Probabilistic Modelling and Evaluation of Soft Real-Time Embedded Systems[⋆]

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

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

Abstract. Soft real-time systems are often analysed using hard real-time techniques, which are not suitable to take into account the deadline misses rate allowed in such systems. Therefore, the resulting system is over-dimensioned, thus expensive. To appropriately dimension soft real-time systems, adequate models, capturing their varying runtime behaviour, are needed. By using the concepts of a mathematically defined language, we provide a modelling approach based on patterns that are able to express the variations appearing in the system timing behaviour. Based on these modelling patterns, models can be easily created and are amenable to average case performance evaluation. By the means of a case study, we show the type of results that can be obtained from such an evaluation and how these results are used to dimension the system.

1 Introduction

Due to the high time-to-market constraint in the embedded systems industry, accompanied by increasing demand for more functionality and tighter requirements on cost, speed (throughput) and energy consumption of the final product, the industry has shifted its focus from improving the system *implementation* phase to improving the system *design* phase. To this end, early evaluation of system properties is needed to make correct decisions that guarantee the satisfaction of the *functional* and *non-functional* requirements. This is where design space exploration and system-level performance modelling techniques come into scene. In the past, such 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. As no suitable techniques are available, the timing requirements of such systems are treated as hard, and consequently, the resulting system is over-dimensioned. However, as these requirements are not critical factors, 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.

[⋆] 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.

Contributions of the paper. In this paper, we present an approach for *probabilistic modelling and evaluation* of soft real-time embedded systems. The approach is based on the concepts of a formally defined general-purpose modelling language, POOSL, which enables creation of models that describe systems behaviour using probabilistic distributions. Based on this, we developed a library of *probabilistic modelling patterns* to be used when composing models for design space exploration of soft real-time systems. These patterns act like templates that can be used in any situation, reducing the necessary modelling effort. Based on them, we can analyse the *varying* timing behaviour (average case analysis) of the system, instead of considering only its worst case. The analysis results expose the degree to what extent the requirements can be met by a certain system architecture and decisions can be made with respect to reducing the performance of the necessary architecture for lowering the cost and the energy consumption.

Related research. An extensive overview of performance modelling and analysis methodologies is given in [1] and [2]. They range from analytical computation (Modular Performance Analysis [3], UPPAAL [4]) to simulation-based estimation (Spade [5], Artemis [6]). The analytical computation techniques are exhaustive in the sense that *all* possible behaviours of the system are taken into account, whereas simulation of models allows the investigation of a *limited* number of behaviours. Thus, the obtained analysis results are *estimates* of the real performance of the system. For credibility of results, models created in both types of techniques need to be amenable to mathematical analysis (see [7]), using mathematical structures like Real-Time Calculus [8], timed automata [9] or Kahn process networks [10]. 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.

With respect to timing behaviour, an impressive amount of work has been carried out in the area of schedulability analysis (e.g. [11], [12], [13]) focussing on worst case. However, less work addresses the analysis of systems with probabilistic behaviour. For soft real-time systems, it is important to analyse the variations in the runtime behaviour to determine the likelihood of occurrence of certain undesired situations and, based on that, to dimension the system. In [14] and [7] it is showed that the techniques proposed in this area are quite restrictive. Some of them target certain application classes, being limited to uni-processor architectures or supporting only exponential distributions for expressing the probabilistic behaviour; other approaches address specific scheduling policies or assume highly-loaded systems. Overcoming these issues, the modelling approach presented in this paper can capture any kind of probabilistic distribution of system behaviour and any scheduling policy is allowed for the analysis of timing behaviour. Although the evaluation of the system properties is based on simulations, due to the formal semantics of the modelling language, the accuracy of the results can be determined.

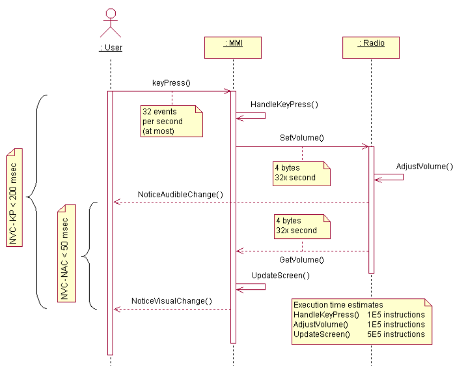
The paper is organised as follows. The case study used throughout the paper to illustrate various ideas is presented in section 2. In section 3, the modelling approach is described together with the modelling language used, whereas the performance analysis method is presented in section 4 next to the results obtained for the case study. Conclusions are drawn in section 5.

2 Case Study

The case study discussed in this paper is an in-car navigation system. The system has 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: the ChangeVolume scenario allows users to change the volume; the ChangeAddr scenario enables route planning by looking up addresses in the maps stored in the database; in the HandleTMC scenario the system needs to handle the navigation messages received from the network. Each of these scenarios is described by a UML message sequence diagram, like the one shown in fig. 1. A detailed description of the system and of its scenarios can be found in [3].

The *problem* related to this system was *to find suitable platform candidates* that meet the timing requirements of the application. To explore the design space, a few platforms, presented in fig. 2, were proposed and analysed using Modular Performance Analysis (MPA) in [3]. MPA is an analytical technique in which the functionality of a system is characterised 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. However, these bounds are in general not exact, meaning that they are larger/smaller than the *actual* worst/best case. Thus, the analysis performed is conservative.

As the in-car navigation is a soft real-time system that allows a certain percentage of deadline misses, it is doubtfully interesting to explore if there is an



Execution time estimates
HandleKeyPress() 1E5 instructions
AdjustVolume() 1E5 instructions
UpdateScreen() 5E5 instructions

Fig. 1. ChangeVolume scenario

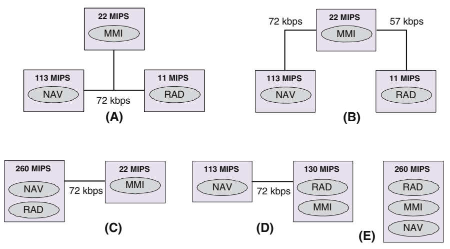


Fig. 2. Platforms proposed for analysis

architecture of lower cost and performance than what have been obtained with MPA that can still meet the timing requirements.

3 Modelling of the System

One of the approaches for performing systematic design space exploration is the Y-chart scheme introduced in [15]. This scheme makes a distinction between applications (the required functional behaviour) and platforms (the infrastructure used to perform this functional behaviour). The design space can be explored by evaluating different mappings of applications onto platforms. In the following subsections, first the modelling language POOSL is briefly presented and then the models of the application and of the platform are explained, whereas the environment model and the mapping are detailed in [16].

3.1 POOSL Modelling Language

The Parallel Object-Oriented Specification Language (POOSL) [17] lies at the core of the system-level design method called Software/Hardware Engineering (SHE). POOSL contains a set of powerful primitives to formally describe concurrency, probabilistic behaviour, (synchronous) communication, timing and functional features of a system into a single executable model. Its formal semantics is based on timed probabilistic labelled transition systems. This mathematical structure guarantees a unique and unambiguous interpretation of POOSL models. Hence, POOSL is suitable for specification and, subsequently, verification of correctness and analytical computation of performance for real-time systems. However, due to the state space explosion problem, simulation-based estimations are used for the evaluation of system properties.

The SHE method is accompanied by two simulation tools. 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. Both tools have been proved to correctly simulate a model with respect to the formal semantics of the language ([18]).

3.2 Application Model

The functional behaviour of a real-time embedded system is implemented through a number of tasks that communicate with each other. In our approach, they are modelled as POOSL process objects. Using the primitives of the language, any kind of real-time behaviour can be expressed (e.g. concurrency, communication, data computations).

As an example, the HandleKeyPress (visualised in the UML diagram in fig. 1) task model is presented in fig. 3. The activation of the task is triggered by an event (i.e. turning the knob by the user). The computations performed by the task, modelled by the method Computation, impose a certain *load* on a CPU and have a deadline D, modelled by the **delay** statement. When the deadline expires, or when the computation finishes (if it takes longer than D) the result

```
HANDLEKEYPRESS()()
    | E : Event, R : Results |
  /* a new event E is received */
  in?event(E);
  par
    par
      COMPUTATION(E)(R)
    and
      delay D
    rap;
    /* the result R is sent */
    out!result(R)
  and
    /* handle another event */
    HANDLEKEYPRESS()()
  rap.
```

Fig. 3. HandleKeyPress task model

```
SCHEDULE()() | req, oldreq : Request |
  sel
    task?schedule(req);
    req setCurrentLoad();
    SchPolicy scheduleRequest(req);
    if (SchPolicy hasHighestPriority(req) == true)
    then
      sel
        toResource!execute(req)
      or
        toResource!preemption;
        fromResource?stopped(oldreq);
        toResource!execute(req);
        SchPolicy update(oldreq)
      les
    fi;
    SCHEDULE()()
  or
    fromResource?stopped(oldreq);
    task!executed;
    req := SchPolicy removeRequest(oldreq);
    if (req != nil)
    then toResource!execute(req) fi;
    SCHEDULE()()
  les.
```

Fig. 4. Scheduler model

is sent as a message to another task. By recursively calling HandleKeyPress method in the **and** branch of the outer **par** statement, it is ensured that another available message can be immediately received.

The *deadline* and the *load* (expressed as the number of instructions to be executed by a CPU) represent the parameters of a real-time task. As the Computation performed by a task usually depends on the incoming event, the *load* is not a fixed value, but varies between a minimum and a maximum (best case and worst case). These parameters affect the scheduling of tasks on a platform.

3.3 Platform Model

The platform on which the software runs is described as a collection of computation and communication resources. As there is no large conceptual difference between them (they receive requests, execute them and send back notification on completion), we have conceived a single model for both types of resources.

As a resource is usually shared by a number of concurrent tasks, a scheduler is needed to arbitrate the access. The modelling pattern for a scheduler is given in fig. 4. The scheduler can either receive scheduling requests from newly activated tasks (the outer **sel** branch), or notifications from the platform about completed requests (the **or** branch). In case of a newly activated task, the *setCurrentLoad* method sets its current load according to a probabilistic distribution which appropriately captures the fluctuations in the task load. The data object *SchPolicy* is an instance of a data class implementing the actual scheduling algorithm. For specifying different policies, different subclasses can be defined. *Any* type of policy can be modelled (e.g. EDF, RMA, round-robin). An EDF scheduling

```
scheduleRequest(req : Request): SchPolicy
  | i, j : Integer |
 i := 1;
 while(req getDeadline() >
       list get(i) getDeadline()) do
   i:=i+1
 od;
 list insert(i, req);
 return self.
```

Fig. 5. EDF scheduling policy

```
RESOURCE()() | req: Request,
       loadLeft, tstart, tstop : Integer |
 sch?execute(req);
 delay initialLatency sample();
 tstart := currentTime;
 abort
   delay req getLoad() / throughput
 with sch?preemption;
 tstop := currentTime;
 loadLeft := req getLoad() -
        (tstop - tstart) * throughput;
 req setLoad(loadLeft);
 sch!stopped(req);
 RESOURCE()().
```

Fig. 6. Resource model

policy is given as an example in fig. 5. A list is kept with all the ready requests, and the new request *req* is inserted in this list based on its deadline value. For the requests completed by the resource, the scheduler checks if the deadline was missed and monitors the percentage of misses during simulation.

Fig. 6 presents the resource model as a POOSL process. The parameters of this modelling pattern are the *initialLatency*, which is due to task context switch time, in case of a CPU, and to the time to transfer the first bit of a message, in case of a bus, and the *throughput*. While *throughput* of a resource has a constant value, the *initialLatency* may vary due to diverse factors (e.g. cache). Therefore, we have modelled it is as a data object of some distribution type. Furthermore, we have enabled preemption of the execution of a request on a resource using the **abort** statement. Once finished or preempted, the remaining execution time of a request is computed and the request is sent back to the scheduler.

4 Average Case Performance Analysis

The modelling patterns presented in the previous section can be used to automatically generate a Y-chart-compliant model of a system. Different application-platform configurations can be specified and evaluated. During the simulation of such a model, the scheduler reports if there are deadline misses. 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 that meets all the system requirements, although simulation completeness cannot be claimed. However, for soft real-time systems, it is allowed that a certain percentage of deadlines are missed. Thus, in this case, it is useful to keep track of the rate of deadlines missed and check if the underlying platform meets the requirements. With the modelling approach presented above, the average case behaviour can be monitored and an appropriate dimensioning of the system can be made.

Task name	Min [*instr.*]	Max [*instr.*]
HandleKeyPress	7.5E4	1E5
AdjustVolume	7.5E4	1E5
UpdateScreen	3.75E5	5E5
DatabaseLookup	3.75E6	5E6
ReceiveTMC	7.5E5	1E6
DecodeTMC	3.75E6	5E6

Fig. 7. Tasks loads in the case study

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

Fig. 8. Timeliness requirements

4.1 Analysis Results for the Case Study

For the case study considered in this paper (the in-car navigation system) we have assumed that the loads of all tasks variate according to a uniform distribution, based on the inspiration got from measurements of similar systems. As the UML diagrams provide only the worst case value of the load of each task, we have considered that the actual load varies between 75% and 100% of the value provided. The limits of the load variation for each task are given in fig. 7. Based on the MIPS rate of the CPUs on the proposed architectures, given in fig. 2, we can compute the execution times of tasks. Depending on the scenario in which it is used, a task may be called at different rates. The frequencies of tasks activations per scenario are given in fig. 8. Based on these activation rates, priorities were assigned to tasks according to the rate monotonic approach. The timing requirements of the system are specified in the UML diagrams as end-to-end deadlines for each scenario, provided also in fig. 8.

During simulations[4] of the system behaviour for each of the architectures proposed in fig. 2, the end-to-end delays were monitored. The results obtained were graphically plotted as distribution histograms, 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 the parallel execution of two scenarios is likely to lead to more variation in the end-to-end delay, fig. 9 shows the distribution histogram for the HandleTMC scenario when it runs in parallel with ChangeVolume on architecture A. From such distribution histograms, the minimum (best case) and the maximum (worst case) values for the end-to-end delays can be deduced. Columns 3 and 4 in fig 11 show these values for all the combinations of scenarios running on architecture A. Moreover, the relative frequency of occurrence of the maximum value can also be deduced. During 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.

[4] By using the fast execution engine Rotalumis, 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.

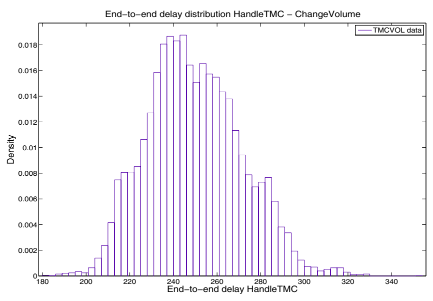


Fig. 9. HandleTMC distribution histogram on architecture A

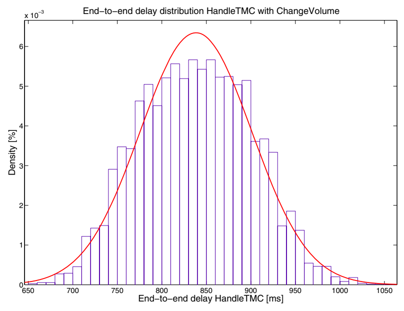


Fig. 10. Distribution fitted over the HandleTMC distribution histogram on the improved A

Measured scenario	Active scenario	Min. delay $[ms]$	Max. delay $[ms]$	Mean delay $[ms]$	Max. delay $[ms]$
ChangeVolume	HandleTMC	28.17	47.82	49.66	58.48
HandleTMC	ChangeVolume	180.9	353.51	838.32	1056.06
ChangeAddr	HandleTMC	61.08	127.51	134.12	270.8
HandleTMC	ChangeAddr	132.59	204.06	349.712	496.03

Fig. 11. End-to-end delays of all scenarios

4.2 Dimensioning of the System

The in-car navigation system is a soft real-time system that allows a rate of 5% of deadline misses. Based on this, together with the utilisation rates of the resources, which were also monitored during simulation, and the observed maximum values of the delays, one can reason about possible platform performance reduction in order to reduce cost and energy consumption of the system.

In [3], where this case study was analysed using MPA, the authors investigated the robustness of architecture A. Therefore, in this paper we have also focussed on this architecture to reason about its resources. The utilisation of **MMI** is 88%. As the periods and loads of the tasks mapped on this processor are quite heavy, there is not much room for the decrease of its capacity. The **NAV** processor is used 6%. The histograms of scenarios ChangeAddr and HandleTMC showed a difference of 80ms and 200ms respectively between the worst case delays obtained and the requirements. Hence, we reduced **NAV** capacity to 40MIPS. The utilisation of **RAD** is 33%. The analysis showed a difference of 100ms for ChangeAddr and 200ms for HandleTMC respectively between the maximum delays and the deadlines. As there is potential for capacity reduction, we reduce the capacity of this processor to 5MIPS.

With this new configuration for architecture A, we resumed our simulations using the same variances in the task loads and the same task priorities. The distribution histograms of the end-to-end delays were plotted and, as an example, fig. 10 shows the histogram for the HandleTMC scenario. The mean and maximum values of the end-to-end delays for all the scenarios are presented in columns

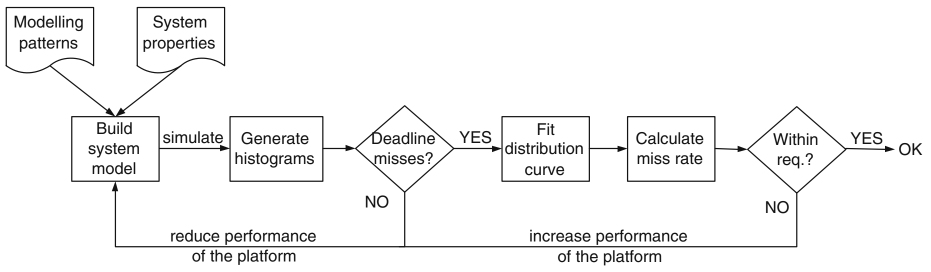


Fig. 12. Flow of the steps in the analysis approach

5 and 6 in fig. 11. From the confidence intervals calculated during simulation, we observed that the rate of deadline misses is within 5%, thereby fulfilling the requirements. In this way, we have found a better dimensioning of the system than what was found using MPA, reducing two of the processors with 65% (NAV) and respectively 55% (RAD).

Furthermore, in order to use such analysis results in an multi-disciplinary model of complex systems aiming at design trade-offs across disciplines, an abstraction of the timing behaviour of the software part is needed. To this end, we propose to fit the resulting distribution curves into known types of distribution. According to the *central limit theorem* in probability theory, due to the uniformly distributed loads of the tasks and to the fact that tasks in different scenarios are independent, the end-to-end delay of a scenario has approximately a normal distribution. Therefore, over the distribution histogram obtained from a simulation, a normal distribution curve is fitted. Fig. 10 shows such a curve fitted over the HandleTMC histogram. The parameters of the normal distribution are the mean value (μ) of 838.32 (ms) (the mean value of the delay) and the standard deviation (σ^2) of 3953.36 (ms). From such curves, the rate of deadline misses can be deduced, based on their characteristics. For example, the deadline for HandleTMC, which is 1000ms, can be found between two and three standard deviations from the mean. Thus, the probability of missing the deadline is less than 5%, which means the requirements are met. Furthermore, from these curves the probability of rare events occurrence can also be computed.

The analysis approach presented in this section is summarised in fig. 12 in which the steps to be performed for the analysis of a soft real-time system are provided.

5 Conclusions

As soft real-time systems are often analysed using hard real-time techniques, which are not suitable to account for the deadline misses rate allowed in such systems, the resulting system is over-dimensioned. To overcome this issue, in this paper, we have presented a modelling approach, based on the concepts of the POOSL language, that enables probabilistic modelling of soft real-time embedded systems. This approach relies on patterns that allow composition of system models consisting of tasks, resources and their associated schedulers, capturing the varying runtime system behaviour using distributions. By using them, models for design space exploration can be built easily.

Moreover, we presented an approach to perform average case performance analysis to appropriately dimension soft real-time systems. We show for a case study that, using this approach, we could reduce the dimension of the system with more than 50% than what was found using analytical techniques. Furthermore, we presented a way to make an abstraction of the analysis results of the timing behaviour to use it as input for multi-disciplinary models.

As future work, we aim at extending the probabilistic 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

1. Balsamo, S., et al.: Model-based performance prediction in software development: A survey. IEEE Trans. on Software Engineering **30**(5) (2004) 295–310
2. Gries, M.: Methods for evaluating and covering the design space during early design development. Integration, the VLSI Journal **38**(2) (2004) 131–183
3. Wandeler, E., et al.: System architecture evaluation using Modular Performance Analysis - A case study. (Accepted in the STTT Journal)
4. Behrmann, G., et al.: A tutorial on UPPAAL. In: Proc. of SFM. (2004) 200–236
5. Lieverse, P., et al.: A methodology for architecture exploration of heterogeneous signal processing systems. VLSI Signal Processing Systems **29**(3) (2001) 197–207
6. Pimentel, A.D., et al.: Exploring embedded-systems architectures with Artemis. Computer **34**(11) (2001) 57–63
7. Theelen, B.D.: Performance modelling for system-level design. PhD thesis, Eindhoven University of Technology (2004)
8. Chakraborty, S., et al.: A general framework for analysing system properties in platform-based embedded system designs. In: Proc. of DATE, IEEE (2003)
9. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2) (1994)
10. Kahn, G.: The semantics of simple language for parallel programming. In: Proc. of IFIP Congress. (1974)
11. Liu, C., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real time environment. J. of ACM **20**(1) (1973) 46–61
12. Buttazzo, G.C.: Hard real-time computing systems: predictable scheduling algorithms and applications. Kluwer Academic Publishers (1997)
13. Bini, E., et al.: A hyperbolic bound for the rate monotonic algorithm. In: Proc. of ECRTS, IEEE (2001) 59–66
14. Manolache, S.: Analysis and optimisation of real-time systems with stochastic behaviour. PhD thesis, Linkpings University (2005)
15. Kienhuis, B., et al.: An approach for quantitative analysis of application-specific dataflow architectures. In: Proc. of ASAP. (1997)
16. Florescu, O., et al.: Performance modelling and analysis using poosl for an in-car navigation system. In: Appear in Proc. of ASCI. (2006)
17. (POOSL) http://www.es.ele.tue.nl/poosl.
18. Geilen, M.G.: Formal techniques for verification of complex real-time systems. PhD thesis, Eindhoven University of Technology (2002)