions, how to add necessary parameters to the UML model, how to separate between experimental decisions and design decisions, or how to handle different variants of a system. In this chapter, we describe a methodology that covers these aspects for an integration of scheduling analyses into a UML based development process. The methodology describes process steps that define how to create a UML model containing the timing aspects, how to parameterise it (e.g., by using external specialised tools), how to do an analysis, how to handle different variants of a model, and how to carry design decision based on analysis results over to the design model. The methodology specifies guidelines on how to integrate a scheduling analysis for systems using static priority scheduling policies in a development process. We present this methodology on a case study on a robotic control system.

To handle the complexity and fulfil the sometimes safety critical requirements, the model based development approach has been widely appreciated. The UML (Object Management Group (2003)) has been established as one of the most popular modelling languages. Using extension, e.g., SysML (Object Management Group (2007)), or UML profiles, e.g., MARTE (Modelling and Analysis of Real-Time and Embedded Systems) (Object Management Group (2009)), UML can be better adapted to the needs of embedded systems, e.g., the non functional requirement scheduling. Especially MARTE contains a large number of possibilities to add timing and scheduling aspects to a UML model. However, because of the size and complexity of the profile it is hard for common developers to handle it. Hence, it requires guidance in terms of a methodology for a successful application of the MARTE profile.

Besides specification and tracing of timing requirements through different design stages, the major goal of enriching models with timing information is to enable early validation and verification of design decisions. As designs for an embedded or safety critical systems may have to be discarded if deadlines are missed or resources are overloaded, early timing analysis has become an issue and is supported by a number of specialised analysis tools, e.g., SymTA/S (Henia et al. (2005)), MAST (Harbour et al. (2001)), and TIMES (Fersman & Yi

(2004)). However, the meta models used by these tools differ from each other and in particular from UML models used for design. Thus, to make an analysis possible and to integrate it into a development process, the developer has to remodel the system in the analysis tool. This leads to more work and possibly errors made by the remodelling. Additionally, the developer has to learn how to use the chosen analysis tool. To avoid this major effort, an automatic model transformation is needed to build an interface that enables automated analysis of a MARTE extended UML model using existing real-time analysis technology.

There has been some work done developing support for the application of the MARTE profile or to enable scheduling analysis based on UML models. The Scheduling Analysis View (SAV) (Hagner & Huhn (2007), Hagner & Huhn (2008)) is one example for guidelines to handle the complexity of the UML and the MARTE profile. A transformation from the SAV to an analysis tool SymTA/S is already realised (Hagner & Goltz (2010)). Additional tool support was created (Hagner & Huhn (2008)) to help the developer to adapt to guidelines of the SAV. Espinoza et al. (2008) described how to use design decisions based on analysis results and showed the limitations of the UML concerning these aspects. There are also methodical steps identified, how the developer can make such a design decision. However, there are still important steps missing to integrate the scheduling analysis into a UML based development process. In Hagner et al. (2008), we observed the possibilities MARTE offers for the development in the rail automation domain. However, no concrete methodology is described. In this chapter, we want to address open questions like: Where do the scheduling parameters come from (e.g., priorities, execution patterns, execution times), considering the development stages (early development stage: estimated values or measured values from components-off-the-shelf, later development stages: parameters from specialised tools, e.g., aiT (Ferdinand et al. (2001))? How to bring back design decision based on scheduling analysis results into a design model? How to handle different criticality levels or different variants of the same system (e.g., by using different task distributions on the hardware resources)? In this chapter, we want to present a methodology to integrate the scheduling analysis into a UML based development process for embedded real-time systems by covering these aspects. All implementations presented in this chapter are realised for the case tool Papyrus for UML[1].

This chapter is structured as follows: Section 2 describes our methodology, Section 3 gives a case study of a robotic control system on which we applied our methodology, Section 4 shows how this approach could be adopted to other non-functional properties, and Section 5 concludes the chapter.

2. A methodology for the integration of scheduling analysis into a UML based development process

The integration of scheduling analysis demands specified methodologies, because the UML based development models cannot be used as an input for analysis tools. One reason is that these tools use their own input format/meta model, which is not compatible with UML. Another reason is that there is important scheduling information missing in the development model. UML profiles and model transformation help to bridge the gap between development models and analysis tools. However, these tools have to be adapted well to the needs of the development. Moreover, the developer needs guidelines to do an analysis as this cannot be fully automated.

[1] http://www.papyrusuml.org

Figure 1 depicts our methodology for integrating the scheduling analysis into a UML based development process. On the left side, the Design Model is the starting point of our methodology. It contains the common system description by using UML and SysML diagrams. We assume that it is already part of the development process before we add our methodology. Everything else depicted in Figure 1 describes the methodology.

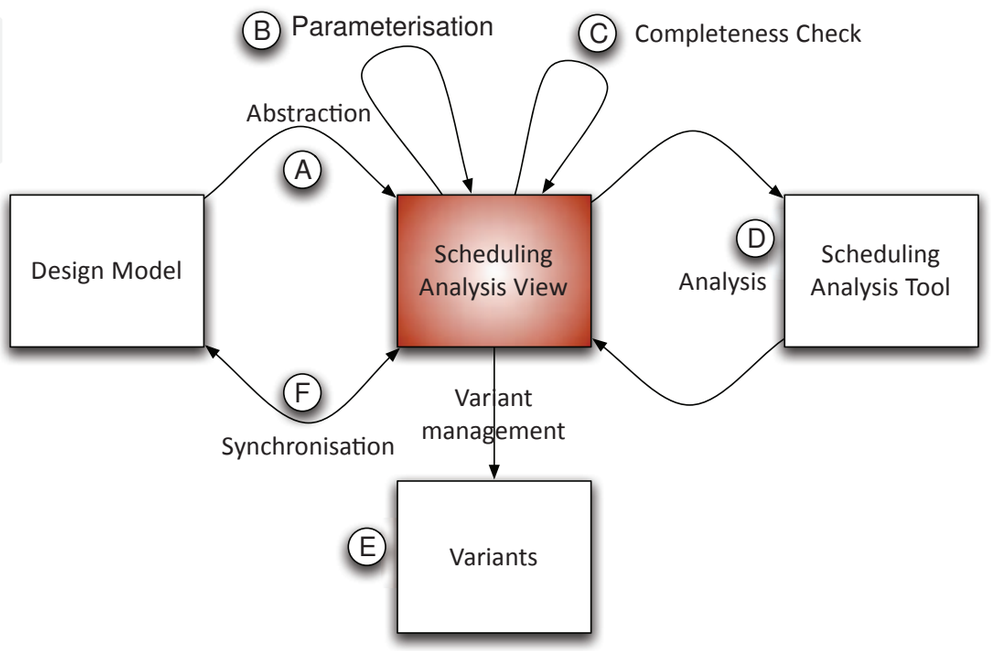


Fig. 1. Methodology for the integration of scheduling analysis in a UML based development process

The centre of the methodology is the Scheduling Analysis View (SAV). It is a special view on the system under a scheduling analysis perspective. It leaves out not relevant information for a scheduling analysis, but offers possibilities to add important scheduling information that are usually difficult to specify in a common UML model and are often left out of the normal Design Model. The SAV consists of UML diagrams and MARTE elements. It is an intermediate step between the Design Model and the scheduling analysis tools. The rest of the methodology is based on the SAV. It connects the different views and the external analysis tools. It consists of:

- an abstraction, to create a SAV based on the Design Model using as much information from the Design Model as possible,
- a parameterisation, to add the missing information relevant for the analysis (e.g., priorities, execution times),
- a completeness check, to make sure the SAV is properly defined,
- the analysis, to perform the scheduling analysis,
- variant management, to handle different variants of the same system (e.g., using different distribution, other priorities), and
- a synchronisation, to keep the consistency between the Design Model and the SAV.

The developer does not need to see or learn how to use the analysis tools, as a scheduling analysis can be performed automatically from the SAV as an input.

The following subsections describe these steps in more detail. Figure 1 gives an order in which the steps should be executed (using the letters A, B, . . .). A (the abstraction) is performed only once and F (the synchronisation) only if required. Concerning the other steps, B, C, D, E can be executed repeatedly until the developer is satisfied. Then, F can be performed.

2.1 The scheduling analysis view

Independent, non-functional properties should be handled separately to allow the developer to concentrate on the particular aspect he/she is working on and masking those parts of a model that do not contribute to it. This is drawn upon the cognitive load theory (Sweller (2003)), which states that human cognitive productivity dramatically decreases when more different dimensions have to be considered at the same time. As a consequence in software engineering a number of clearly differentiated views for architecture and design have been proposed (Kruchten (1995)).

As a centre of this methodology, we use the Scheduling Analysis View (SAV) (Hagner & Huhn (2008)) as a special view on the system. The SAV is based on UML diagrams and the MARTE profile (stereotypes and tagged values). MARTE is proposed by the "ProMarte" consortium with the goal of extending UML modelling facilities with concepts needed for real-time embedded systems design like timing, resource allocation, and other non-functional runtime properties. The MARTE profile is a successor of the profile for Schedulability, Performance, and Time (SPT profile) (Object Management Group (2002)) and the profile for Modelling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS profile) (Object Management Group (2004)).

The profile consists of three main packages. The MARTE Foundations package defines the basic concepts to design and analyse an embedded, real-time system. The MARTE Design Model offers elements for requirements capturing, the specification, the design, and the implementation phase. Therefore, it provides a concept for high-level modelling and a concept for detailed hard- and software description. The MARTE Analysis Model defines specific model abstractions and annotations that could be used by external tools to analyse the described system. Thus, the analysis package is divided into three parts, according to the kind of analysis. The first part defines a general concept for quantitative analysis techniques; the second and third parts are focused on schedulability and performance analysis.

Because runtime properties and in particular timing are important in each development phase, the MARTE profile is applicable during the development process, e.g., to define and refine requirements, to model the partitioning of software and hardware in detail, or to prepare and complete UML models for transformation to automated scheduling or performance analysis. One application of the MARTE profile is shown in Figure 2. MARTE is widespread in the field of developing of embedded systems (e.g., Argyris et al. (2010); Arpinen et al. (2011); Faugere et al. (2007)).

We only use a small amount of the stereotypes and tagged values for the SAV, as the MARTE profile offers much more applications. One goal of the SAV is to keep it as simple as possible. Therefore, only elements are used that are necessary to describe all the information that is needed for an analysis. In Table 1 all used stereotypes and tagged values are presented. Additionally, we offer guidelines and rules, how to define certain aspects of the systems in the SAV. The SAV was designed regarding the information required by a number of scheduling

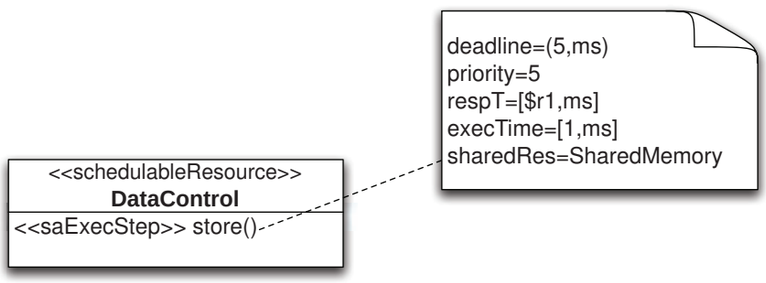


Fig. 2. Example of a UML profile

analysis tools. It concentrates on and highlights timing and scheduling aspects. It is based on the Design Model, but abstracts/leaves out all information that is not needed for a scheduling analysis (e.g., data structure). On the other side, it includes elements that are usually not part of the Design Model, but necessary for scheduling analysis (e.g., priorities, deadlines, scheduling algorithms, execution times of tasks).

Stereotype	used on	Tagged Values
«saExecHost»	Classes, Objects	Utilization, mainScheduler, isSched
«saCommHost»	Classes, Objects	Utilization, mainScheduler, isSched
«scheduler»	Classes, Objects	schedPolicy, otherSchedPolicy
«schedulableResource»	Classes, Objects	
«saSharedResources»	Classes, Objects	
«saExecStep»	Methods	deadline, priority, execTime, usedResource, respT
«saCommStep»	Methods	deadline, priority, execTime, msgSize, respT
«saEndToEndFlow»	Activities	end2endT, end2endD, isSched
«gaWorkloadEvent»	Initial-Node	pattern
«allocated»	Associations	

Table 1. The MARTE stereotypes and tagged values used for the SAV

Another advantage of the SAV is the fact, that it is separate from the normal Design Model. Besides the possibility to focus just on scheduling, it also gives the developer the possibility to test variants/design decisions in the SAV without changing anything in the Design Model. As there is no automatic and instant synchronisation (see Section 2.6), it does not automatically change the Design Model if the developer wants to experiment or e.g., has to add provisional priorities to the system to analyse it, although at an early stage these priorities are not a design decision.

Moreover, an advantage of using the SAV is that the tagged values help the developer to keep track of timing requirements during the development, as these parameters are part of the development model. This especially helps to keep considering them during refinement.

Class diagrams are used to describe the architectural view/the structure of the modelled system. The diagrams show resources, tasks, and associations between these elements. Furthermore, schedulers and other resources, like shared memory, can be defined. Figure 3 shows a class diagram of the SAV that describes the architecture of a sample system. The functionalities/the tasks and communication tasks are represented by methods. The tasks are described using the «saExecStep» stereotype. The methods that represent the communication tasks (transmitting of data over a bus) are extended with the «saCommStep» stereotype. The tasks or communication tasks, represented as methods, are part of schedulable resource classes (marked with the «schedulabeResource» stereotype), which combine tasks or communications that belong together, e.g., since they are part of the same use case or all of them are service routines. Processor resources are represented as classes with the «saExecHost» stereotype and bus resources are classes with the «saCommHost» stereotype. The tasks and communications are mapped on processors or busses by using associations between the schedulable resources and the corresponding bus or processor resource. The associations are extended with the «allocated» stereotype. Scheduling relevant parameters (deadlines, execution times, priorities, etc.) are added to the model using tagged values (see an example in Figure 2).

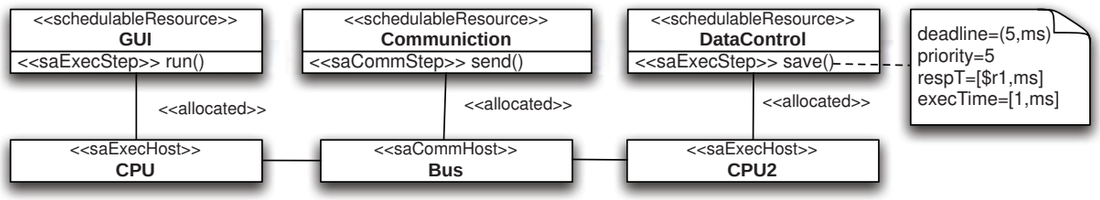


Fig. 3. Architectural Part of the SAV

The object diagram or runtime view is based on the class diagram/architectural view of the SAV. It defines how many instances are parts of the runtime system respectively and what parts are considered for the scheduling analysis. It is possible that only some elements defined in the class diagram are instantiated. Furthermore, some elements can be instantiated twice or more (e.g., if elements are redundant). Only instantiated objects will later be taken into account for the scheduling analysis.

Activity diagrams are used to describe the behaviour of the system. Therefore, workload situations are defined that outline the flow of tasks that are executed during a certain mode of the system. The dependencies of tasks and the execution order are illustrated. The «gaWorkloadEvent» and the «saEnd2EndFlow» stereotypes and their corresponding tagged values are used to describe the workload behaviour parameters like the arrival pattern of the event that triggers the flow or the deadline of the outlined task chain. For example, in Figure 4 it is well defined that at first *cpu.run()* has to be completely executed, before *communication.send()* is scheduled etc.. As activity diagrams are more complex concerning their behaviour than most analysis tools, there are restrictions for the modelling of runtime situations, e.g., no hierarchy is allowed.

The SAV can be easily extended, if necessary. If a scheduling analysis tool offers more possibilities to describe or to analyse a system (e.g., a different scheduling algorithm) and needs more system parameters for it, these parameters have to be part of the SAV. Therefore, the view can be extended with new tagged values that offer the possibility to add the necessary parameters to the system description (added to Table 1).

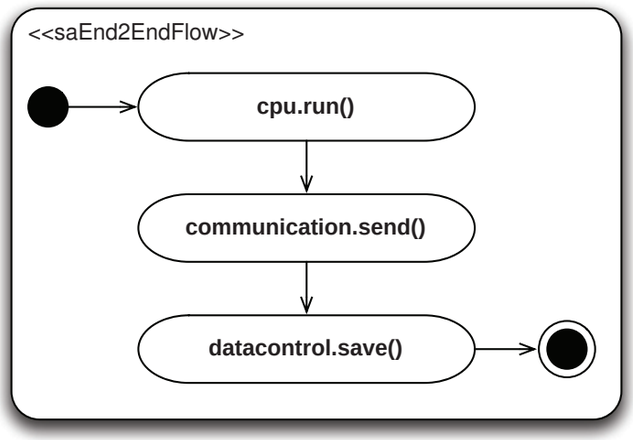


Fig. 4. Workload situation in a SAV

2.2 Abstraction of the design model

The first step of the methodology is the abstraction of the Design Model to the SAV. The Design Model is used as a basis for the scheduling analysis. The basic idea is to find the relevant parts from the Design Model and abstract them in the format of the SAV. Hence, all relevant information for the analysis is identified and transformed into the format of the SAV.

The UML offers many possibilities to describe things. Consequently, most UML Design Models do look different. Even similar things can be described using different expressions (e.g., behaviour could be described using activity diagrams, sequence diagrams, or state charts; deployment can be described using deployment diagrams, but it is also possible to describe it using class diagrams). As a result, an automatic abstraction of the parts necessary for a scheduling analysis is not possible.

As the integration of the scheduling analysis in a UML based development process should be an adaption to the already defined and established development process and not the other way around, our approach offers a flexibility to abstract different Design Models. Our approach uses a rule-based abstraction. The developer creates rules, e.g., "all elements of type device represent a CPU". Based on these rules, the automatic abstraction creates a SAV with the elements of the Design Model. This automatic transformation is implemented for Papyrus for UML[2].

There are two types of rules for the abstraction. The first type describes the element in the Design Model and its representation in the SAV:

```
ID(element_type, diagram_name, limit1, ...)-> sav_element_type
```

The rule begins with a unique ID, afterwards the element type is specified (element_type). The following element types can be abstracted: method, class, device, artifact. Then, the diagram can be named on which the abstraction should be done (diagram_name). Finally, it is possible to define limitations, all separated by commas. Limitations can be string filtering or stereotypes. After the arrow, the corresponding element in the SAV can be named. All elements that have a stereotype in the SAV are possible (see Table 1).

[2] http://www.papyrusuml.org

The second type of rules abstracts references:

```
(element_type, diagram_name, ID_ref1, ID_ref2)-> Allocation
```

The rule specifies mappings in the SAV. It begins with the element type. Here, only deploys or associations are allowed. After the name of the diagram, the developer has to give two IDs of the basic rules. The abstraction searches for all elements that are affected by the first given rule (ID_ref1) and the second given rule (ID_ref2) and checks, if there is a connection between them, specified through the given element_type. If this is the case, an allocation between the abstracted elements in the SAV is created.

Additionally, it is possible to use the ID_ref as a starting point to use different model elements that are connected to the affected element (e.g., ID_ref1 affects methods, then ID_ref1.class affects the corresponding classes that contain the methods).

Figure 5 gives a simple example of an abstraction. On the left side the Design Model is represented and on the right side, the abstracted SAV. At the beginning, only the left side exists. In this example, one modelling convention for the Design Model was to add the string "_task" to all method names that represent tasks. Another convention was to add "_res" to all class names that represent a CPU.

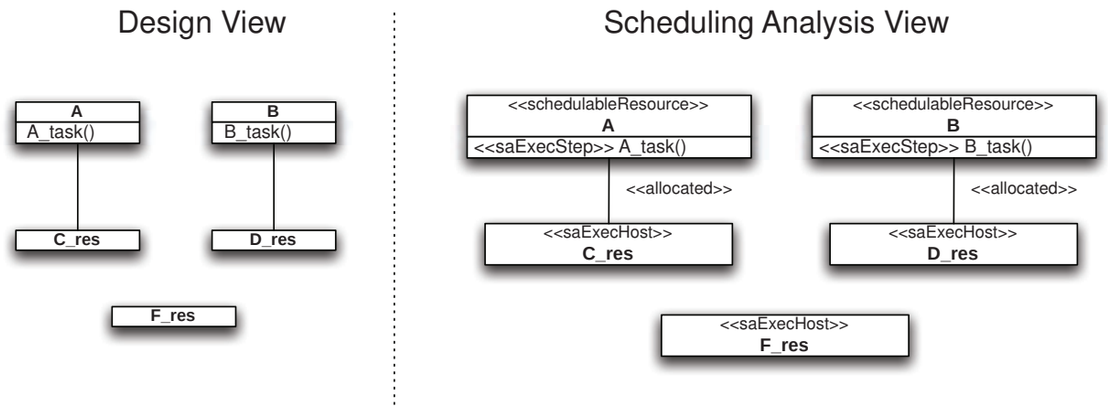


Fig. 5. Simple example of an abstraction from the Design Model to the SAV

The following rules define the abstraction of tasks and CPUs:

```
A1(Class, ''*'', ''*_res'')->CPU
A2(Method, ''*'', ''*_task'')->Task
```

The mapping is described using the following rule:

```
(Association, ''*'', A2.class, A1)->Allocation
```

This rule is used on associations in all diagrams (Association, ''*''). All methods that are part of classes (A2.class), which are affected by rule A2, that do have an association with a class that is affected by rule A1, are abstracted to allocations.

It is also possible to define, that model elements in one diagram are directly connected to a model element in another diagram using "<=>" (e.g., a package in one diagram represents a

device in another diagram by using the construct “package<=>device”, for more information see our case study in Section 3 and Bruechert (2011).

The automatic abstraction of the behaviour using activity diagrams for scheduling analysis is as follows: Using the defined rules, it will be determined which methods are to be considered in the SAV. The corresponding activity diagrams are analysed (all actions that represent a task). All other actions will be deleted and skipped. All activities that do not contain a method representing a task will be removed. In a similar way this is done with sequence diagrams and state machines.

Besides the creating of the SAV during the process of abstraction, there is also a synchronisation table created that documents the abstraction. The table describes the elements in the Design Model and their representation in the SAV. This table is later used for the synchronisation (see Section 2.6). More details about the abstraction and the synchronisation (including a formal description) can be found in Bruechert (2011).

As it is possible that there is still architectural or behaviour information missing after the abstraction, we created additional tool support for the UML case tool Papyrus to help the developer add elements to the SAV (Hagner & Huhn (2008)). We implemented a palette for simpler adding of SAV elements to the system model. Using this extension, the developer does not need to know the relevant stereotypes of how to apply them.

2.3 Parameterisation

After the abstraction, there is still important information missing, e.g., priorities, execution times. The MARTE profile elements are already attached to the corresponding UML element but the values to the parameters are missing. Depending on the stage of the development, these parameters must be added by experts or specialised tools. In early development phases, an expert might be able to give information or, if COTS[3] are used, measured values from earlier developments can be used. In later phases, tools, like aiT (Ferdinand et al. (2001)), T1[4], or Traceanalyzer[5] can be used for automatic parameterisation of the SAV. These tools use static analysis or simple measurement for finding the execution times or the execution patterns of tasks. aiT observes the binary and finds the worst-case execution cycles. As the tool also knows the processor the binary will be executed on, it can calculate the worst-case execution times of the tasks. T1 orchestrates the binary and logs parameters while the tasks are executed on the real platform. Traceanalyzer uses measured values and visualises them (e.g., examines patterns, execution times).

In other development approaches, the parameters are classified with an additional parameter depending on its examination. For example, AUTOSAR[6] separates between worst-case execution time, measured execution time, simulated execution time, and rough estimation of execution time. There are possibilities to add these parameters to the SAV, too. This helps the developer understanding the meaningfulness of the analysis results (e.g., results based on worst-case execution times are more meaningful than results based on rough estimated values).

[3] Components-off-the-shelf
[4] http://www.gliwa.com/e/products-T1.html
[5] http://www.symtavision.com/traceanalyzer.html
[6] The AUTOSAR Development Partnership. Automotive Open System Architecture. http://www.autosar.org

Additionally, depending on the chosen scheduling algorithm, one important aspect in this step is the definition of the task priorities. Especially in early phases of a development this can be difficult. There are approaches to find automatically parameters like priorities based on scheduling analysis results. In our method, we suggest to define the priorities manually, do the analysis, and create new variants of the system (see Section 2.5). If, at an early stage, priorities are not known and (more or less) unimportant, the priorities can be set arbitrary, as analysis tools demand these parameters to be set.

2.4 Completeness check and analysis

After the parameterisation is finished and the system is completely described, with respect to the scheduling parameters, an analysis is possible. Before the analysis is done, the system is checked if all parameters are set correctly (e.g., every tasks has to have an execution time; if round robin is set as a scheduling algorithm, tasks need to have a parameter that defines the slot size).

For the analysis, specialised tools are necessary. There are e.g., SymTA/S (Henia et al. (2005)), MAST (Harbour et al. (2001)), and TIMES (Fersman & Yi (2004)). All of these tools are using different meta models. Additionally, these tools have different advantages and abilities.

We created an automatic transformation of the SAV to the scheduling analysis tool SymTA/S (Hagner & Goltz (2010)) and to TIMES (Werner (2006)) by using transformation languages (e.g., ATLAS Group (INRIA & LINA) (2003)). As all information necessary for an analysis is already included in the SAV, a transformation puts all information of the SAV into the format of the analysis tool, triggers the analysis, and brings back the analysis results into the SAV. The developer does not need to see SymTA/S or TIMES, remodel the system in the format of the analysis tool, and does not need to know how the analysis tool works.

SymTA/S links established analysis algorithms with event streams and realises a global analysis of distributed systems. At first, the analysis considers each resource on its own and identifies the response time of the mapped tasks. From these response times and the given input event model it calculates the output event model and propagates it by the event stream. If there are cyclic dependencies, the system is analysed from a starting point iteratively until reaching convergence.

SymTA/S is able to analyse distributed systems using different bus architectures and different scheduling strategies for processors. However, SymTA/S is limited concerning behavioural description, as it is not possible to describe different workload situations. The user has to define the worst-case workload situation or has to analyse different situation independently. Anyhow, as every analysis tool has its advantages it is useful not to use only one analysis tool.

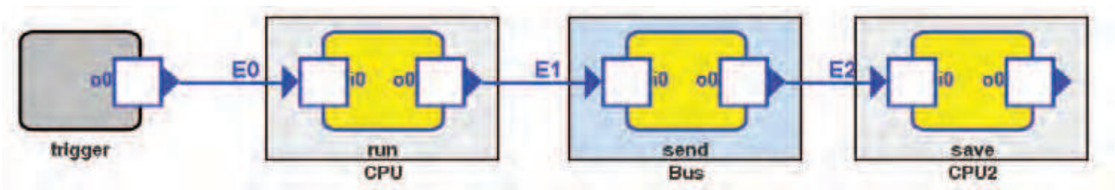


Fig. 6. Representation in SymTA/S

The example depicted in Figure 6 is the SymTA/S representation of the system described in Section 2.1 and illustrated in Figure 3 and Figure 4. There is one source (trigger), two

CPUs (CPU and CPU2), which execute two tasks (run and save), and a bus (Bus) with one communication task (send). All tasks are connected using event streams, representing task chains.

As already mentioned, it is also possible to use other tools for scheduling analysis, e.g., TIMES (Fersman & Yi (2004)). TIMES is based on UPPAAL (Behrmann et al. (2004)) and uses timed automata (Alur & Dill (1994)) for an analysis. Consequently, the results are more precise compared to the over approximated results from SymTA/S. Besides this feature, it also offers code generator for automatic synthesis of C-code on LegoOS platform from the model and a simulator, in which the user can validate the dynamic behaviour of the system and see how the tasks execute according to the task parameters and a given scheduling policy. The simulator shows a graphical representation of the generated trace showing the time points when the tasks are released, invoked, suspended, resumed, and completed. On the other side, as UPPAAL is a model checker, the analysis time could be very long for complex systems due to state space explosion. TIMES is only able to analyse one processor systems. Consequently, for an analysis of distributed systems other tools are necessary.

Figure 7 gives a TIMES representation of the system we described in Section 2.1, with the limitation that all tasks are executed on the same processor. The graph describes the dependencies of the tasks.

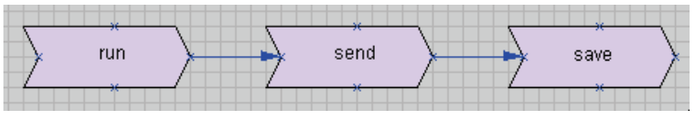


Fig. 7. Representation in TIMES

In TIMES it is also possible to specify a more complex task behaviour/dependency description by using timed automata. Figure 8 gives the example from Section 2.1 using timed automata to describe the system. Timed automata contain locations (in Figure 8 Location_1, Location_2, and Location_3) and switches, which connect the locations. Additionally, the system can contain clocks and other variables. A state of a system is described using the location, the value of the clocks, and the value of other variables. The locations describe the task triggering. By entering a location, the task connected to the location is triggered. Additionally, invariants in locations or guards on the switches are allowed. The guards and the invariants can refer on clocks or other variables.

After the analysis is finished, the analysis results are published in the SAV. In the SAV, the developer can see if there are tasks or task chains that miss their deadlines or if there are resources with a utilisation higher than 100%. The SAV provides tagged values that are used to give the developer a feedback about the analysis results. One example is given in Figure 2, where the respT tagged value is set with a variable ($r1), which means that the response time of the corresponding task is entered at this point after the analysis (this is done automatically by our implemented transformations). There are also other parameters, which give a feedback to the developer (see also Table 1, all are set automatically by the transformations):

- The **respT** tagged values gives a feedback about the worst-case response time of the (communication) tasks and is offered by the «saExecStep» and the «saCommHost» stereotype.
- As the respT, the **end2endT** tagged values offers the worst case response time, in this case for task paths/task chains and is offered by the «saEnd2EndFlow» stereotype. It is not

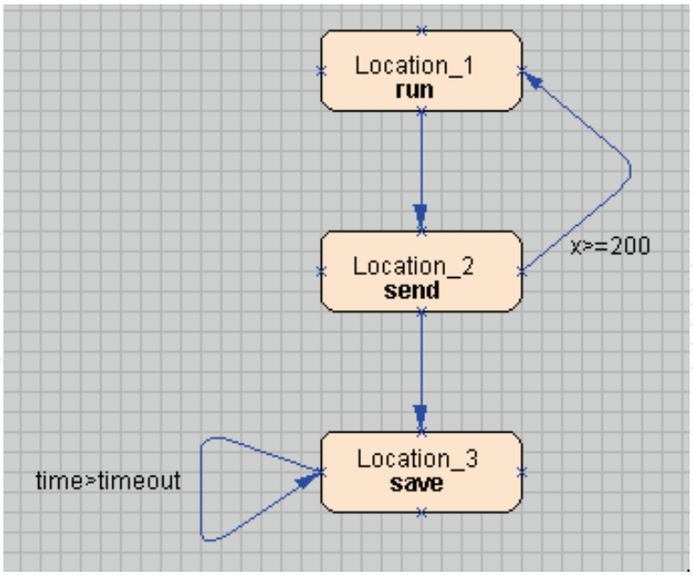


Fig. 8. More advanced representation in TIMES

a summation of all worst-case response times of the tasks that are part of the path, but a worst-case calculated response time of the whole path examined by the scheduling analysis tool (for more details see Henia et al. (2005)).

- The «saExecHost» and the «saCommHost» stereotypes offer a **Utilization** tagged value that gives a feedback about the load of CPUs or busses. If the value is higher than 100% this resource is not schedulable (and the isShed tagged value is false, too). If this value is under 100%, the system might be schedulable (depending on the other analysis results). A high value for this variable always indicates a warning that the resource could be overloaded.
- The tagged value **isShed** gives a feedback if the tasks mapped on this resource are schedulable or not and is offered by the «saExecHost» and the «saCommHost» stereotypes. The tagged values are connected to the Utilization tagged value (e.g., if the utilisation is higher than 100%, the isShed tagged value is false). The **isShed** is also offered by the «saEnd2EndFlow» stereotype. As the «saEnd2EndFlow» stereotype defines parameters for task paths/task chains, the **isShed** tagged value gives a feedback whether the deadline for the path is missed or not.

Using these tagged values, the developer can find out if the system is schedulable by checking the isShed tagged value of the «seEnd2EndFlow» stereotype. If the value is false, the developer has to find the reason why the scheduling failed using the other tagged values. The end2EndT tagged value shows to what extent the deadline is missed, as it gives the response time of the task paths/task chains. The response times of the tasks and the utilisation of the resources give also a feedback where the bottleneck might be (e.g., a resource with a high utilisation and tasks scheduled on it with long response times are more likely a bottleneck compared to resources with low utilisation).

If this information is not sufficient, the developer has to use the scheduling analysis tools for more detailed information. TIMES offers a trace to show the developer where deadlines are missed. SymTA/S offers Gantt charts for more detailed information.

2.5 Variant management

Variant management helps the developer to handle different versions of a SAV. In case of an unsuccessful analysis result (e.g., system is not schedulable) the developer might want to change parameters or distributions directly in the SAV without having to synchronise with the Design Model first, but wants to keep the old version as a backup. Even when the system is schedulable, the developer might want to change parameters to see if it is possible to save resources by using lower CPU frequencies, slower CPUs, or slower bus systems.

It is also possible to add external tools that find good distributions of tasks on resources. Steiner et al. (2008) explored the problem to determine an optimised mapping of tasks to processors, one that minimises bus communication and still, to a certain degree, balances the algorithmic load. The number of possibilities for the distribution of N tasks to M resources is M^N. A search that evaluates all possible patterns for their suitability can be extremely costly and will be limited to small systems. However, not all patterns represent a legal distribution. Data dependencies between tasks may cause additional bus communication if they are assigned to different resources and communication over a bus is much slower than a direct communication via shared memory or message passing on a single processor. Thus, minimising bus communication is an important aspect when a distribution pattern is generated. To use additionally provided CPU resources and create potential for optimisations also the balance of the algorithmic load has to be considered.

In Steiner et al. (2008) the distribution pattern generation is transformed into a graph partitioning problem. The system is represented as an undirected graph, its node weights represent the worst-case execution time of a task and an edge weight corresponds to the amount of data that is transferred between two connected tasks. The algorithm presented searches for a small cut that splits the graph into a number of similar sized partitions. The result is a good candidate for a distribution pattern, where bus communication is minimised and the utilisation of CPU resources is balanced.

Another need for variant management is different criticality levels, necessary e.g., in the ISO 26262 (Road Vehicles Functional Safety (2008)). Many safety-critical embedded systems are subject to certification requirements; some systems are required to meet multiple sets of certification requirements from different certification authorities. For every Safety Integrity Level (SIL) a different variant of the system can be used. In every different variant, the mapping of the tasks and the priorities will be the same. However, the values for the scheduling parameters can be different, e.g., the execution times, as they have to be examined using different methods for each different SIL and consequently for each variant representing a different SIL (see Section 2.3 for different possibilities to parameterise the SAV).

2.6 Synchronisation

If the developer changes something in the SAV (due to analysis results) later and wants to synchronise it with the Design Model, it is possible to use the rule-based approach. During the abstraction (Section 2.2), a matching table/synchronisation table is created and can be used for synchronisation. This approach also works the other way around (changes in the Design Model are transferred to the SAV). During a synchronisation, our implementation is updating the synchronisation table automatically.

One entry in the synchronisation table has two columns. The first specifies the item in the Design Model and the second the corresponding element in the SAV. According to the two rule types (basic rule or reference rule), two types of entries are distinguished in the

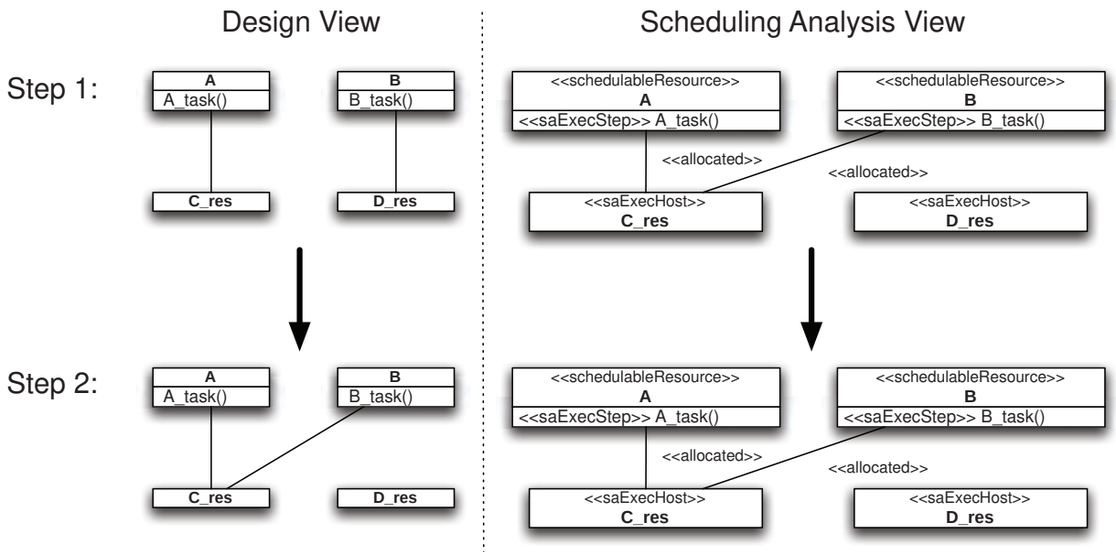


Fig. 9. Synchronisation of the Design Model and the SAV

synchronisation table. The basic entry corresponds to the abstraction of an item that is described by a basic rule. The single entry is described in a Design Model column and a SAV column. The Design Model column contains the element type in the Design Model, the XMI[7] ID in the Design Model, and the name in the Design Model. The SAV column contains the element type, the XMI ID, and the name in the SAV. Regarding a reference entry, based on the reference rules, the Design Model column contains the element type, the XMI ID, the XMI IDs of the two elements with the connection from the Design Model. The SAV column contains the element type, the XMI ID, and, again the XMI IDs from the elements that are connected.

Design Model	**SAV**
Class, ID_C_res, C_res	CPU, ID_C_res, C_res
Class, ID_D_res, D_res	CPU, ID_D_res, D_res
Method, ID_A_task, A_task	Task, ID_A_task, A_task
Method, ID_B_task, B_task	Task, ID_B_task, B_task
Association, ID, ID_A_task, ID_C_res	Allocation, ID, ID_A_task, ID_C_res
Association, ID, ID_B_task, ID_D_res	Allocation, ID, ID_B_task, ID_D_res

Table 2. The synchronisation table before the synchronisation

Figure 9 gives a simple example, where synchronisation is done. It is based on the example given in Section 2.2 and illustrated in Figure 5. Table 2 gives the corresponding synchronisation table before the synchronisation (for simplification we use a variable name for the XMI IDs).

Because of analysis results, the mapping has been changed and B_task() will now be executed on CPU C_res. Consequently, the mapping has changed in the SAV column in the synchronisation table (see last row in Table 3). Additionally, this is happening in the Design

[7] XML Interchange Language (Object Management Group (1998))

Design Model	SAV
Class, ID_C_res, C_res	CPU, ID_C_res, C_res
Class, ID_D_res, D_res	CPU, ID_D_res, D_res
Method, ID_A_task, A_task	Task, ID_A_task, A_task
Method, ID_B_task, B_task	Task, ID_B_task, B_task
Association, ID, ID_A_task, ID_C_res	Allocation, ID, ID_A_task, ID_C_res
Association, ID, ID_B_task, ID_C_res	Allocation, ID, ID_B_task, ID_C_res

Table 3. The synchronisation table after the synchronisation

Model column and finally in the Design Model, too (see Figure 9). More details can be found in Bruechert (2011)

3. Case study

In this Section we want to apply the above introduced methodology to the development of a robotic control system of a parallel robot developed in the Collaborative Research Centre 562 (CRC 562)[8]. The aim of the Collaborative Research Centre 562 is the development of methodological and component-related fundamentals for the construction of robotic systems based on closed kinematic chains (parallel kinematic chains - PKMs), to improve the promising potential of these robots, particularly with regard to high operating speeds, accelerations, and accuracy (Merlet (2000)). This kind of robots features closed kinematic chains and has a high stiffness and accuracy. Due to low moved masses, PKMs have a high weight-to-load-ratio compared to serial robots. The demonstrators which have been developed in the research centre 562 move very fast (up to 10 m/s) and achieve high accelerations (up to 100 m/s^2). The high velocities induced several hard real-time constraints on the software architecture *PROSA-X* (Steiner et al. (2009)) that controls the robots. *PROSA-X* (**P**arallel **R**obots **S**oftware **A**rchitecture - e**X**tended) can use multiple control PCs to distribute its algorithmic load. A middleware (*MiRPA-X*) and a bus protocol that operates on top of a FireWire bus (IEEE 1394, Anderson (1999)) (*IAP*) realise communication satisfying the hard real-time constraints (Kohn et al. (2004)). The architecture is based on a layered design with multiple real-time layers within QNX[9] to realise e.g., a deterministic execution order for critical tasks (Maass et al. (2006)). The robots are controlled using cyclic frequencies between 1 and 8 kHz. If these hard deadlines are missed, this could cause damage to the robot and its environment. To avoid such problems, a scheduling analysis based on models ensures the fulfilment of real-time requirements.

Figure 10 and Figure 11 present the Design Model of the robotic control architecture. Figure 10 shows a component diagram of the robotic control architecture containing the hardware resources. In this variant, there is a “Control_PC1” that performs various computations. The “Control_PC1” is connected via a FireWire data bus with a number of digital signal processors (“DSP_1-7”), which are supervising and controlling the machine. Additionally, there are artefacts («artifact») that are deployed (using the associations marked with the «deploy» stereotype) to the resources. These artefacts represent software that is executed on the corresponding resources.

The software is depicted in Figure 10. This diagram contains packages where every package represents an artefact depicted in Figure 11 (the packages IAP_Nodes_2-7 have been omitted

[8] http://www.tu-braunschweig.de/sfb562
[9] QNX Neutrino is a micro kernel real-time operating system.

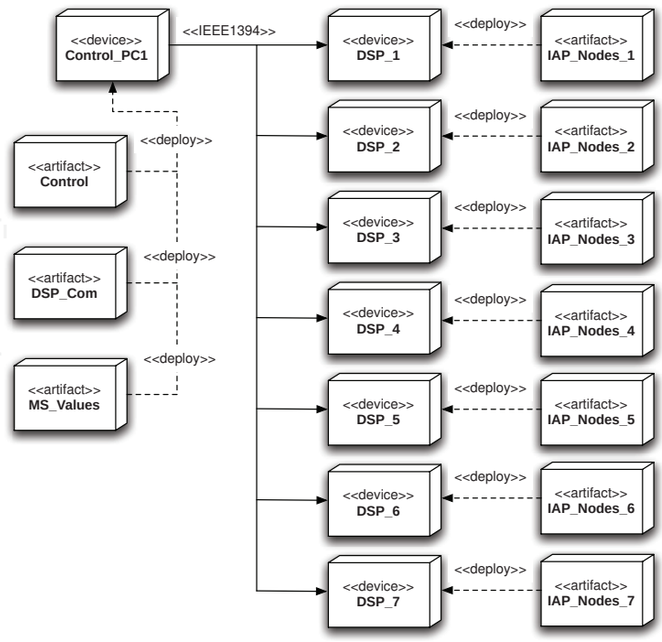


Fig. 10. Component diagram of the robotic control architecture

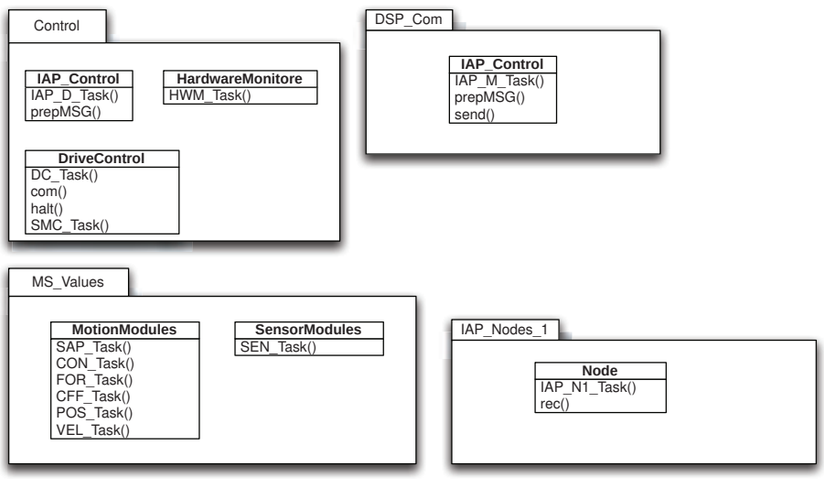


Fig. 11. Package diagram of the robotic control architecture

due to space and are only represented by IAP_Nodes_1). The packages are containing the software that is executed on the corresponding resource. The packages are containing classes and the classes are containing methods. Some methods represent tasks. These methods are marked using the addition of "_Task" to their name (e.g., the package "Control" contains the class "DriveControl" and this class contains three methods, where method *DC_Task()* represents a task). The tasks that are represented using methods have the following functionality:

- *IAP_D*: This instance of the *IAP* bus protocol receives the *DDTs (Device Data Telegram)* that contain the instantaneous values of the DSP nodes over the FireWire bus.

- *HWM*: The *Hardware Monitoring* takes the instantaneous values received by the *IAP_D* and prepares them for the control.
- *DC*: The *Drive Controller* operates the actuators of the parallel kinematic machine.
- *SMC*: The *Smart Material Controller* operates the active vibration suppression of the machine.
- *IAP_M*: This instance of the bus protocol *IAP* sends the setpoint values, calculated by DC and SMC, to the DSP node.
- *CC*: The *Central Control* activates the currently required sensor and motion modules (see below) and collects their results.
- *CON*: *Contact Planner*. Combination of power and speed control. For the end effector of the robot to make contact with a surface.
- *FOR*: *Force Control*, sets the force for the end effector of the robot.
- *CFF*: Another *Contact Planner*, similar to CON.
- *VEL*: *Velocity Control*, sets the speed for the end effector of the robot.
- *POS*: The *Position Controller* sets the position of the end effector.
- *SAP*: The *Singularity Avoidance Planner* plans paths through the work area to avoid singularities.
- *SEN*: An exemplary *Sensor Module*.

There are three task paths/task chains with real-time requirements. The first task chain receives the instantaneous values and calculates the new setpoint values (using the tasks IAP_D, HWM, DC, SMC). The deadline for this is 250 microseconds. The second task chain contains the sending of the setpoint values to the DSPs and their processing (using tasks IAP_M, MDT, IAP_N1, ..., IAP_N7, DDT1, ..., DDT7). This must be finished within 750 microseconds. The third chain comprises the control of the sensor and motion modules (using tasks CC, CON, FOR, CFF, POS, VEL, SEN, SAP) and has to be completed within 1945 microseconds. The tasks chains including their dependencies were described using activity diagrams.

To verify these real-time requirements we adapted out methodology to the Design Model of the robotic control architecture. The first step was the abstraction of the scheduling relevant information and the creation of the corresponding SAV. As described in Section 2.2, we had to define rules for the abstraction. The following rules were used:

```
A1(Device, ''ComponentDiagram'', ''*'')->CPU
A2(Method, ''PackageDiagram'', ''*_Task'')->Task
```

Rule A1 creates all CPUs in the SAV (classes containing the «saExecHost» stereotype). Rule A2 creates schedulable resources containing the tasks (methods with the «saExecStep» stereotype). Here, we were using the option to sum all tasks that are scheduled on one resource into one schedulable resource representing class (see Figure 12). The corresponding rule to abstract the mapping is:

```
(Deploy, ''*'', A2.class.package<=>Artifact, A1)->Allocation
```

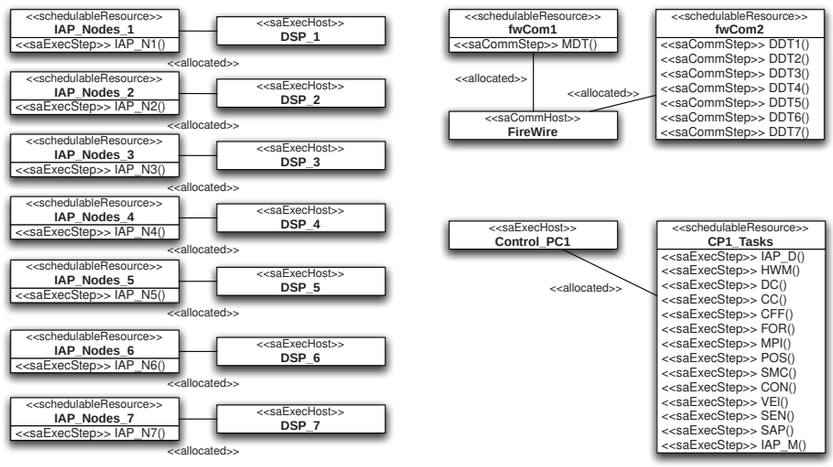


Fig. 12. The architectural view of the PROSA-X system

The packages that contain classes that contain methods that are effected by rule A2, under the assumption that there is an artefact that represents the package in another diagram, are taken into account. It is observed if there is a deploy element between the corresponding artefact and a device element that is effected by rule A1. If this is the case, there is an allocation between these elements. As not all necessary elements are described in the Design Model, e.g., the FireWire bus was not abstracted; it has to be modelled manually in the SAV, as it is important for the scheduling analysis. The result (the architectural view of the SAV) is presented in Figure 3

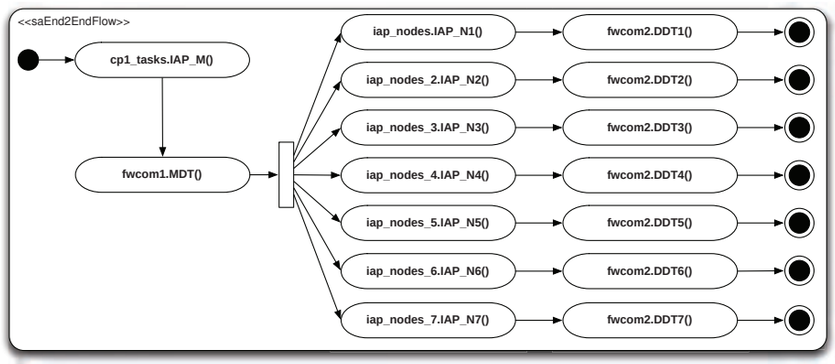


Fig. 13. Sending of the setpoint values to the DSPs

Additionally, a runtime view is created and the behaviour (the workload situations) are created. Figure 13 represents the task chain that sends the setpoint values to the DSPs and describes their processing (IAP_M, MDT, IAP_N1, …, IAP_N7, DDT1, …, DDT7). The deadline is 750 microseconds.

Besides the SAV, a synchronisation table is created. Exemplarily, it is presented in Table 4.

After the SAV is created, it can be parameterised. We have done this by expert knowledge, measuring, and monitoring prototypes. Using these methods, we were able to set the necessary parameters (e.g., execution times, activation pattern, priorities).

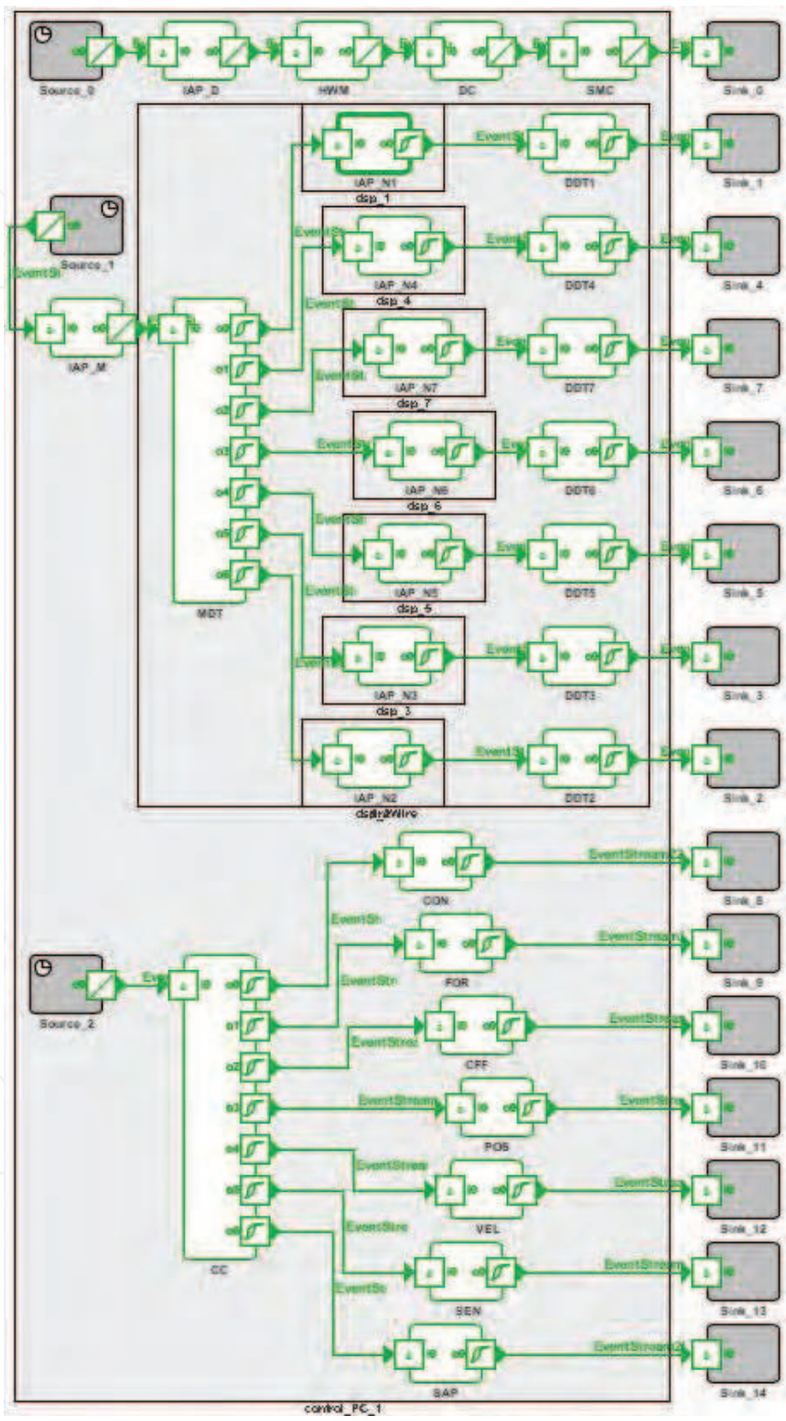


Fig. 14. The SymTA/S description of the PROSA-X system

Design View	SAV
Method, ID, IAP_D_Task	Task, ID, IAP_D_Task
Device, ID, Control_PC1	CPU, ID, Control_PC1
Deploy, ID, IAP_D_Task.IAP_Control.Control <=>Control, Control_PC1	Association, ID, IAP_D_Task, Control_PC1
...	...

Table 4. The synchronisation table of the robotic control system

As we have created automatic transformation to the scheduling analysis tool SymTA/S, the transformation creates a corresponding SymTA/S model and makes it possible to analyse the system. The completeness check is included in the transformation. Afterwards, the output model was analysed by SymTA/S and the expectations were confirmed: The analysis was successful, all paths keep their real-time requirements, and the resources are not overloaded. The SymTA/S model is depicted in Figure 14.

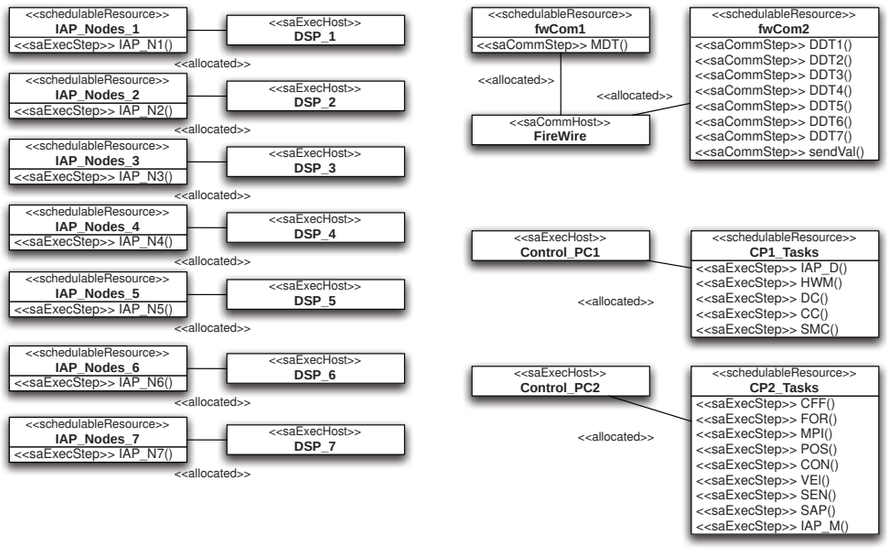


Fig. 15. The new architectural view of the PROSA-X system containing a second control pc

After the successful analysis, the results are automatically published back into the SAV (see Section 2.4). However, we created a new variant of the same system to observe if a faster distribution is possible by adding a new control pc ("Control_PC2"). Consequently, we changed the distribution and added tasks to the second control pc that were originally executed on "Control_PC1") (see Figure 15). As the tasks are more distributed now, we had to add an additional communication task (*sendVal()*) to transfer the results of the calculations. We went through the parameterisation and the analysis again and found out, that this distribution is also valid in terms of scheduling.

As a next step, we can synchronise our results with the Design Model. During the synchronisation, the relevant entries in the synchronisation table were examined. New entries (e.g., for the new control pc) are created and, consequently, the mapping of the artefact "Control" is created corresponding to the SAV. The result is depicted in Figure 16.

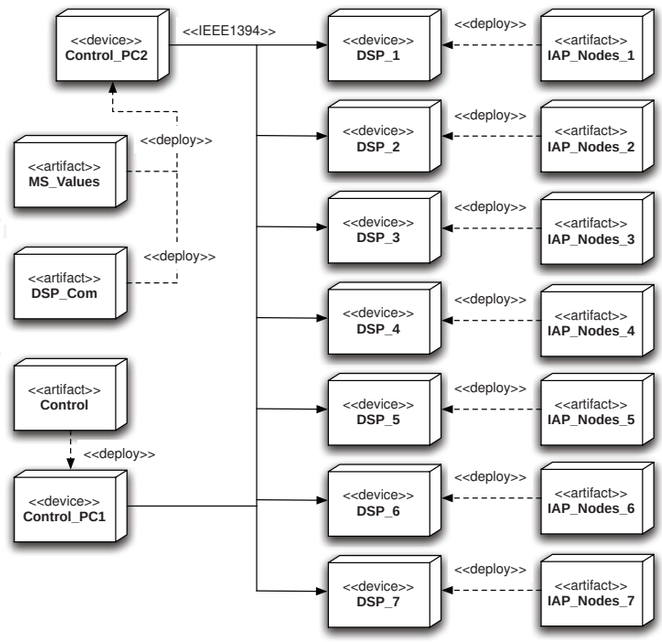


Fig. 16. Component diagram after the synchronisation containing the new device

4. Adapting the approach to other non-functional properties

The presented approach can be adapted to other non-functional requirements (e.g., power consumption or reliability). For every non-functional requirement, there can be an individual view to help the developer concentrate on the aspect he/she is working on. This is drawn upon the cognitive load theory (Sweller (2003)). Consequently, besides the view, a methodology (like the one in this paper) is necessary. Depending on which requirements are considered, the methodologies differ from each other; other steps are necessary and the analysis is different. Additionally, there can be dependencies between the different views (e.g., between the SAV and a view for power consumption as we will explain later).

Power is one of the important metrics for optimisation in the design and operation of embedded systems. One way to reduce power consumption in embedded computing systems is processor slowdown using frequency or voltage. Scaling the frequency and voltage of a processor leads to an increase in the execution time of a task. In real-time systems, we want to minimise energy while adhering to the deadlines of the tasks. Dynamic voltage scaling (DVS) techniques exploit the idle time of the processor to reduce the energy consumption of a system (Aydin et al. (2004); Ishihara & Yasuura (1998); Shin & Kim (2005); Walsh et al. (2003); Yao et al. (1995)).

We defined a Power Consumption Analysis View (PCAV), according to the SAV (Hagner et al. (2011)), to give the developer the possibility to add energy and power consumption relevant parameters to the UML model. Therefore, we created the PCAV profile as an extension of the MARTE profile and an automatic analysis algorithm. The PCAV supports DVS systems. In Figure 17 an example for a PCAV is given. It uses different stereotypes than the SAV as there are different parameters to describe. However, the implementation is similar to the SAV. Additionally, we developed and implemented an algorithm to find a most power aware, but still real-time schedulable system configuration for a DVS system.

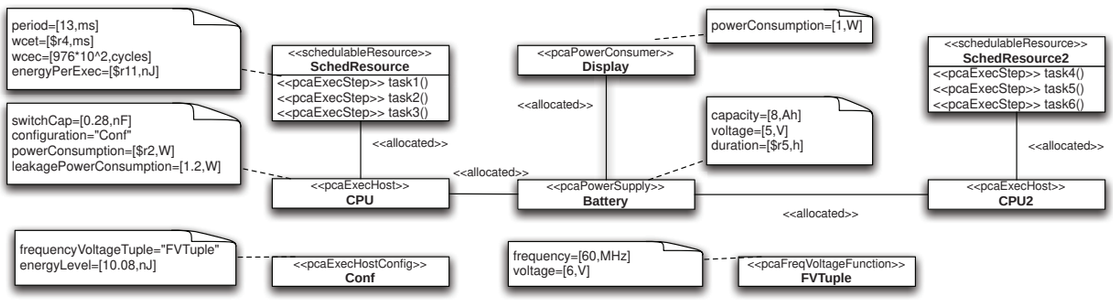


Fig. 17. Power Consumption Analysis View (PCAV)

The power consumption and the scheduling depend on each other (Tavares et al. (2008)). If slower hardware is used to decrease the power consumption, the scheduling analysis could fail due to deadlines that are missed because tasks are executed slower. If faster hardware is used, the power consumption increases. The solution is to find a system configuration that is most power aware but still real-time with respect to their deadline. For our algorithm, we were using both, the SAV and the PCAV. Based on the Design Model we created both views, used the PCAV to do the power consumption analysis and to calculate the execution times and then used the SAV to check the real-time capabilities (Aniculaesei (2011)).

5. Conclusion

In this chapter we have presented a methodology to integrate the scheduling analysis in a UML based development. The methodology is based on the Scheduling Analysis View and contains steps, how to create this view, independently how the UML Design Model looks like, how to process with this view, analyse it, handle variants, and synchronise it with the Design Model. We have presented this methodology in a case study of a robotic control system. Additionally, we have given an outlook on the possibility to create new views for other non-functional requirements.

Future work can be to add additional support concerning the variant management to comply with standards (e.g., Road Vehicles Functional Safety (2008)). Other work can be done by creating different views for other requirements and observe the dependencies between the views.

6. Acknowledgment

The authors would like to thank Symtavision for the grant of free licenses.

7. References

Alur, R. & Dill, D. L. (1994). A theory of timed automata, *Theoretical Computer Science* 126(2): 183 – 235.
URL: *http://www.sciencedirect.com/science/article/pii/0304397594900108*

Anderson, D. (1999). *FireWire system architecture (2nd ed.): IEEE 1394a*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Aniculaesei, A. (2011). *Uml based analysis of power consumption in real-time embedded systems*, Master's thesis, TU Braunschweig.

Argyris, I., Mura, M. & Prevostini, M. (2010). Using marte for designing power supply section of wsns, *M-BED 2010: Proc. of the 1st Workshop on Model Based Engineering for Embedded Systems Design (a DATE 2010 Workshop)*, Germany.

Arpinen, T., Salminen, E., Hännikäinen, T. D. & Hännikäinen, M. (2011). Marte profile extension for modeling dynamic power management of embedded systems, *Journal of Systems Architecture, In Press, Corrected Proof* .

ATLAS Group (INRIA & LINA) (2003). Atlas transformation language, http://www.eclipse.org/m2m/atl/.

Aydin, H., Melhem, R., Mossé, D. & Mejía-Alvarez, P. (2004). Power-aware scheduling for periodic real-time tasks, *IEEE Trans. Comput*. pp. 584–600.

Behrmann, G., David, R. & Larsen, K. G. (2004). A tutorial on uppaal, *A tutorial on UPPAAL*, Springer, pp. 200–236.

Bruechert, A. (2011). *Abstraktion und synchronisation von uml-modellen fÃijr die scheduling-analyse*, Master's thesis, TU Braunschweig.

Espinoza, H., Servat, D. & Gérard, S. (2008). Leveraging analysis-aided design decision knowledge in uml-based development of embedded systems, *Proceedings of the 3rd international workshop on Sharing and reusing architectural knowledge*, SHARK '08, ACM, New York, NY, USA, pp. 55–62.
URL: *http://doi.acm.org/10.1145/1370062.1370078*

Faugere, M., Bourbeau, T., Simone, R. & Gerard, S. (2007). MARTE: Also an UML profile for modeling AADL applications, *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pp. 359–364.

Ferdinand, C., Heckmann, R., Langenbach, M., Martin, F., Schmidt, M., Theiling, H., Thesing, S. & Wilhelm, R. (2001). Reliable and precise wcet determination for a real-life processor, *EMSOFT '01: Proc. of the First International Workshop on Embedded Software*, Springer-Verlag, London, UK, pp. 469–485.

Fersman, E. & Yi, W. (2004). A generic approach to schedulability analysis of real-time tasks, *Nordic J. of Computing* 11(2): 129–147.

Hagner, M., Aniculaesei, A. & Goltz, U. (2011). Uml-based analysis of power consumption for real-time embedded systems, *8th IEEE International Conference on Embedded Software and Systems (IEEE ICESS-11), Changsha, China*, Changsha, China.

Hagner, M. & Goltz, U. (2010). Integration of scheduling analysis into uml based development processes through model transformation, *5th International Workshop on Real Time Software (RTS'10) at IMCSIT'10*.

Hagner, M. & Huhn, M. (2007). Modellierung und analyse von zeitanforderungen basierend auf der uml, *in* H. Koschke (ed.), *Workshop*, Vol. 110 of *LNI*, pp. 531–535.

Hagner, M. & Huhn, M. (2008). Tool support for a scheduling analysis view, *Design, Automation and Test in Europe (DATE 08)*.

Hagner, M., Huhn, M. & Zechner, A. (2008). Timing analysis using the MARTE profile in the design of rail automation systems, *4th European Congress on Embedded Realtime Software (ERTS 08)*.

Harbour, M. G., García, J. J. G., Gutiérrez, J. C. P. & Moyano, J. M. D. (2001). Mast: Modeling and analysis suite for real time applications, *ECRTS '01: Proc. of the 13th Euromicro Conference on Real-Time Systems*, IEEE Computer Society, Washington, DC, USA, p. 125.

Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K. & Ernst, R. (2005). System level performance analysis - the SymTA/S approach, *IEEE Proc. Computers and Digital Techniques* 152(2): 148–166.

Ishihara, T. & Yasuura, H. (1998). Voltage scheduling problem for dynamically variable voltage processors, *Proc. of the 1998 International Symposium on Low Power Electronics and Design (ISLPED '98)* pp. 197–202.

Kohn, N., Varchmin, J.-U., Steiner, J. & Goltz, U. (2004). Universal communication architecture for high-dynamic robot systems using QNX, *Proc. of International Conference on Control, Automation, Robotics and Vision (ICARCV 8th)*, Vol. 1, IEEE Computer Society, Kunming, China, pp. 205–210. ISBN: 0-7803-8653-1.

Kruchten, P. (1995). The 4+1 view model of architecture, *IEEE Softw.* 12(6): 42–50.

Maass, J., Kohn, N. & Hesselbach, J. (2006). Open modular robot control architecture for assembly using the task frame formalism, *International Journal of Advanced Robotic Systems* 3(1): 1–10. ISSN: 1729-8806.

Merlet, J.-P. (2000). *Parallel Robots*, Kluwer Academic Publishers.

Object Management Group (1998). XML model interchange(XMI).

Object Management Group (2002). UML profile for schedulability, performance and time.

Object Management Group (2003). Unified modeling language specification.

Object Management Group (2004). UML profile for modeling quality of service and fault tolerance characteristics and mechanisms.

Object Management Group (2007). Systems Modeling Language (SysML).

Object Management Group (2009). UML profile for modeling and analysis of real-time and embedded systems (MARTE).

Road Vehicles Functional Safety, i. O. f. S. (2008). Iso 26262.

Shin, D. & Kim, J. (2005). Intra-task voltage scheduling on dvs-enabled hard real-time systems, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* .

Steiner, J., Amado, A., Goltz, U., Hagner, M. & Huhn, M. (2008). Engineering self-management into a robot control system, *Proceedings of 3rd International Colloquium of the Collaborative Research Center 562*, pp. 285–297.

Steiner, J., Goltz, U. & Maass, J. (2009). Dynamische verteilung von steuerungskomponenten unter erhalt von echtzeiteigenschaften, *6. Paderborner Workshop Entwurf mechatronischer Systeme*.

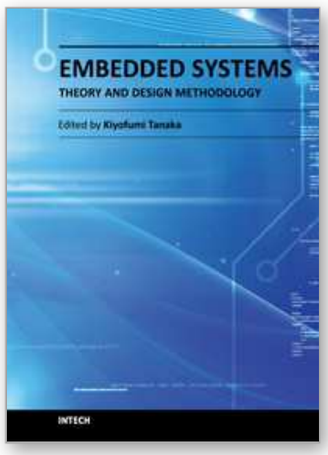
Sweller, J. (2003). Evolution of human cognitive architecture, *The Psychology of Learning and Motivation*, Vol. 43, pp. 215–266.

Tavares, E., Maciel, P., Silva, B. & Oliveira, M. (2008). Hard real-time tasks' scheduling considering voltage scaling, precedence and . . . , *Information Processing Letters* . URL: *http://linkinghub.elsevier.com/retrieve/pii/S0020019008000951*

Walsh, B., Van Engelen, R., Gallivan, K., Birch, J. & Shou, Y. (2003). Parametric intra-task dynamic voltage scheduling, *Proc. of COLP 2003* .

Werner, T. (2006). *Automatische transformation von uml-modellen fuer die schedulability analyse*, Master's thesis, Technische UniversitâĂřt Braunschweig.

Yao, F., Demers, A. & Shenker, S. (1995). A scheduling model for reduced cpu energy, *Proc. of the 36th Annual Symposium on Foundations of Computer Science* .



Embedded Systems - Theory and Design Methodology
Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0167-3
Hard cover, 430 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - the computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permitted various aspects of industry. Therefore, we can hardly discuss our life and society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 19 excellent chapters and addresses a wide spectrum of research topics on embedded systems, including basic researches, theoretical studies, and practical work. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book will be helpful to researchers and engineers around the world.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen