Platform-independent Design for Embedded Real-time Systems*

Jinfeng Huang, Jeroen Voeten
Faculty of Electrical Engineering Eindhoven University of Technology
5600MB Eindhoven, The Netherlands
E-mail: {J.Huang,J.P.M.Voeten}@tue.nl

Andre Ventevogel
TNO Industrial Technology
5600HE Eindhoven, the Netherlands

Leo van Bokhoven
Magma Design Automation B.V.
5600MB Eindhoven, The Netherlands

Abstract

With the increasing complexity of the emerging embedded real-time systems, traditional design approaches can not provide sufficient support for the development of these systems anymore. They especially lack the ability to trace and analyse real-time system properties. In this paper, we investigate the design difficulties for embedded real-time systems and propose several principles for coping with these difficulties, which should be incorporated by an "adequate" design approach. Several prevailing design approaches are evaluated against these principles and their merits and drawbacks are examined and illustrated by examples. Finally, a platform-independent approach (POOSL[vdPV97, Gei02] + rotalumis[vB02]) is introduced to remedy these design problems for embedded real-time systems. Initial experiments have been performed that confirm the advantages of this approach.

1 Introduction

Over the past decades, we have witnessed a significant increase of the application of embedded real-time (RT) systems. These applications range from microchip product lines with microsecond response time to electric appliance controllers with seconds or minutes response time. The common feature of these systems is that the correctness of the system depends not only on the value of the computation but also on the time when the results are produced [SR88]. To guarantee the timeliness feature in the system implementation, we expect the system behaviour to be predictable and we would like to ensure in advance that all critical timing constraints are met. However, with the increase of complexity, traditional design approaches are not well-suited for developing time-critical applications. We list below several typical characteristics in traditional design approaches that lead to unpredictable behaviour of embedded RT systems.

- *Craft-based system implementation:* Traditional methods are often bottom-up and driven by the specifics of the implementation platform. Critical timing constraints are dealt with in ad-hoc heuristic fashions. During system development, developers make implementation choices based on rough estimations of computation time. However, computation time is

*This research is supported by PROGRESS, the embedded systems research program of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs, the Technology Foundation STW and the Netherlands Organisation for Applied Scientific Research TNO.

in fact influenced by many non-deterministic factors of the underlying platform [1]. This severely hampers to make adequate estimations of computation time. Hence inadequate estimations can easily result in faulty design and implementation decisions.

- *Ineffective design languages:* Traditional programming or modelling languages usually do not provide sufficient support for embedded RT systems.

  1. These developing languages usually have platform-dependent execution semantics, which means that the behaviour of the system is heavily affected by the (non-deterministic characteristics of the) underlying platform. This makes it very difficult to establish design correctness.
  2. Design languages often have inadequate support for modularisation. Modularity means that the system can be decomposed into independent subsystems and the composition of these subsystems have little impact on their own properties and functionalities[Par90]. Object-orientation helps developers improve the modularity of the software by grouping together data-structures and operations and encapsulating them from the outside world. In this way, a complex system can be divided into independent subsystems that can more easily be designed , analysed and composed. However, in embedded RT systems, module independency is ruined by the fact that all modules running on one processor share the time resource. Therefore, after the composition of subsystems, the original real-time properties of the individual subsystems can not be sustained.
  3. Last but not the least, design languages often have little or no support for expressing timing requirements (such as deadlines and periods). This means that timing requirements cannot be taken into account when the system is mapped onto the target platform.

Due to the above pitfalls, embedded real-time systems lack portability, adaptability, maintainability and reusability. Furthermore, the correctness and performance has to be established by extensive testing on the target platform. The disadvantages are obvious:

- Tests are usually carried out at the observable boundary of the system. It is hard to locate the specific internal details that cause (timeliness) errors.
- Additional test code may change the (real-time) behaviour of the system and may cause new errors (Heisenberg principle in testing [Vra98, Ros96]).
- Tests can only be carried out at a late stage in the design cycle. Failures detected at this stage often lead to expensive and time-consuming design iterations.
- Test results are affected by many uncontrollable external (environmental) and internal (non-deterministic factors). Some "rare" errors are hard to capture or repeat by test.

2 The Dream: Platform-Independent Design

The key problem of traditional approaches mentioned in the previous section is that low-level details have a big influence on the system behaviour as a whole. Traditional design approaches do not provide facilities to abstract from these details adequately. In this sense, embedded

[1] A platform, in this context, refers to those hardware and software infrastructures on which the implementation relies. Typically, a platform includes hardware, operating systems and middle-ware as well. Non-deterministic factors of the platform are caused by techniques which have been widely applied to boosting the overall computation performance, such as caches, pipelines, instruction pre-fetch techniques and memory management[But97].

RT system design suffers from "butterfly-effects": a cache miss might result in a missed deadline. To cope with this problem, design approaches should offer adequate abstraction facilities and shield the design from the details of the platform. Platform-independent approaches [LDG02, SHvRH01] have been proposed to address this problem, which ideally have the following characteristics:

- ***A well-founded*** [2] ***and expressive modelling language:***
  The core of such a design approach is a modelling language which can help designers express and verify their design ideas in an adequate way. This means that the language should be expressive, should have platform-independent semantics, operational semantics and adequate support for modularization:

  1. *Adequate expressive power:* embedded RT systems often have features of timeliness, concurrency, distribution and complex functionality. To be able to expresses these features in a model, a modelling language should have facilities to describe timing, concurrency, communication, system structure, data types and non-determinism.
  2. *Platform-independent semantics:* the non-deterministic factors introduced by the underlying platform make the simulation and verification results of the design model unreliable and unpredictable. Platform-independent semantics of the modelling language gives a unique interpretation of the model and makes simulation and verification unambiguous. Furthermore, it provides the flexibility to reuse the design model and target it to different or modified platforms.
  3. *Operational semantics:* operational semantics of the modelling language lends itself naturally to executability [z1000]. Inconsistency between different aspects can then be located and correctness and performance properties can be checked either exhaustively (e.g. model-checking) or non-exhaustively (e.g. simulation). As a result, many design errors can be corrected in an early development stage, avoiding costly and time-consuming iterations.
  4. *Modularity support:* modularity is generally considered as the only available way for managing complexity of a system [GJ87]. Since complexity is a common feature of current embedded RT systems, it is necessary to embody modularity in the modelling language. This means that a concept of modules should be supported in such a way that module composition does not cause the real-time properties of the individual components to change. This allows components to be composed on basis of their interfaces, without having to understand the internal details.

- ***Automatic and correctness-preserving transformation:***
  System generation should take a design model as blueprint and build a complete hardware/software implementation. Preferable, the generation of the implementation should be largely automated. Furthermore, the transformation should also guarantee that the implementation behaves the same as the model. In this way, many errors caused by human-factors can be avoided during the system generation and correctness of the system can be guaranteed during the transformation.

The above characteristics of a design approach can help designers overcome the existing drawbacks of traditional design approaches and serve as guidelines for new embedded RT system design approaches. In the sequel, we will discuss several prevailing design approaches for RT systems and evaluate them based on the above characteristics.

[2] A well-founded language refers a language with formal semantics which enables an unambiguous interpretation on its expressions.

		Rational Rose RealTime (ROOM)	*Cinderella SDL (CSDL)*	*TAU Generation 2 (Tau2)*	*Esterel*
Design language		UML	SDL*	SDL+UML**	Esterel
Expre -ssive Power	*Timing*	limited	limited	limited	good
	Structure, data type, concurrency, communication	yes	yes	yes	yes
	non-determinism	yes	yes	yes	no
Platform-independent Semantics		no	no	yes	yes
Operational Semantics		yes	yes	yes	yes
Modularity Support		limited	limited	yes	yes

(a)

	Rational Rose RealTime (ROOM)	*Cinderella SDL (CSDL)*	*(TAU2) TAU Generation 2*	*Esterel*
Target language	C, C++, Java	no	C, C++	C, C++, Java, Hardware
Automatic generation	yes	no	yes	yes
Correctness-preserving transformation	no	no	no	yes

(b)

Figure 1: Comparison of several design approaches

* The semantics of SDL is based on SDL-96 in CSDL[CSD]. ** TAU2 integrates concepts of SDL-2000 and UML 2.0 in its modelling language[TAU].

3 Comparison of several design approaches for embedded RT systems

Figure 1 gives a brief comparison of several typical design approaches [3], all of which provide a powerful design language and most of which also support automatic software/hardware generation. In this way, software/hardware productivity can be improved and costly design interpretation errors can be reduced. However, none of them satisfies all those characteristics of platform-independent design. In the following, we will investigate how those characteristics are supported by these approaches.

3.1 Expressive power

Timing: CSDL and TAU2 only support time delays. There is no explicit timing expression in Esterel. Instead, it captures the time passing by counting activations of the system [BG92]. The modelling of physical time can be accomplished by counting activations issued at a regular time interval. Due to the deterministic characteristic of Esterel models (see below in this section), this form of real-time mechanism is adequate enough for describing and analysing timing constraints. Compared to the other three approaches, ROOM [Ros] provides various timing services to express different timing constraints, such as delay timers, periodic timers, informIn timers. However, its platform-dependent semantics has an ambiguous interpretation of time expressions in the model, which inhibits designers from analysing and predicting timing behaviour of the model.

Structure, data type, concurrency and communication: These features are supported by all of the above approaches. In this paper we are not going into detail about how these approaches support these features. For more information, we refer to [VvdPGS98, TVK03].

[3] Figure 1a gives a comparison of design languages of these approaches and Figure 1b gives a brief comparison of their code generation ability and quality.

Non-determinism: Non-determinism is an essential way to manage the system complexity. It not only leaves freedom to obtain optimal implementations but also largely reduces the complexity of the model by abstracting it from irrelevant implementation details. Except for Esterel, all the three approaches can describe an embedded RT system at a high level of abstraction by using non-determinism. At the same time, they offer the possibility to model interested details of the system. Esterel sacrifices non-determinism to obtain a deterministic model of the system, in which the system behaviour is predictable. The cost of the deterministic characteristics is that complex behaviour is difficult to be modelled and analysed efficiently.

3.2 Platform-independent semantics

CSDL relies on an asynchronous timer mechanism which is able to access a global clock referring to the physical clock [4]. In this time mechanism, the delay between the moment of timer expiration and the moment at which the model reacts to this expiry is unbounded [VvdPGS98]. The unbounded delay is caused by several factors, such as the waiting time of timer-expired message before it is inserted into the input message queue of the process, the time for consuming all messages before the timer message and the interaction time between the process and its input message queue [Leu95]. When a timer is set to t seconds, the interpretation of this timer is in fact an arbitrary time duration d_t ($d_t \in [t, \infty)$). Such a weak interpretation of timers cannot provide enough expressive power to describe the timing behaviour of real-time systems. The unbounded uncertainty of d_t contradicts with the predictability of real-time system behaviour.

Example 1 *Consider a simple digital clock, which issues an action at the end of each second to count the time passing. Due to the unbounded uncertainty of a timer in CSDL, it is impossible to ensure that actions can be issued at (or close to) the expected time points.*

The time mechanism in CSDL is heavily affected by the platform-dependent physical clock. Such a platform-dependent timing mechanism cannot provide facilities to debug and analyse timing behaviour of a model, because any debugging and analysis observation may introduce extra time passing, which changes the real-time behaviour of the model and leads to unreliable debugging and analysis results. The same problem holds for the ROOM approach.

Different from CSDL and ROOM, TAU2 adopts a two-step execution model[NS91]. The state of a system can change either by asynchronously executing some atomic actions such as communication and data computation without time passing (phase 1) or by letting time pass synchronously without any action being performed (phase 2). The semantics of Esterel is based on the perfect synchrony hypothesis[Ber92, BG92], which assumes that a system's reaction to an input is instantaneous. The time measurement is achieved by counting the number of events. The computation models of both TAU2 and Esterel adopt a virtual time, whose progress is not affected by the physical-time passing directly. In this way, real-time behaviour of their models is always predictable with respect to this virtual time. Therefore, the above unbounded uncertainty problem does not exist in these design approaches. Furthermore, in these approaches, a model can be uniquely interpreted and analysed by verification and simulation techniques. Based on the analysis results, the model can be refined to meet predefined timing requirements. A clock example in section 4.1 illustrates that timing behaviour can be adequately modelled by such design approaches.

3.3 Modularity support

All of these approaches have object-oriented characteristics supporting data and functionality encapsulation. Due to the platform-dependent semantics of CSDL and ROOM, timing charac-

[4] Although SDL-2000 uses a virtual time to count the time passing in its models[z1000], SDL-96 adopts physical time[Leu95].

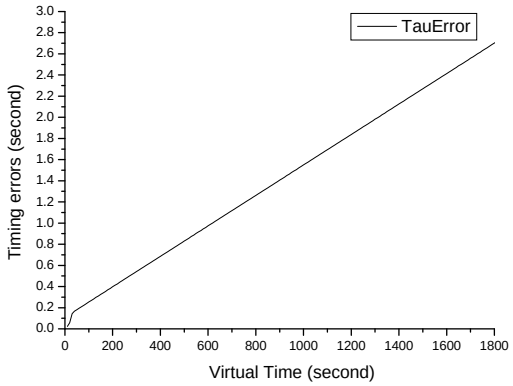

Figure 2: Accumulated timing errors of a TAU 2 clock implementation

teristics of every module in the model are disturbed by the time consumption of other modules in the model and by other processes running on the same platform. On the contrary, both Esterel and TAU2 assume the underlying platform to be infinitely fast and the timing behaviour of the system is not constrained by the computational power of the underlying platform. Therefore, the composition of separate modules does not change their timeliness. As a consequence, the analysis of the design is easier and predictable.

3.4 Correctness-preserving transformation

As shown in Figure 1b, all approaches facilitate automatic software or hardware generation except for CSDL. In ROOM, automatic code generation is accomplished by linking a model to a so-called service library which acts as a virtual machine on top of different target platforms. In this sense, the implementation is in fact an executable model and problems encountered in the model, such as platform-dependent semantics, are automatically inherited in the implementation. Although TAU2 can provide a reliable way to analyse a model and refine it to ensure the correctness, it does not have a transformation mechanism to guarantee that correctness properties verified in the model can be transferred to the implementation. In the automatically generated implementation, timing expressions are simply interpreted as unbounded physical time, which faces the same unbounded uncertainty problem as in the CSDL model. The issuing time of actions can deviate much from those observed in the model. Furthermore, the ordering of events can also be different from those observed in the model. Here are several examples.

Example 2 ***Accumulated timing errors:***

Consider a digital clock whose functionality is similar to that described in Example 1. The difference is that its accuracy is one tenth of a second instead of 1 second. Figure 2 shows that the timing errors [5] *are accumulated during the execution of the implementation, which is automatically generated from the clock model in TAU2.*

Example 3 ***Incorrect functionality caused by accumulated timing errors:***

*Consider a controller for a flash board showing 4 consecutive letters "**IEEE**", with the following functionality. The four letters of the word are sequentially displayed on the board, and then wiped off altogether at the same time. The iteration will continue unless it is interrupted manually. One solution to designing this controller is to use three parallel processes. Process **I** emits letter I every 0.3 seconds, process **E** emits letter E every 0.1 seconds and process **space** issues four blank spaces every 0.3 seconds to erase the letters. The three processes starts from*

[5] Timing errors represent the deviation of the issuing time of actions in the implementation from that in the model, i.e., they represent difference between the virtual time and the physical time of the issuing actions.

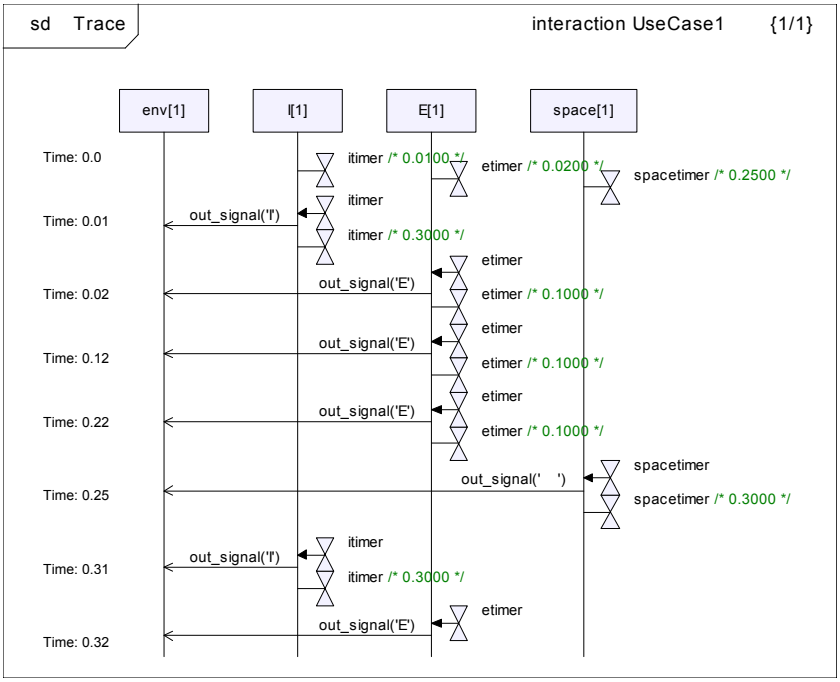

Figure 3: The sequence diagram of the controller

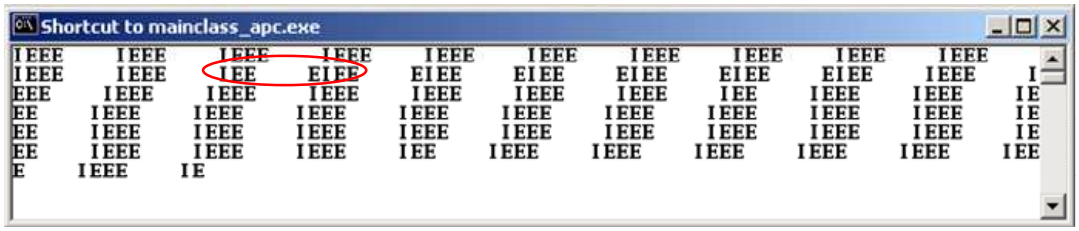

Figure 4: Output of the controller

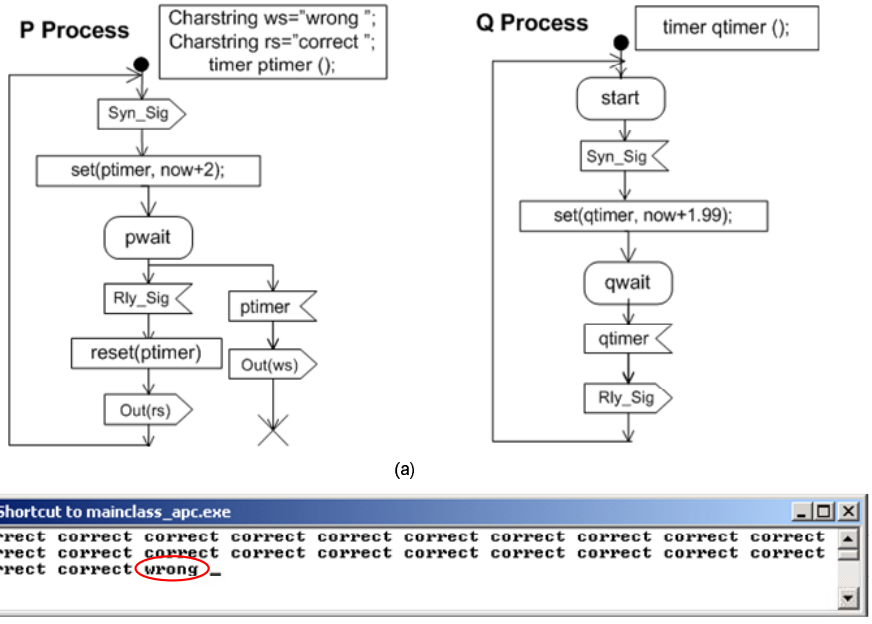

Figure 5: Two parallel processes and the output of their TAU2 implementation

0.01,0.02 and 0.25 second respectively. Figure 3 illustrates a piece of the sequence diagram created during the model simulation (working with the virtual time). It is easy to verify that this model behaves correctly according the functionality specification. However, we will see a different picture when we look at the behaviour of the implementation (working with the physical time). Figure 4 gives a snapshot of the output of the implementation. We can see that after several iterations, the accumulated timing errors have led to an incorrect output sequence.

Example 4 ***Incorrect functionality caused by duration actions:*** *Not only can accumulated timing errors lead to incorrect event order, computational expense of individual actions can also result in unexpected behaviour. Consider the simple example of Figure 5a. Two parallel processes **P** and **Q** synchronize at the beginning of each iteration to avoid the accumulation of timing errors. **P** sets a timer with 2 seconds delay and **Q** sets a timer with 1.99 seconds delay. After the timer of **Q** expires, **Q** sends a "Rly-sig" message to **P**. At the **P** side, there are two possibilities:*

1. ***P** receives the timer expiration message and outputs the message "wrong".*
2. ***P** receives the reply message from **Q**, resets its own timer and outputs the message "correct".*

*It is not difficult to verify that the output message of the **P** should always be "correct" in the model. However, the automatically generated software implementation exhibits unexpected behaviour (see Figure 5b).*

Different from TAU2, Esterel provides a correctness-preserving transformation from a model to an implementation. The Esterel language is a synchronous concurrent programming language based on perfect synchrony hypothesis, which assumes computation actions to take zero time. A pure Esterel model can be implemented in a digital circuit or software by using a formal transformation. Such a transformation keeps the semantics of the Esterel model to the hardware/software implementation except that perfect synchrony hypothesis is replaced by digital synchrony hypothesis (i.e. zero time is replaced by one cycle clock). The correctness of the generated implementation relies on the fact that perfect synchrony does not deviate very much from digital circuit synchrony [Ber92].

4 Towards Platform-independent Design

In the previous section, we have reviewed several typical design approaches for embedded RT systems and analysed the drawbacks and merits of each approach. Among them, TAU2 provides a relatively good support for modelling complex embedded RT systems but it provides no facilities for correctness-preserving transformation from a model to its implementation. Esterel has full support for correctness-preserving transformation for automatic hardware/software generation but its modelling language lacks the ability to model complex interactive RT software systems [Ber00]. In this section we propose an approach that considers all aspects introduced in section 2 and aims at platform-independent design. It provides an expressive and well-founded language (POOSL) for modelling and analysing complex embedded RT systems and a correctness-preserving transformation tool (rotalumis) for automatic software generation. Its ability to preserve correctness during transformation have been proven in [HVG03].

4.1 POOSL

In this section we give a brief overview of the POOSL language (Parallel Object-Oriented Specification Language), which is employed in the SHEsim tool and developed at the Eindhoven

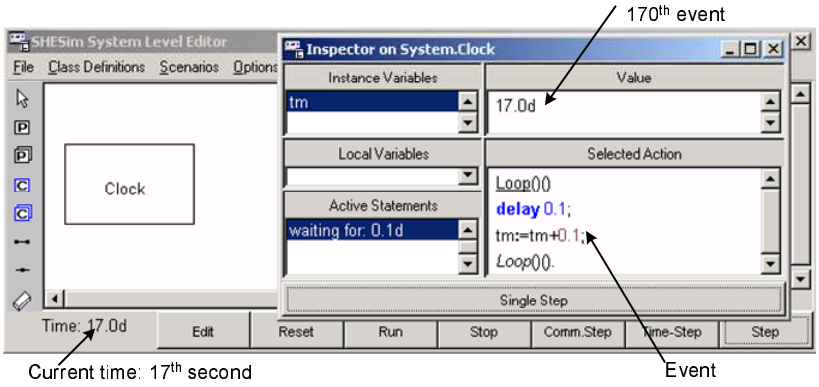

Figure 6: A clock model in SHEsim

University of Technology. The POOSL language is equipped with a complete mathematical semantics and can formally describe concurrency, distribution, communication, real-time and complex functionality features of a system in a single executable model.

Similar to TAU2, the semantics of the POOSL language is based on a two-phase execution model which guarantees a unique interpretation of the model. Hence, the behaviour of the model is not affected by underlying platforms. The detailed mathematical framework behind the POOSL language is given in [vdPV97, vB02], and a formal description of its execution engine can be found in [Gei02]. Figure 6 shows a simple clock in POOSL which performs the same functionality as in Example 2. Each event is accurately issued at the expected (virtual) time in this model (for example, the 170th event is issued at the 17th second.).

Because of the expressiveness and well-founded semantics of the POOSL language, it has been successfully applied to model and analyse many industrial systems such as an internet router[TVvB$^+$01], a network processor[TVK03] a microchip manufacture device[HVvdP$^+$02] and a multimedia application[vWVtB02].

4.2 Rotalumis

In this section, we outline the formal transformation mechanism of software generation tool rotalumis, which can transform a POOSL model into a software implementation for single processor platforms [vBVG99]. Different from other software generation tools, the rotalumis supports the correctness-preservation during the transformation by applying the following techniques.

1. Execution trees are used to bridge the gap between the expressibility difference between POOSL and C++ language. POOSL provides ample facilities to describe system characteristics such as parallelism, preemption, nondeterministic choice, delay and communication that are not directly supported by C++. In order to provide a correct and smooth mapping, execution trees are adopted to represent individual processes of the model and a scheduler calculates their next step actions in the execution of the model. The correctness of this execution method with respect to the semantics of the POOSL model has been proven in [Gei02]. Therefore, the generated C++ software implementation will always have the same (untimed) event order as observed in the POOSL model. More details about the execution trees can be found in [vB02].

2. Correctness property preservation is guaranteed by the ϵ-hypothesis in the software implementation, which assumes that the timed execution trace of the implementation is always ϵ-neighbouring [6] to a timed execution trace in the model. It has been proven that the

[6] A timed execution trace is a state sequence with a time interval attached to every state. If two timed

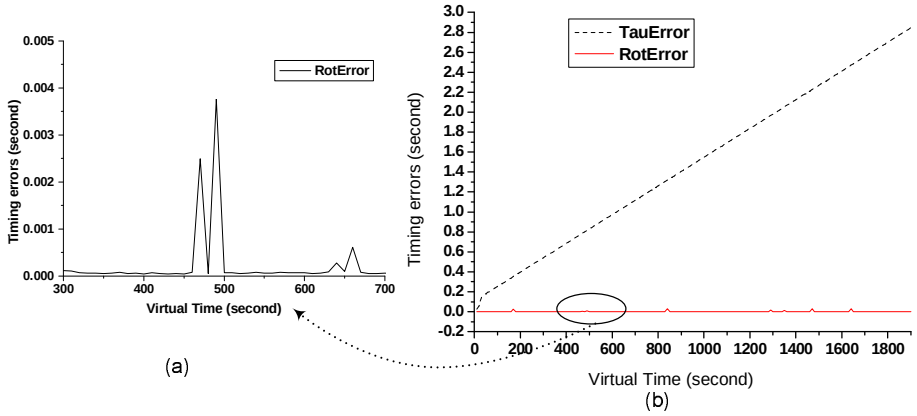

Figure 7: A comparison of the timing errors

evolvement of execution trees always follows one of these state traces in the model. Furthermore, during the execution, the scheduler of execution trees tries to synchronize the virtual time and the physical time, which ensures that the execution of the implementation is always as close as possible to a trace in the model with regard to the distance between timed state sequences. Due to the limitation of the platform, the scheduler may fail to guarantee the timing constraints specified in the model, even with ϵ-neighbouring relaxation. In this case, designers can get the information about the missed actions. Correspondingly, they can either refine the model and reduce computation cost, or replace the target platform by a platform of better performance.

In Figure 7b, we give a comparison of the timing errors of two automatically generated clocks from TAU2 and rotalumis respectively, which run on the same platform [7]. The functionality of the clocks is as stated in Example 2. The clock generated by rotalumis is supposed to satisfy 0.1-hypothesis (i.e, the scheduler tries to synchronize the virtual time and the physical time within 0.1 second difference.). Figure 7a shows that the timing errors of the implementation is controlled within 0.1 second and does not accumulate with the time passing [8]. In addition to the clock example, errors in Example 3 and Example 4 can also be avoided in the generated implementations by rotalumis.

5 Conclusions

In this paper, we have analysed the difficulties experienced when designing complex embedded RT systems and proposed several essential characteristics that an "adequate" design approach should possess in order to overcome these difficulties. Several prevailing approaches have been evaluated based on how well they support these characteristics. The experiments on these approaches indicate that none of them has full support for complex embedded RT system design. A platform-independent design approach is proposed which considers all essential characteristics. The approach consists of two individual procedures, platform-independent design and correctness-preserving transformation. Platform-independent design guarantees an unambiguous interpretation to the design description, whose performance and correctness can be analysed

execution traces are ϵ-neighbouring, they have exactly the same state sequence and the least upper bound of the absolute difference between the left-end points of the corresponding intervals is less than or equal to ϵ. For more information, see [HVG03].

[7] CPU 150Mhz, Memory 128M and Windows 2000

[8] Several peaks in Figure 7a are caused by the underlying OS (Windows 2000). We have tried to execute the same implementation in other OS, such as Windows 98, and no such high peaks exist. In most situations, timing errors are around $5 * 10^{-5}$ seconds in rotalumis. Note that the timing errors of both implementations are observed after the issuing of the actions, i.e. the actual timing errors should be less than those shown in Figure 7.

by verification and simulation techniques. Based on the analysis results, a design description can be refined to meet the predefined requirements — a prerequisite for the procedure of the correctness-preserving transformation from the model to the implementation. The correctness-preserving transformation takes a model as input, and generates a complete and executable implementation whose correctness is guaranteed during the transformation. The time and cost to perform test is thereby saved. The tools for both procedures have been developed at the Eindhoven University of Technology. The SHEsim tool with the POOSL language provides a platform-independent environment for designing, simulating and analysing a model. The rotalumis tool can automatically transform a POOSL model into executable code for target platforms preserving the correctness properties verified in the model. Initial experiments have been performed that confirm the advantages of this approach.

References

[Ber92] G. Berry. A hardware implementation of pure esterel. In *Academy Proceedings in Engineering Sciences*, volume 17, pages 95–130. Indian Academy of Sciences, 1992.

[Ber00] G. Berry. *Proof, Language and Interaction: Essays in Honour of Robin Milner*, chapter The Foundations of Esterel, pages 425–454. MIT Press, 2000.

[BG92] Gérard Berry and Georges Gonthier. The esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[But97] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers, Boston, 1997.

[CSD] Cinderella SDL 1.3. http://www.cinderella.dk/.

[Gei02] M.C.W Geilen. *Formal Techniques for Verification of Complex Real-time Systems*. PhD thesis, Eindhoven University of Technology, The Netherlands, 2002.

[GJ87] C. Ghezzi and M. Jazayeri. *Programming Language Concepts: 2nd Edition*. John Wiley & Sons, Inc., New York, 1987.

[HVG03] Jinfeng Huang, Jeroen Voeten, and Marc Geilen. Real-time Property Preservation in Approximations of Timed Systems. In *Proceedings of First ACM & IEEE International Conference on Formal Methods and Models for Codesign*, Mont Saint-Michel, France, June 2003. IEEE Computer Society Press.

[HVvdP+02] Jinfeng Huang, Jeroen Voeten, Piet van der Putten, Andre Ventevogel, Ron Niesten, and Wout van de Maaden. Performance evaluation of complex real-time systems, a case study. In *Proceedings of 3rd workshop on embedded systems*, pages 77–82, Utrecht, the Netherlands, Oct 2002.

[LDG02] L. Lavagno, S. Dey, and R. Gupta. Specification, modeling and design tools for system-on-chip. In *Design Automation Conference, 2002. Proceedings of ASP-DAC 2002. 7th Asia and South Pacific and the 15th International Conference on VLSI Design. Proceedings.*, pages 21–23. IEEE Computer Society Press, 2002.

[Leu95] S. Leue. Specifying Real-Time Requirements for SDL Specifications - A Temporal Logic-Based Approach. In *Proceedings of the Fifteenth International Symposium on Protocol Specification, Testing, and Verification*, Chapmann & Hall, 1995.

[NS91] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In K. G. Larsen, editor, *Proceedings of the 3rd workshop on Computer-Aided Verification*, Alborg, Denmark, July 1991.

[Par90] H. A. Partsch. *Specification and transformation of programs: a formal approach to software development.* Springer-Verlag New York, Inc., 1990.

[Ros] Rational Rose RealTime. http://www.rational.com/tryit/rosert/index.jsp.

[Ros96] J. B. Rosenberg. *How Debuggers Work: Algorithms, Data Structures, and Architecture: 1st edition.* John Wiley & Sons, Inc., Oct 1996.

[SHvRH01] A. Sintotski, D.K. Hammer, O. van Roosmalen, and J. Hooman. Formal platform-independent design of real-time systems. In *Proceedings 13th Euromicro Conference on Real-Time Systems*, pages 163–170. IEEE Computer Society Press, 2001.

[SR88] J. Stankovic and K. Ramamritham, editors. *Tutorial on Hard Real-Time Systems.* IEEE Computer Society Press, 1988.

[TAU] TAU Generation 2. http://www.taug2.com/.

[TVK03] B.D. Theelen, J.P.M. Voeten, and R.D.J. Kramer. Performance Modelling of a Network Processor using POOSL. *Journal of Computer Networks, Special Issue on Network Processors*, 41(5):667–684, April 2003.

[TVvB+01] B.D. Theelen, J.P.M. Voeten, L.J. van Bokhoven, P.H.A. van der Putten, G.G. de Jong, and A.M.M. Niemegeers. Performance modeling in the large: A case study. In *Proceedings of the European Simulation Symposium*, pages 174–181, Ghent (Belgium), October 2001.

[vB02] L.J. van Bokhoven. *Constructive Tool Design for Formal Languages from semantics to executing models.* PhD thesis, Eindhoven University of Technology, The Netherlands, 2002.

[vBVG99] L.J. van Bokhoven, J.P.M. Voeten, and M.C.W. Geilen. Software Synthesis for System Level Design Using Process Execution Trees. In *Proceedings of 25th Euromicro Conference*, pages 463–467, Milan, Italy, 1999. IEEE Computer Society Press, Los Alamitos, California.

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

[Vra98] H.P.E. Vranken. *Design for test and debug in hardware/software systems.* PhD thesis, Eindhoven University of Technology, The Netherlands, 1998.

[VvdPGS98] J.P.M. Voeten, P.H.A. van der Putten, M.C.W. Geilen, and M.P.J. Stevens. System Level Modelling for Hardware/Software Systems. In *Proceedings of EUROMICRO'98*, pages 154–161, Los Alamitos, California, 1998. IEEE Computer Society Press.

[vWVtB02] F.N. van Wijk, J.P.M. Voeten, and A.J.W.M. ten Berg. An abstract modeling approach towards system-level design-space exploration. In *Proceedings of the Forum on specification and Design Language*, Marseille, France, September 2002.

[z1000] Z.100 Annex F1: Formal Description Techniques (FDT)–Specification and Description Language (SDL). Telecommunication standardization sector of ITU, Nov 2000.