Performance Modelling and Analysis Using POOSL for an In-Car Navigation System*

Oana Florescu[1] Menno de Hoon[2] Jeroen Voeten[1,3] Henk Corporaal[1]

[1]Eindhoven University of Technology, [2]Chess Information Technology BV,
[3] Embedded Systems Institute

o.florescu@tue.nl menno.de.hoon@chess.nl j.p.m.voeten@tue.nl h.corporaal@tue.nl

Keywords: soft real-time systems, simulation-based estimations, worst case and average case analysis

Abstract

To ensure quality and performance of soft real-time embedded systems, the evaluation of their properties is needed from early phases of the design. These systems typically allow a certain rate of deadline misses. However, as they are often analysed using hard real-time techniques, which determine hard bounds of their performance properties, they are over-dimensioned and thus expensive. Using a case study inspired by industrial practice, we present how to compose a suitable model for soft real-time systems based on the formally defined modelling language POOSL. By means of simulations for different usage scenarios, evaluation of the timing properties of the system is provided. Furthermore, we compare our results with two other performance modelling techniques, which are based on analytical computation, showing that our approach leads to a more appropriate dimensioning of soft real-time systems.

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.

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.

*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.

Related Research. An extensive overview of system-level design space exploration methodologies is given in [1] and [2]. They range from analytical computation (Modular Performance Analysis (MPA) [3], UPPAAL [4]) to simulation-based estimation (Spade [5], Artemis [6]). The techniques for analytically computing the performance of a system are exhaustive in the sense that *all* possible behaviours of the system are taken into account. Therefore, a practical problem of them is that they are not usually applicable to industrial systems because of the state space explosion problem. On the other hand, simulation of models allows investigation of a *limited* number of all the possible behaviours of the system. Consequently, the obtained analysis results are *estimates* of the real performance of the system.

To obtain credible results, both techniques require that the models created are amenable to mathematical analysis (see [7]), using mathematical structures like Real-Time Calculus [8] or timed automata [9]. As in general analytical approaches do not scale with the complexity of the industrial systems, simulation-based estimation of performance properties is used more often. In this context, the estimation of performance is based on statistical analysis of simulation results.

In the past, performance analysis techniques were applied mainly in the design of hard real-time systems. However, the higher demands on the quality of products require such techniques also for soft real-time systems, like DVD players for the synchronisation of the audio and video stream decoding, or printers for the accuracy of printing an image on a sheet. Often, in the analysis of such systems hard bounds of their performance properties are determined because the techniques available are not suitable enough. Consequently, the resulting system is over-dimensioned. Nevertheless, as long as no human life is endangered by the incorrect behaviour of a system, the timing re-

quirements are not critical factors. Therefore, instead of having all the deadlines met, one should be able to reason about the rate of deadlines misses which is allowed in soft real-time systems.

Contributions of this paper. In this paper, we show how a formally defined modelling language, called POOSL, can be used to model and explore the design space of a soft real-time system. Moreover, we show how the models can incorporate knowledge about the varying timing behaviour of the system. The analysis approach proposed for the evaluation of the timing properties is not restricted to any scheduling policy and provides results based on simulations which are run until the desired accuracy is obtained. By the means of a case study, we compare the results of our analysis approach with two other analytical techniques. We show that our method leads to a better dimensioning of the system. We reduced with more than 50% the required capacity of two processors from the architecture found suitable by the other approaches.

The paper is organised as follows. The case study is presented in sec. 2. In sec. 3, a short presentation of the modelling language is given besides the description of the modelling approach for this system, whereas in sec. 4 the results of the performance analysis are presented. Conclusions are drawn in sec. 5.

2 The In-Car Navigation System

The high-level view of an in-car navigation system, considered as case study in this paper, is presented in fig. 1. There are three clusters of functionality: the man-machine interface (MMI) handles the interaction with the user; the navigation functionality (NAV) deals with route-planning and navigation guidance; the radio (RAD) is responsible for basic tuner and volume control, as well as receiving traffic information from the network. For this system, three application scenarios are possible. Users are allowed to change the volume (ChangeVolume scenario which is also shown in the UML diagram in fig. 2) and to look addresses up in the maps in order to plan their routes (ChangeAddr scenario). Moreover, the system needs to handle timely the navigation messages received from the network (HandleTMC scenario). All scenarios share the same platform, however, ChangeVolume and ChangeAddr cannot run in parallel because they need the same knob for the selection of the volume level or of the route in a list. A detailed description of the system and its scenarios can be found in [3].

Each of the scenarios has its own individual timeliness requirements that need to be satisfied. These requirements are specified in the UML diagram accompanying each scenario. For ChangeVolume, they can be seen on the left side of fig. 2. However, these requirements are not hard, as soft real-time systems typically allow a certain percentage of missed timing requirements. In this case, a rate of 5% misses is considered affordable.

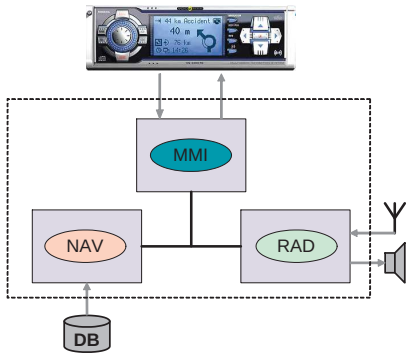

Figure 1: In-car navigation system

The *problem* related to this system was *to find suitable platform candidates* that meet the timing requirements of the application. For exploration of the design space, a few available platforms, presented in fig. 3, were analysed. Two approaches have been applied for the analysis of this system, Modular Performance Analysis (MPA) in [3] and UPPAAL in [10].

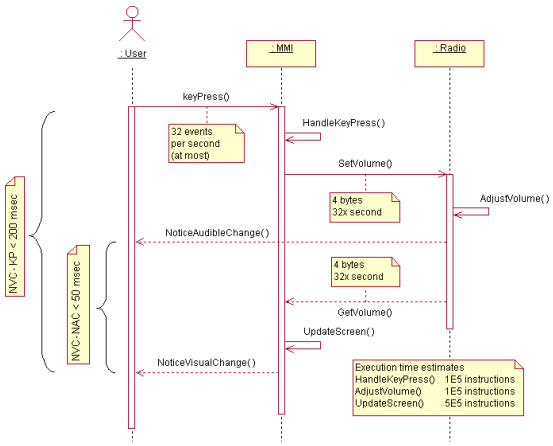

Figure 2: ChangeVolume scenario

MPA is an analytical technique in which the functionality of a system is characterised at a high level of abstraction by the incoming and outgoing event rates, message sizes and execution times. Based on Real-Time Calculus, hard upper and lower bounds of the system performance are computed. While these bounds are always hard, they are in general not exact, meaning that they are larger/smaller than the *actual* worst/best case. Thus, the analysis performed is conservative.

The UPPAAL model checker is a tool for modelling and verifying networks of timed automata. The analysis results obtained by applying this technique are exact computations of the performance properties. Nevertheless, the method suffers severely from the state space explosion problem. Limitations, stat-

ing for example that tasks can be preempted only up to a certain number of times, are necessary, otherwise model checking is not possible anymore. Moreover, combination of scenarios with large difference in the time scale of the requirements (milliseconds *vs*. seconds) proved to be another problem for the model checker.

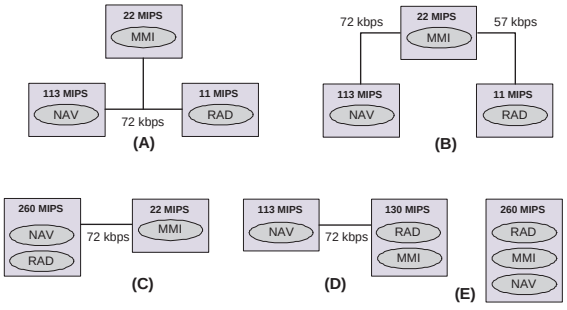

Figure 3: Platforms proposed for analysis

In this paper, we show how the in-car navigation system can be modelled appropriately based on the expressivity of POOSL modelling language. Furthermore, we show how the analysis results of the POOSL model compare with the other two techniques and how to dimension the system based on them.

3 Model of the System

One of the approaches for performing systematic design space exploration is the Y-chart scheme (fig. 4), introduced in [11]. This scheme makes a distinction between applications (the required functional behaviour) and platforms (the infrastructure used to perform this functional behaviour). We have added to this scheme the model of the environment connected to the application that controls it. The design space can be explored by evaluating different mappings of applications onto platforms.

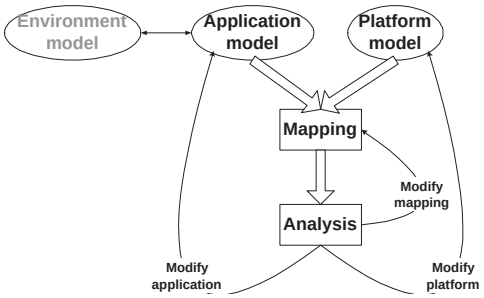

Figure 4: Y-chart scheme

In the following, first a brief introduction of the modelling language is provided. Afterwards, it will be explained how the model of the in-car navigation system was built according to the Y-chart scheme.

3.1 POOSL Modelling Language

To support the design and development of real-time systems, the system-level design method Software/Hardware Engineering (SHE) [12] provides a well-founded and expressive language, Parallel Object-Oriented Specification Language (POOSL) [13], for modelling and analysis of complex real-time systems. SHE uses a UML profile to formulate the concepts needed for the realisation of the requested functionality of a system. POOSL formalises the behaviour specified in informal UML diagrams, establishing a formal model.

POOSL contains a set of powerful primitives to formally describe concurrency, probabilistic distribution, communication, timing and functional features of a system into a single executable model. Its formal semantics is based on timed probabilistic labelled transition systems [14]. 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.

The SHE method is accompanied by two simulation tools, SHESim and Rotalumis. SHESim is a graphical environment intended for incremental specification, modification and validation of POOSL models. Rotalumis is a high-speed simulator, enabling fast evaluation of system properties. Compared with SHESim, Rotalumis improves the simulation speed by a factor of 100 by compiling the model into an intermediate format before executing it. Both tools have been proved to correctly simulate a model with respect to the formal semantics of the language ([15]).

3.2 Application Model

The application part of a system can be modelled in terms of tasks that communicate with each other in order to accomplish the system desired behaviour. The tasks are characterised by their parameters, the events they are triggered by and the tasks that they trigger in their turn. The parameters of a task are:

- **Computation load**: represents the number of instructions needed to perform the functionality of the task.
- **Task identifier**: a unique identifier in the application task graph, which can be a name or a number, and that is used to keep track of the order in which tasks are executed.
- **Priority / relative deadline**: is used for scheduling tasks according to the policy chosen for the resource.

An example of a POOSL model for a task is given in fig. 5. HANDLEKEYPRESS task (visualised in the UML diagram from fig. 2) is modelled as a recursive process class method which enables the task to receive any incoming event from the knob that is turned by the user. Each event is handled by the EXECUTE method, which sends (as a message) a request for execution to the platform. The parameters of the task

are also sent along with the message. When the request is accomplished, the task is returned a message `executed` and it then triggers another task in the system by sending a message `trigger`.

```
1  HANDLEKEYPRESS()() | E : Event |
2  in?event(E);
3  par
4    HANDLEKEYPRESS()()
5  and
6    EXECUTE()();
7    out!trigger
8  rap.

9  EXECUTE()() | Task : Integer |
10 platform!execute(TaskId, ComputationLoad, Priority);
11 platform?executed(Task | Task = TaskId).
```

Figure 5: HANDLEKEYPRESS task model

Besides the tasks that accomplish the required functionality of the system, we have also taken into account the model of the buffers for the communication between tasks that are mapped onto different resources. Buffers are considered "communication tasks" that are "executed" by communication resources, like buses. Their execution requests are in fact message transfer requests. The model of a buffer is similar with the task described in fig. 5, except for the meaning of the load, which in this case represents the size of the message, and the lack of the priority parameter.

3.3 Platform Model

The platform on which the software runs is a collection of resources that perform the computations and the communication required by the application. As resources are shared by several tasks, a scheduler needs to be modelled for the arbitration (based on a certain scheduling policy) of the access. To model a resource with priority-based scheduling policy, we have used the specification in fig. 6. CPU is the initialisation method of a process class modelling a computation resource. It is recursive which makes the handling of new computation requests possible when the resource is idle. After receiving a task, which has a specific priority and computation load, the EXECUTETASK method is called. The execution of the computation is modelled in line 8 by the *delay* statement. While the computation is being performed, it can be preempted by a request coming from another task with a higher priority. This is modelled by the *interrupt* statement. The EXECUTETASK method is called again recursively to handle the higher priority task. When the computation associated to a task is finished, EXECUTETASK returns a `task!executed` message to the corresponding task. Once the recursiveness of EXECUTETASK is finished, the resource enters the idle mode and method CPU is called again.

The model presented in fig. 6 can easily be changed to use another scheduling policy. For example, to model the earliest deadline first algorithm, it is enough to replace lines 11 and 12 with the ones given in fig. 7.

```
1  CPU()()
2  task?execute(Task, ComputationLoad, Priority);
3  EXECUTETASK(Task, ComputationLoad, Priority)();
4  CPU()().

5  EXECUTETASK(ServingTask : Integer,
6    ServingLoad : Real, ServingPriority : Integer)()
7   | ReqTask, ReqPriority : Integer, ReqLoad : Real |
8  interrupt
9   delay ServingLoad / MIPS
10 with
11   (task?execute(ReqTask, ReqLoad,
12       ReqPriority | ReqPriority > ServingPriority);
12   EXECUTETASK(ReqTask, ReqLoad, ReqPriority)() );
13 task!executed(ServingTask).
```

Figure 6: CPU model

```
11  task?execute(ReqTask, ReqLoad,
12      ReqDeadline | ReqDeadline > ServingDeadline);
```

Figure 7: Change of CPU for EDF model

For the communication resource, we have modelled a point-to-point communication bus, which was needed in the case study, shown in fig. 8. This type of resource has a First Come First Serve (FCFS) scheduling policy. Method BUS receives requests from the buffers that interconnect tasks at the application level. TRANSFERMSG method models the transfer of the message through the communication resource by the *delay* statement. The transfer time depends on the size of the message and the bandwidth of the resource.

```
1  BUS()() | MessageId, MessageSize : Integer |
2  msg?transfer(MessageId, MessageSize);
3  TRANSFERMSG(MessageSize)();
4  msg!transferred(MessageId);
5  BUS()().

6  TRANSFERMSG(MessageSize : Integer)()
7  delay MessageSize / Bandwidth.
```

Figure 8: BUS model

3.4 Environment Model

To reason properly about the properties of an embedded system, its whole behaviour should be modelled realistically, including the environment that triggers the events. For this purpose, a discrete-event approximation of the continuous-time behaviour of the physical components can be modelled in terms of event streams that occur according to some arrival patterns.

As an example of such an arrival pattern, fig. 10 illustrates the model of a periodic event stream with jitter. Such an arrival pattern is usually met in real-time systems, although most of the analysis techniques assume in such situation perfect periodicity of events.

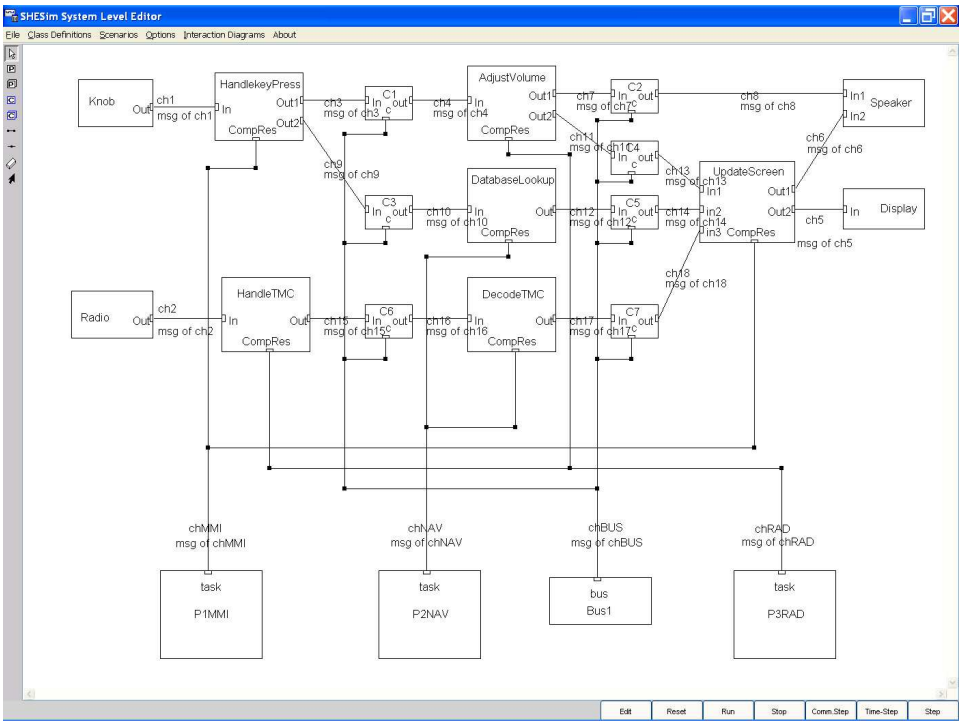

Figure 9: POOSL model of the system

The parameters of the model are the period T of the event occurrence and the maximum value of the jitter J. The event is modelled to occur anywhere between T-J and T+J. For this, *jitter* is a data object of uniform distribution type whose limits are 0 and J/2. At every period, a new *sample* j is generated from this distribution and $2 * j$ units of time pass by before the event produced arrives at the application.

```
1  INIT()()
2  jitter = new(Distribution)ofType(Uniform);
3  jitter withParameters(0, J/2);
4  KNOBEVENT()().

5  KNOBEVENT()()
6  par
7    delay T - J;
8    j:= 2 * jitter sample();
9    delay j;
10   out!event(E)
11 and
12   delay T;
13   KNOBEVENT()()
14 rap.
```

Figure 10: Model of a periodic event with jitter

3.5 Mapping Model

For the mapping stage of the Y-chart in fig. 4, the proposed modelling approach uses communication channels provided by POOSL to map the application components to the resource components. A message channel is created between a task and the resource on which it is mapped, as shown in fig. 11. Moreover, if there is communication between tasks mapped onto different CPUs, then the buffer in between is mapped onto the communication link.

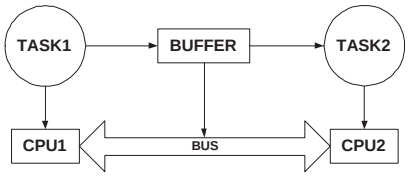

Figure 11: Mapping stage of Y-chart

By using the modelling approach presented above, the complete model of the in-car navigation system was built and a snapshot from the graphical tool is presented in fig. 9.

4 Performance Analysis of the System

A model built according to the Y-chart scheme allows analysis of different application-platform configurations, as all the components modelled are parameterisable and thus easy to be changed. To evaluate a certain configuration, during model simulation, the scheduler reports if there are tasks that miss their deadlines. Furthermore, based on the POOSL semantics, it can be detected if there is a deadlock in the system. If all the deadlines are met and there is no deadlock during the simulation, then the corresponding platform is a *good* candidate to meet all the system requirements, although there is no guarantee for

Table 1: Timeliness requirements of the system

Scenario name	Deadline [*ms*]	Task name	Load [*instructions*]	f [*1/s*]
ChangeVolume	200	HandleKeyPress	1E5	32
		AdjustVolume	1E5	32
		UpdateScreen	5E5	32
ChangeAddr	200	HandleKeyPress	1E5	1
		DatabaseLookup	5E6	1
		UpdateScreen	5E5	1
HandleTMC	1000	ReceiveTMC	1E6	1/3
		DecodeTMC	5E6	1/3
		UpdateScreen	5E5	1/30

that as simulation completeness cannot be claimed. However, for soft real-time systems, it is allowed that a certain percentage of deadlines are missed. Therefore, in this case, it is especially useful to keep track of the rate of deadlines missed and check if the underlying platform meets the requirements.

In the following subsections, both worst case and average case evaluation of the in-car navigation system are presented.

4.1 Worst Case Analysis

The UML diagrams specifying the case study, as the one shown in fig. 2, provide the worst case values of the load (number of instructions) imposed by tasks on the CPUs. They also specify what is the rate of task activations (how often the events are triggered) which depends on the scenario in which they appear. Based on these activation rates, priorities were assigned to tasks according to the rate monotonic approach. The timing requirements of the system are also specified in the UML diagrams as end-to-end deadlines for each scenario. The loads of the tasks, the frequencies (f) of activations[1] per scenario and the timing requirements are given in table 1.

By simulating[2] the behaviour of the system, using each of the proposed architectures in fig. 3, the end-to-end delays were monitored. Fig. 12 shows, as an illustration, the maximum end-to-end delay obtained for HandleTMC scenario when running alone on each of the proposed platforms (from A to E).

The most interesting situations to monitor were the ones in which two scenarios are running in parallel as such a situation can lead to a larger value for the end-to-end delay. In our simulation, we have observed that all the deadlines are met on all the architectures. As an example, the results obtained for different combinations of scenarios on architecture A are presented in table 2. Next to them the results obtained using MPA and UPPAAL techniques are also provided. Architecture A was chosen for further discussion because it was the one chosen for deeper analysis by both techniques.

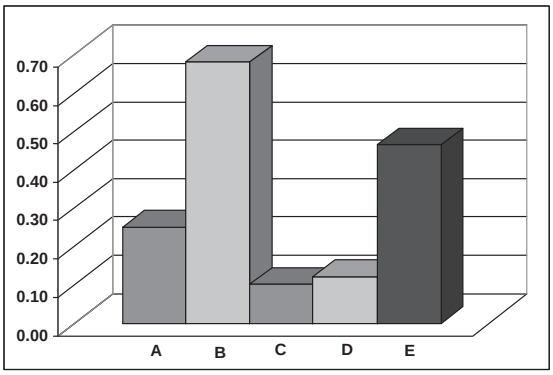

Figure 12: Maximum end-to-end delay for scenario HandleTMC

MPA is an analysis technique which finds hard upper bounds, not necessarily the actual worst case reached by the model. This explains the larger values that are obtained by applying this method. On the other hand, the results computed by UPPAAL are exact values of the worst case end-to-end delay. It is interesting to observe that our results are very close to UPPAAL ($\sim$1% difference which also represents the accuracy of the results), except for HandleTMC scenario for which the difference is 7%. For this situation we suspect a miss-match between the corresponding models and this aspect is still under investigation.

Besides keeping track of the end-to-end delays, during simulation, we have also monitored the resources utilisation. For architecture A, the obtained results are presented in table 3. Based on the amount of idle time of the CPUs and on the fact that the worst case values of the delays are much smaller than the specified deadlines, we concluded that the performance of the underlying architecture could be re-

[1]Tasks are triggered by the events in the environment as knob turning or messages from the network. In this analysis, the events are assumed to arrive periodic, so the value of the maximum jitter J in the environmental model is 0.

[2]Note that, the simulation was run with the fast execution engine Rotalumis; thus, a few minutes of system simulation represent several hours of runtime behaviour. The simulation was run until an accuracy of 99% of the results was reached.

Table 2: Architecture **A** worst case end-to-end delays

Measured scenario	Other active scenario	POOSL [*ms*]	MPA [*ms*]	UPPAAL [*ms*]
ChangeVolume	HandleTMC	41.771	42.2424	41.796
HandleTMC	ChangeVolume	357.81	390.086	381.632
ChangeAddr	HandleTMC	78.89	84.066	79.075
HandleTMC	ChangeAddr	171.77	265.849	172.106

Table 3: Resources utilisations in architecture **A**

Scenario ChangeVolume	Scenario ChangeAddr	Scenario HandleTMC	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

duced in order to have a platform with less cost and energy consumption.

4.2 Average Case Analysis

For a more appropriate dimensioning of the system, we have modelled more realistically the behaviour of the in-car navigation system, in which we have assumed that the tasks loads vary according to a uniform distribution. In order to do this, we modelled the *ComputationLoad* parameter of a task as a data object of a uniform distribution type, from which, at every task invocation, the *current* load of the task is obtained. As the UML diagrams provide only the worst case value of the load of each task, based on previous knowledge about similar systems, we have assumed that the actual load varies between 75% and 100% of the value provided. The limits of the load variation for each task is given in table 4. However, such a model involving probabilistic distributions is possible in neither MPA nor UPPAAL. Therefore, the analysis results obtained could not be compared with them.

Table 4: Tasks loads limits

Task name	Lower limit	Upper limit
HandleKeyPress	7.5E4	1E5
AdjustVolume	7.5E4	1E5
UpdateScreen	3.75E5	5E5
DatabaseLookup	3.75E6	5E6
HandleTMC	7.5E5	1E6
DecodeTMC	3.75E6	5E6

By simulating the system under the new circumstances, the end-to-end delays were monitored again. Their values can be graphically plotted as a distribution histogram, showing on the horizontal axis the values of the end-to-end delay and on the vertical axis the rate of occurrence of each value. As an example, fig. 13 shows the distribution histogram of the delay for the HandleTMC scenario when it runs in parallel with ChangeVolume on architecture A. From such distribution histograms it can be seen which are the minimum (best case) and the maximum (worst case) values for the end-to-end delays. Table 5 shows these values for all the combinations of scenarios running on architecture A. Moreover, from distribution histograms it can be deduced how often the maximum value of the end-to-end delay occurs.

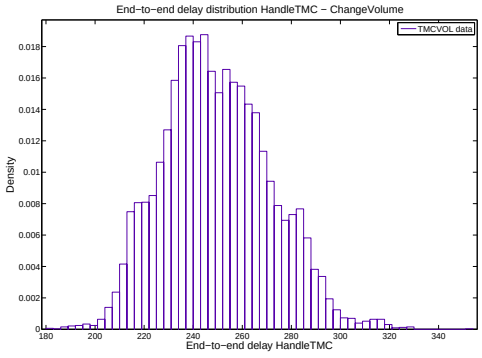

Figure 13: HandleTMC histogram on **A**

In our simulations, we have observed that the requirements are met for all the scenarios on all the proposed architectures and that the maximum delays are much smaller than the deadlines. Moreover, as the in-car navigation is a soft real-time system, it allows a certain rate of deadline misses, which is 5%. Based on this information and on the analysis of the idle time of the resources, in order to reduce cost, we decided to choose slower CPU cores for NAV and RAD on architecture A as they are not heavily used. For NAV, we have chosen a 40MIPS and for RAD a 5MIPS. The end-to-end delays obtained with this new configuration are shown in table 6. From the confidence inter-

vals calculated during simulation (see [7] for explanations), we observed that the rate of deadline misses is within 5%, thereby fulfilling the requirements.

Table 5: End-to-end delays on architecture **A**

Measured scenario	Active scenario	Minimum delay (ms)	Maximum delay (ms)
ChangeVolume	HandleTMC	28.17	47.82
HandleTMC	ChangeVolume	180.9	353.51
ChangeAddr	HandleTMC	61.08	127.51
HandleTMC	ChangeAddr	132.59	204.06

In this way, we have found a better dimensioning of the system than what was found using both MPA and UPPAAL. Using the proposed analysis approach, the performance of two CPUs of the platform found suitable by the other techniques could be reduced with 65% for NAV and with 55% for RAD.

Table 6: End-to-end delays on the improved **A**

Measured scenario	Active scenario	Minimum delay [*ms*]	Maximum delay [*ms*]
ChangeVolume	HandleTMC	42.84	58.48
HandleTMC	ChangeVolume	654.18	1056.06
ChangeAddr	HandleTMC	115.22	270.8
HandleTMC	ChangeAddr	298.27	496.03

5 Conclusions

In this paper, we have shown how POOSL modelling language can be used to build models suitable for soft real-time system. Based on simulations, we show how such a model can be analysed for worst case. For sanity-check, we also position our results against the results obtained for the same case study using two other performance analysis techniques which are based on analytical computations. The comparison confirms the accuracy of our simulation-based estimations.

Furthermore, we present how models can incorporate knowledge about the varying timing behaviour of the system using probabilistic distributions in order to analyse the system for average behaviour. Based on that, a better dimensioning of the system is possible. We show that the performance of the architecture found by the analytical techniques as suitable can be reduced with more than 50%, thus reducing cost.

As future work, we aim at applying this approach to larger and more complex systems in order to improve it and extend it.

Acknowledgments. The authors would like to thank Marcel Verhoef for his support and valuable comments to this work.

References

[1] 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.

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

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

[4] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A Tutorial on UPPAAL. In *Proc. of SFM*, 2004.

[5] 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.

[6] 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.

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

[8] Samarjit Chakraborty, Simon Kunzli, and Lothar Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. of DATE*, 2003.

[9] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2), April 1994.

[10] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures. In *Proc. of WPDRTS*, 2006.

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

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

[13] POOSL. http://www.es.ele.tue.nl/poosl.

[14] Jeroen Voeten. Performance evaluation with temporal rewards. *Perform. Eval.*, 50(2/3):189–218, 2002.

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