Reusing Real-Time Systems Design Experience Through Modelling Patterns*

Oana Florescu, Jeroen Voeten
Eindhoven University of Technology
Embedded Systems Institute

Marcel Verhoef
Chess Information Technology BV
Radboud University Nijmegen

Henk Corporaal
Eindhoven University of Technology

Abstract

To ensure correctness and performance of real-time embedded systems, early evaluation of properties is needed. Based on design experience for real-time systems and using the concepts of the POOSL language, we introduce modelling patterns that allow easy composition of models for design space exploration. These patterns cover different types of real-time tasks, resources and mappings, and include also aspects that are usually ignored in classical analysis approaches, like task activation latency or execution context switches. The construction of system models can be done by integrating the necessary patterns, as illustrated in two case studies.

1 Introduction

Complex real-time embedded systems are usually comprised of a combination of hardware and software components that are supposed to synchronise and coordinate different processes and activities. From early stages of the design, many decisions must be made to guarantee that the realisation of such a complex machine meets all the functional and non-functional (timing) requirements.

Related research. One of the main problems to address concerns the most suitable architecture of the system such that all the requirements are met. To properly deal with this question, the common approaches are design space exploration and system level performance analysis. An extensive overview of such methodologies is given in [BMIS04] and [Gri04]. They range from analytical computation (Modular Performance Analysis [WTVL05]) to simulation-based estimation (Spade [LvdWVD01], Artemis [PHL+01]). These are often specialised techniques which claim that general purpose languages are ill-suited for system-level analysis. However, due to the heterogeneity and complexity of systems, for the analysis of different aspects different models need to be built and their coupling is difficult. Therefore, a unified model, covering all the interesting aspects, is actually needed to speed up the design process. This is how the Unified Modelling Language (UML) [OMG03] came to be conceived. The language was designed mainly for object-oriented software specification, but recently it was extended (UML 2.0) to include (real-time) systems as well.

During the development of new systems, specific problems are encountered again and again, and experienced designers apply the solutions that worked for them in the past [GHJV95]. These pairs of problem-solution are called *design patterns* and their application helps in getting a design "right" faster. With the increase in the development of real-time systems, design patterns were needed for dealing with issues like concurrency, resource sharing, distribution [Dou02]. As UML has become the standard language for modelling, these patterns are described in UML. However, the semantics of the language is not strong enough to properly deal with the analysis of *real-time* system behaviour. Therefore, an expressive and formal modelling language is required instead in order to capture in a compact model *timing*, *concurrency*, *probabilities* and *complex behaviour*.

Design patterns refer to problems encountered in the design process itself, but problems appear also in the specification of components that are commonly encountered in complex systems [GJN99]. Although components of the analysed systems exhibit some common aspects for all real-time systems (e.g. characteristics of tasks like periodicity or aperiodicity, processors, schedulers and their overheads), they are built everytime from scratch and similar issues are encountered over and over.

Contributions of the paper. To reduce the amount of time needed to construct models for design space exploration, we propose *modelling patterns* to easily compose models for the design space exploration of real-time embedded systems. These modelling patterns, provided as a library, act like templates that can be applied in many different situations by setting the appropriate parameters. They are based on the concepts of a mathematically defined general-purpose modelling language, POOSL [vdPV97], and they are presented as UML diagrams. These boilerplate solutions are a critical success factor for the practical application in an industrial setting and are a step towards the (semi-) automated design space exploration in the early phases of the system life-cycle.

This paper is organised as follows. Section 2 briefly presents the POOSL language, whereas Section 3 provides the modelling patterns. The composition of these patterns into a model is discussed in Section 4 and their analysis approach in Section 5. The results of applying this approach on two case studies are presented in Section 6. Conclusions are drawn in Section 7.

*This work has been carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Senter TS program.

2 POOSL Modelling Language

The Parallel Object-Oriented Specification Language (POOSL) [vdPV97] lies at the core of the Software/Hardware Engineering (SHE) system-level design method. POOSL contains a set of powerful primitives to formally describe concurrency, distribution, synchronous communication, timing and functional features [TOO] of a system into a single executable model. Its formal semantics is based on timed probabilistic labelled transition systems [LS91]. This mathematical structure guarantees a unique and unambiguous interpretation of POOSL models. Hence, POOSL is suitable for specification and, subsequently, verification of correctness and evaluation of performance for real-time systems.

POOSL consists of a *process* part and a *data* part. The process part is used to specify the behaviour of active components in the system, the processes, and it is based on a real-time extension of the Calculus of Communicating Systems (CCS) [Mil89]. The data part is based on traditional concepts of sequential object-oriented programming. It is used to specify the information that is generated, exchanged, interpreted or modified by the active components. As mostly POOSL processes are presented in this paper, fig. 1 presents the relation between the UML class diagram and the POOSL process class specification. The name compartment of the class symbol for process classes is stereotyped with <<process>>. The attributes are named <<parameters>> and allow parameterising the behaviour of a process at instantiation. The behaviour of a process is described by its <<methods>> which may include the specification of sending (!) and/or receiving (?) of <<messages>>[1].

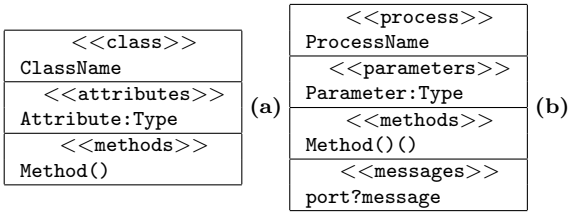

Figure 1: UML (a) vs. POOSL process (b) class

The SHE method is accompanied by two tools, SHESim and Rotalumis. SHESim is a graphical environment intended for incremental specification, modification and validation of POOSL models. Rotalumis is a high-speed execution engine, enabling fast evaluation of system properties. Compared with SHESim, Rotalumis improves the execution speed by a factor of 100. Both tools have been proved to correctly simulate a model with respect to the formal semantics of the language ([Gei02]).

3 Modelling Patterns

Real-time embedded systems usually contain components with common characteristics, like tasks, or computation / communication resources. Based on design experience, modelling patterns can be developed such that when another model of the same or of a similar system needs to be built, the appropriate patterns and their parameters can be chosen and used immediately.

Table 1: Modelling patterns

Y-chart part	Pattern Name	Parameter Name
Application Model	PeriodicTask	period (T) deadline (D) load latency (l) iterations
	AperiodicTask	deadline (D) load latency (l)
Platform Model	Resource	initial latency throughput
	Scheduling	scheduling policy
Environment Model	Environment	arrival stream (Events) upper bound (u) lower bound (l)

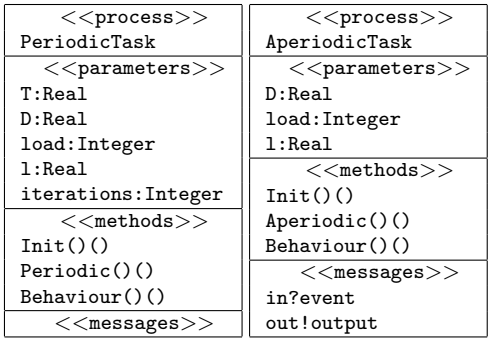

Figure 2: UML task patterns

Table 1 presents the modelling patterns developed and used in the case studies presented in the paper. These patterns comply with the Y-chart, the scheme introduced in [KDVvdW97] for systematic design space exploration. They also cover the model of the system environment, although originally this was not present in the scheme, because real-time systems typically have a continuous interaction with their environment.

The application model is described through real-time tasks, which are characterised by deadline, load (which represents the number of instructions that the task needs to execute), latency of task activation, plus period and number of iterations for periodic tasks. The platform model consists of (computation and/or communication) resources, which are uniformly characterised by an initial latency and throughput, and the scheduling policies that handle the concurrent requests. Furthermore, the model of the environment is characterised by an event stream with a certain distribution of arrival between an upper and a lower bound. The corresponding UML class diagrams of each of these patterns are presented in fig. 2, 3, 4 and 5 respectively, whereas their POOSL specifications can be found in [FVC06].

4 Model Composition

To build a model of a real-time system for design space exploration, its specific components that correspond to

[1] More details about the UML profile for POOSL can be found in [The04].

the modelling patterns described in the previous section must be identified together with their parameters. The names of the necessary patterns and their parameters, together with the specification of the mapping (which task is scheduled on which processor, etc.) and the layout of the platform (which processor is connected to which bus) can be provided as the configuration of the system. From such a configuration, the POOSL model of the system can be automatically generated and fed to SHESim or Rotalumis tools for analysis. As an example, for the system in fig. 6a, the specification of the necessary patterns may look like the one in fig. 6b, and the structure of the generated model is shown in fig. 6c.

For design space exploration, different configurations must be compared. To do this, changes in the initial configuration may be done and the POOSL model regenerated in order to analyse them. To specify a different mapping, the `Map` specifications must be changed according to the new task-to-resource mapping. To change the architecture components, simply change the `Resource` specifications and/or their parameters. Similarly, the layout of the platform can be changed in the `Connection` specification tags. In this way, the model can be easily tuned to specify different possibilities in the design space without any knowledge about the underlying formal model that will be generated in accordance with the description of the new configuration.

<<process>>
Scheduling
<<parameters>>
Scheduler:SchedulingPolicy
<<methods>>
Init()() Schedule()()
<<messages>>
fromTask?schedule toResource!execute fromResource?stopped toResource!preemption toTask!executed

Figure 3: UML scheduling pattern

<<process>>
Resource
<<parameters>>
initialLatency:Real throughput:Integer
<<methods>>
Init()() ResourceRun()()
<<messages>>
sch!stopped sch?execute sch?preemption

Figure 4: UML resource pattern

5 Model Analysis

By composing together the necessary modelling patterns, the complete model of a system can be built and validated. For each configuration specified and generated, during the execution of the model, the scheduler can report if there are any tasks that miss their deadlines. Furthermore, based on the POOSL semantics derived from CCS, it can be detected if there is any deadlock in the system. If all the deadlines are met and there is no deadlock, then the corresponding architecture is a good candidate that meets the system requirements.

<<process>>
Environment
<<parameters>>
Events:Distribution u:Integer l:Integer
<<methods>>
Environment()()
<<messages>>
out!event

Figure 5: UML environment pattern

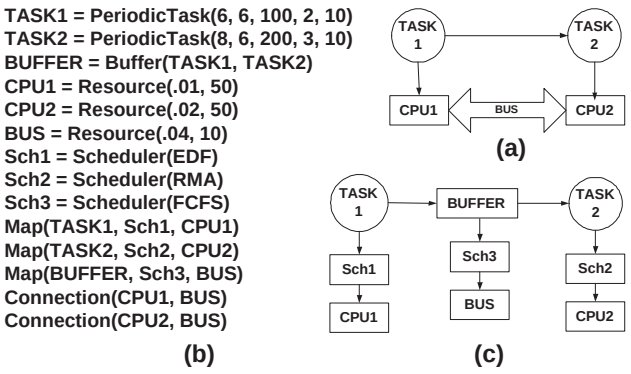

Figure 6: Use of patterns

However, for soft real-time systems, it is allowed that some deadlines are missed (usually there is a requirement for an upper limit). Therefore, in this case, it is especially useful that the analysis of the model can handle and record tasks with multiple active instantiations that have missed their deadlines. The percentage of deadlines missed can be monitored and checked against the requirements if, according to this criterion, the underlying platform is suitable.

To correctly dimension a system (the required CPUs performance and buses) such that it works in any situation, the worst-case behaviour of the system must be analysed. This usually means to consider the worst-case execution times for all the activities in the system. On the other hand, the analysis of the average behaviour, based on probabilities, which can be enabled in the proposed patterns, as shown in [FdHVC06], gives a measure of the suitability of the design. If the dimension of the system, needed for the worst-case situation that appears only once in a while, is far bigger than the one needed in average, that could give useful hints for a re-design (e.g. split tasks into smaller ones in order to spread the load onto different CPUs).

Some other useful results the analysis of the proposed model can provide are the release jitter, the output jitter and the number of instances of a task active at the same time.

6 Case Studies

In this section, two case studies are presented for which worst-case analysis and design space exploration have

been performed using the modelling patterns proposed in this work. The characteristics of the systems and the results of their analysis follow.

6.1 A Printer Paper-Path

The first case study is inspired by a system architecture exploration for the control of the paper-path of a printer.

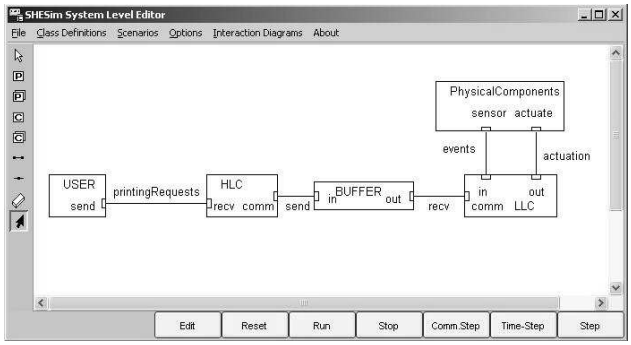

Figure 7: High-level printer control POOSL model

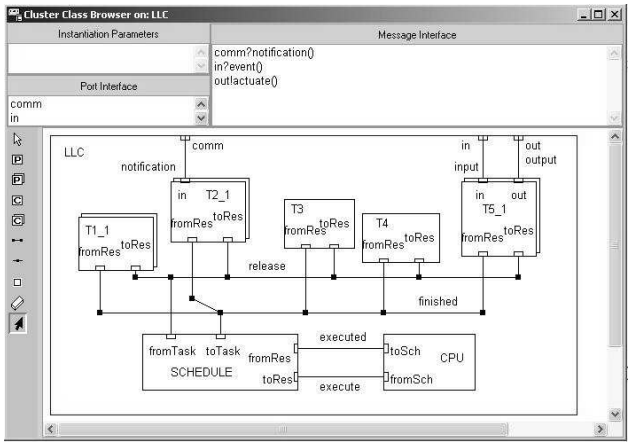

Figure 8: POOSL LLC model

The high-level view of the system model, visualised using SHESim tool, is given in fig. 7. User's printing requests arrive at the high-level control (HLC) of the machine which computes which activities need to take place and when in order to accomplish the request. The HLC tasks activate the tasks representing the low-level control (LLC) of the physical components of the paper path, like motors, sensors and actuators. As HLC tasks are soft real-time, whereas LLC tasks (fig. 8) are hard real-time, a rather natural solution was to consider a distributed architecture. LLC can be assigned to dedicated processor(s) and connected through a network to the general-purpose processor that runs HLC.

Under these circumstances, the *problem* was mainly *to find an economical architecture for LLC*, whose task parameters are shown in table 2. For the models of the time-driven tasks of type T1, T3 and T4, we took into account a latency of upto 10% of their period. Although tasks of type T2 are activated based on notifications from HLC, they behave completely periodic until the next notification arrives. Therefore, their dynamical behaviour was captured using an aperiodic task which triggers a periodic task with a finite number of activations. Tasks of type T5 are event-driven; therefore, a model of the environment was needed (*PhysicalComponents*), for which we considered event streams with a uniform distribution in $[1, 20]$ ms.

Given the frequency of events and the task execution times, we have analysed three commercially available low-end processors, a 40 MIPS, a 20 MIPS and a 10 MIPS, and compared their utilisations under different schedulers. Fig. 9 presents the results obtained using the earliest deadline first scheduling algorithm. Although the 10 MIPS processor seems to be used the most efficiently (close to its maximum capacity), the analysis of the model showed that some of the deadlines are missed; thus this processor is not a good candidate. For the other two, all deadlines are met and there were no deadlocks detected in the system. Due to the fast execution engine Rotalumis, tens of hours of system behaviour could be covered in less than one minute simulation. Moreover, the analysis of the model gave the values of the maximum release jitter, respectively output jitter of the tasks (for the 20 MIPS they are shown in table 3) which could be checked against the expected margins of errors of the environment control design.

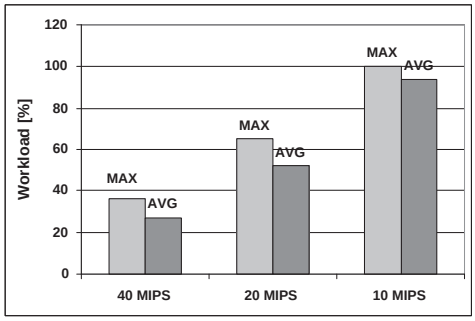

Figure 9: CPU workload comparison

Table 2: LLC task parameters

Task type	No. of Instantiations	Load	T (ms)	D (ms)
T1	3	3200	2	2
T2	8	1200	2	2
T3	1	2000	2	2
T4	3	800	0.66	0.1
T5	4	160	-	0.064

Table 3: Tasks jitter for the 20 MIPS

Task type	Release jitter (ms)	Output jitter (ms)
T1	0.466	1.852
T2	0.466	1.852
T3	0.414	1.884
T4	0.042	0.128
T5	0.472	1.094

6.2 An In-Car Navigation System

The second case study is inspired by a distributed in-car navigation system [Nav]. The high-level view of the system is presented in fig. 10. There are three clus-

ters of functionality, as the picture suggests: the man-machine interface (MMI) that handles the interaction with the user; the navigation functionality (NAV) that deals with route-planning and navigation guidance; the radio (RAD) which is responsible for basic tuner and volume control, as well as receiving traffic information from the network.

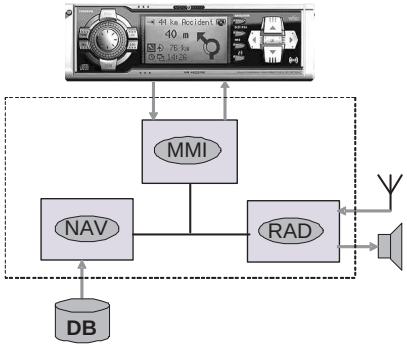

Figure 10: In-car navigation system

In [Nav], three application scenarios are described. Users are allowed to change the volume (scenario I) and to look addresses up in the maps in order to plan their routes (scenario II); moreover, the system needs to handle the navigation messages received from the network (scenario III). Each of these scenarios has its own individual timeliness requirements that need to be satisfied. They all share the same platform, however, not all three of them can run in parallel due to the characteristics of the system (only I with III, or II with III). The characteristics of tasks for each scenario are given in table 4. They are all periodic tasks with infinite behaviour. Notice that, in comparison with the previous case study, the timing requirements of this system are in the seconds domain and the loads imposed on the resources are much larger, as this case study combines control with data streaming.

The *problem* related to this system was *to find suitable platform candidates* that meet all the timing requirements of the application. For exploration of the design space, a few already available platforms (see fig. 11) were proposed for analysis. The end-to-end delay for each scenario on each platform, in the presence or absence of other scenarios, was monitored. The analysis shows that all the timing requirements are met for all scenarios in all configurations. Fig. 12 shows, as an illustration, the maximum end-to-end delay obtained for scenario III in isolation on each of the proposed platforms.

Furthermore, the processor(s) and bus(es) utilisations were monitored and, as an example, table 5 shows the results obtained for architecture `A`. All together, such results help the designer in detecting if there is any scenario likely to miss its deadline, or which processor or bus might be a bottleneck, and make decisions with respect to an appropriate platform to choose.

Due to the easiness of using the patterns and going to different configurations in the design space by just changing their parameters, the construction of models for each of the proposed combinations took several minutes. Moreover, as mentioned for the previous case study as well, due to Rotalumis, the engine for the model execution, the analysis results could be obtained also fast.

Table 4: In-car navigation systems tasks

Scenario	Task	Load	T (s)	D (s)
I	T1	1E5	1/32	1/32
	T2	1E5	1/32	1/32
	T3	5E5	1/32	1/32
II	T4	1E5	1	1
	T5	5E6	1	1
	T6	5E5	1	1
III	T7	1E6	3	3
	T8	5E6	3	3
	T9	5E5	30	30

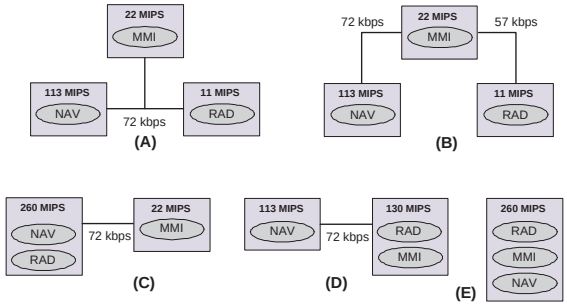

Figure 11: Platforms proposed for analysis

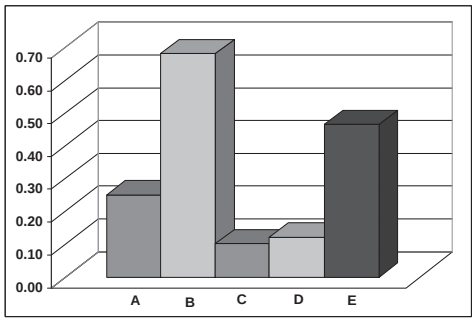

Figure 12: Maximum end-to-end delay for scenario III

7 Conclusions

In this paper, we have presented modelling patterns, specified using the Parallel Object-Oriented Specification Language, for the design space exploration of real-time embedded systems. These patterns allow easy composition of system models consisting of real-time tasks, computation and communication resources and their associated schedulers. Due to the expressiveness of POOSL, important aspects, like task activation latencies and context switches, can be taken into account, enabling the building of realistic models without sacrificing their conciseness. Moreover, due to this reason, the analysis can provide more realistic results than the classical scheduling techniques can.

The use of the patterns presented in this paper reduces both the modelling and the analysis effort. Although completeness cannot be claimed, the efficiency

Table 5: Processors and bus utilisations in arch. A

Scen. I	Scen. II	Scen. III	MMI %	NAV %	RAD %	Bus %
YES	NO	NO	87	0	30	3
NO	YES	NO	3	5	0	1
NO	NO	YES	1	2	4	1
YES	NO	YES	88	2	33	4
NO	YES	YES	4	6	2	2

of the model simulation allows exploration of a substantial part of the design space. As future work, we aim at extending the modelling patterns to cover for complex platforms like networks-on-chip, by taking into account memory components, routing algorithms and even batteries for the analysis of energy consumption.

References

[BMIS04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[Dou02] Bruce Powell Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[FdHVC06] Oana Florescu, Menno de Hoon, Jeroen Voeten, and Henk Corporaal. Probabilistic modelling and evaluation of soft real-time embedded systems. In *Proceedings of SAMOS VI*, LNCS 4017, 2006.

[FVC06] Oana Florescu, Jeroen Voeten, and Henk Corporaal. Modelling patterns for analysis and design of real-time systems. Technical Report ESR-2006-05, 2006.

[Gei02] Marc G.W. Geilen. *Formal Techniques for Verification of Complex Real-Time Systems*. PhD thesis, Eindhoven University of Technology, 2002.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[GJN99] M. Gries, J. Janneck, and M. Naedele. Reusing design experience for petri nets through patterns. In *Proc. of High Performance Computing 1999*, 1999.

[Gri04] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integration*, 38(2):131–183, 2004.

[KDVvdW97] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter van der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *Proceedings of the IEEE ASAP*, 1997.

[LS91] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[LvdWVD01] Paul Lieverse, Pieter van der Wolf, Kees Vissers, and Ed Deprettere. A methodology for architecture exploration of heterogeneous signal processing systems. *VLSI Signal Processing Systems*, 29(3):197–207, 2001.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Nav] `http://www.mpa.ethz.ch/`.

[OMG03] OMG. *Unified Modeling Language (UML) - Version 1.5*. OMG document formal/2003-03-01, Needham MA, 2003.

[PHL+01] Andy D. Pimentel, Louis O. Hertzberger, Paul Lieverse, Pieter van der Wolf, and Ed F. Deprettere. Exploring embedded-systems architectures with Artemis. *Computer*, 34(11):57–63, 2001.

[The04] Bart D. Theelen. *Performance Modelling for System-Level Design*. PhD thesis, Eindhoven University of Technology, 2004.

[TOO] `http://www.es.ele.tue.nl/poosl`.

[vdPV97] Piet H.A. van der Putten and Jeroen P.M. Voeten. *Specification of Reactive Hardware/Software Systems*. PhD thesis, Eindhoven University of Technology, 1997.

[WTVL05] Ernesto Wandeler, Lothar Thiele, Marcel Verhoef, and Paul Lieverse. System architecture evaluation using Modular Performance Analysis - A case study. 2005. Accepted for publication in the STTT Journal.