Combining an Iterative State-Based Timing Analysis with a Refinement Checking Technique

Björn Koopmann, Achim Rettberg, Tayfun Gezgin

Combining an Iterative State-based Timing Analysis with a Refinement Checking Technique

Tayfun Gezgin, Björn Koopmann, and Achim Rettberg

Carl von Ossietzky University Oldenburg,
Ammerländer Heerstr. 114, 26121 Oldenburg, Germany

Abstract. The analysis of real-time properties is crucial in safety critical areas like in automotive applications. Systems have to work in a timely manner to offer correct services. Most of the applications in this domain are distributed over several computation units, inter-connected by bus systems. In previous works we have introduced a state-based analysis approach to validate end-to-end deadlines for distributed systems. The approach is based on the computation of the state spaces of all resources, such as processors and buses, in an iterative fashion. For this, abstraction and composition operations were defined to adequately handle task and resource dependencies. During the design process of a system changes occur typically on both the specification and implementation level, such that already performed analyses of the system have to be repeated. In this work, we extend our timing analysis with a refinement checking approach, detail when it is appropriate to be used, and compare the analysis times with the computation times to perform the refinement check.

Keywords: Real-time Systems, Scheduling Analysis, Re-Validation, Timing Analysis, State-based Timing Analysis

1 Introduction

In recent years the co-operations and inter-connections between individual, geographically distributed systems heavily increased. Also in safety critical areas the significance of these topics increased. As an example, much effort has been invested in the development of Car-to-Car communications with the aim to increase the safety in traffic and optimize traffic flows. Another example is the dynamic partitioning of the airspace with respect to time investigated in the SESAR (Single European Sky ATM Research) program. The recent partitioning of the airspace is performed in a static manner with respect to time, i.e. the trajectories are not changed during the whole landing approach and the takeoff. The shift to a dynamic partitioning, which is called 4D-trajectories, involves a much more intensive co-operation between the tower and each airplane.

For the correct functionality of safety-critical functions of such systems, timing constraints are one crucial aspect. The final product has to satisfy those constraints, as the violation of a requirement could result in high costs or even

threats to human life. Nissan for example had to recall the vehicles of its premium segment cars due to some delays in the emergency program of their new steer-by-wire system. Such a problem could have been avoided, if an early analysis on timing constraints would have been performed. Unfortunately, many changes occur during the design process, such that already performed analyses have to be repeated. Our approach targets these problems.

In [1] we worked out a state-based approach for the analysis of timing properties. In analogy to model checking methods, we consider the full state space, where all task interleavings are preserved. In order to alleviate the problem of state space explosion due to state unfolding, the state space of an architecture is constructed in an iterative manner. Abstraction methods are applied to keep the interfaces between components as small as possible, while composition operations are used to combine a set of triggering sources of a component.

On top of this we worked out an impact analysis approach to minimize re-validation efforts of timing properties needed when the considered system is modified [2]. Adaptations of the architecture of an already existing and analyzed system could be for example the addition of new tasks that are allocated to the existing system. To minimize the effort of a re-validation, it is desirable to reuse the previous results of the analysis that did not change. With this, only the parts are re-validated, which were affected by the architectural changes.

This work is a consecutive extension of our previous work [2]. We illustrate the implementation of the impact analysis. We describe in which cases a refinement check can be applied to reduce the re-verification times when changes occur. We evaluate our approach by a set of test systems demonstrating the computation times needed to perform a full timing analysis and the times needed to perform the impact analysis consisting of the loading and storing of state spaces, and the refinement check between state spaces. Further, we discuss the benefit of applying abstractions of resource interfaces for the refinement relation.

Related Work

Timing analysis on distributed systems is a very large research area. Thus, we cover only the most relevant works for our approach. The classical approach is a holistic one, as it was worked out in e. g. [3, 4]. Local analysis is performed evaluating fixed-point equations. These approaches are very fast and able to handle large systems. Unfortunately, the analytical approaches deliver pessimistic results if inter-ECU task dependencies exist. In [5] activation patterns for tasks are described by upper and lower arrival curves realizing a *compositional* analysis method. Based on this work a compositional scheduling analysis tool, called SymTA/S, was created by SymtaVision [6]. The concept has been developed by Richter et al. The main idea behind SymTa/S is to transform event streams whenever needed and to exploit classical scheduling algorithms for local analysis. Another related approach is the modular performance analysis (MPA) [7] which is based on a formalism with many similarities to event streams named Real-Time Calculus. Arrival functions are used to model the computation that is requested by a process, and service functions are used to model the amount of

computation that can be delivered by a resource. In [8], the MPA approach has been combined with timed automata while offering methods that allow to transform the model of one formalism to another. *CARTS* is another tool for compositional real-time scheduling analysis [9]. Schedulability is checked for tasks whose resource usage is bounded by periodic resource models developed by Lee et al. Composition is done on the resource model level resulting again in periodic resource models by using abstractions.

Another approach is based on model-checking: In [10] non-preemptive schedulers are modeled in terms of timed automata. The advantage of this approach is that one gets exact solutions with respect to the modeled scheduling problem. Since the state space of the analyzed system is preserved, checking complex properties like safety is possible. Unfortunately, state-based approaches do not scale well. The authors of [11] also use timed automata to model preemptive scheduling and verify timing properties by using UPPAAL. As a front-end they employ sequence diagrams, from which timed automata are derived. In [12] these automaton models were reused and the results were compared to other techniques such as MPA or SymTA/S. In [13] timed automata are extended by clocks which may be subtracted by a natural number to handle preemption in a more natural way. The authors derive a sub-class of this formalism, where the reachability is preserved.

Outline

First, we illustrate the considered problem domain. In Section 3 we will detail our general analysis approach in a condensed form. In Section 4 we introduce our implemented impact analysis methodology. Section 5 evaluates our concept and compares plain verification times and refinement checking times. Finally, we conclude the work and give an outlook for future work.

2 Problem Domain

We are interested in safety-critical real-time systems which are typically used in the automotive domain. Typically, the design of the overall system is performed by the original equipment manufacturer (OEM). The OEM designs the software components in form of logical architectures by using, e.g., Autosar software components (SWC), inter-connected by a high level virtual function bus (VFB) like illustrated in the left part of Figure 1. The components and parts of this system are then realized by the suppliers. In order to get adequate realizations from each supplier, the OEM has to specify the extra-functional properties and interfaces unambiguously. This is realized by the usage of so called contracts [14]. Contracts are pairs consisting of an assumption (A) and a guarantee (G). The assumption specifies how the context of the component, i. e. the environment from the point of view of the component, should behave. Only if the assumption holds, then the component will behave as guaranteed. To specify the assumptions and guarantees various formalisms like pattern-based languages could be

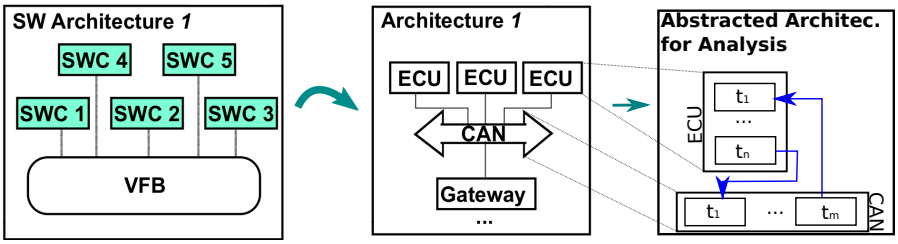

Fig. 1. General concept of modeling and analysis.

used. Contracts follow the principle of separation of concerns, i.e., a contract does not just specify a guarantee about the behavior of a component, but also an assumption about the behavior of the environment in which the component will be integrated.

If all suppliers deliver the implementations of the SWCs, the OEM has to verify whether all SWCs *fit together*, i.e., he has to perform the consistency check in a black box manner, and whether some higher level requirements ranging over several SWCs are realized by the decomposition structure.

After the implementation of all SWCs the logical architecture has to be allocated to the hardware architecture, consisting of electronic control units (ECUs) which are inter-connected by bus systems. At this design stage technical details such as resource consumptions and timing latencies have to be verified. To perform such analyses, typically the architecture is abstracted in an appropriate manner. The abstraction we perform for our analysis is illustrated in the right part of Figure 1: ECUs and bus systems are treated logically equivalent in the sense that both represent computation units on which a set of tasks are allocated. The order of executions of the tasks is determined by the corresponding scheduling policy like fixed priority scheduling. Dependent tasks are directly connected, tasks with no input edges are considered to work independent from other tasks. A task is characterized by a tuple $\tau = (bcet, wcet, pr)$, where $bcet, wcet \in \mathbb{N}_{\geq 0}$, $bcet \leq wcet$, are the best and worst case execution times with respect to the resource the task is allocated to, and $pr \in \mathbb{N}_{\geq 0}$ is the fixed priority of the task. We will refer to the elements of a task by indexing, e.g. $bcet_\tau$ for task τ . The set of all tasks is called T. Independent tasks are triggered by events of a corresponding event stream (ES). An event stream $ES = (p, j)$ is characterized by a period p and a jitter j with $p, j \in \mathbb{N}_{\geq}$. Such streams can be characterized by upper and lower occurrence curves as introduced in the real-time calculus [15]. In this work we restrict to event streams where $j_\tau < p_\tau$ for all $\tau \in T$. Like stated above we will further assume that dependent tasks are directly connected.

3 State-based Timing Analysis

Our timing analysis approach is based on model-checking. For each computation resource its state space is computed. Such a state space encapsulates the relevant

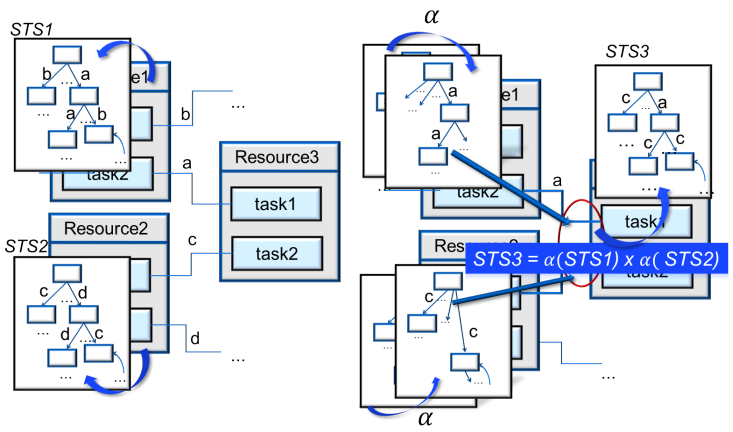

Fig. 2. Timing analysis approach; Left: Computation of resource state spaces; Right: Computation of output interfaces.

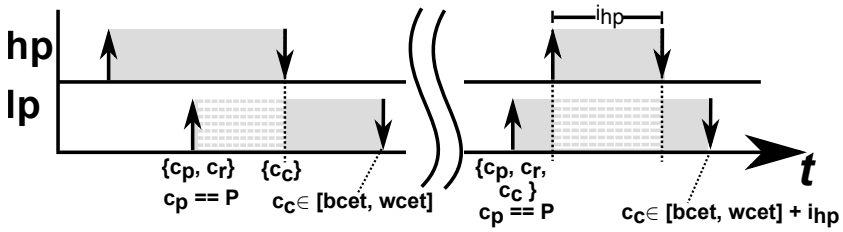

Fig. 3. Two tasks hp, lp and the interrupt scenarios. The clocks refer to the task lp. Clocks in curved brackets indicate a reset, P is the period of lp.

timing information for tasks allocated to the corresponding resource, and end-to-end latencies between a set of tasks. In contrast to standard model-checking, our approach does work in an iterative fashion. The interfaces between resources are tried to be kept as minimal as possible. Note that we assume cyclic free systems. Parts of systems with cycles have to be handled in a holistic fashion.

To build the state space of a computation resource, we have to determine its input behavior, which defines the activation times of all allocated tasks. State spaces are represented by symbolic transition systems (STS): the states determine a range of valuations of clock variables, and include the information, which task is currently running, is interrupted, or in the ready queue. A resource can have multiple sources for its inputs: the independent tasks are triggered by event streams, while dependent tasks are triggered when the tasks on which they depend, terminate. Thus, we get multiple input state spaces. To determine a *single* input state space for each resource, we have to combine all these inputs.

When the input is determined, the next step is to build the state space of the resource itself. For this, the input STS, the behavior of the scheduling policy, and the execution times and priorities of the allocated tasks are taken into account. The approach to compute the state space is illustrated in Fig. 3, where two tasks

hp, lp are allocated to a single resource with a fixed priority scheduling policy. For each task a clock c_p which traces the periodic activation is needed. Further, we need a clock c_c to determine when a task is finished. If we are interested in the exact response times of a task instance, we need multiple clocks c_r, one for each instance of a task.

The computed state space of a resource is then used as an input for *dependent* resources, i. e. for resources on which dependent tasks are allocated. To keep the interface between the resources as small as possible, parts of the state space that are not relevant for the input behavior of the dependent resources are abstracted.

Consider the example in Fig. 2, which consists of three resources where on each resource two tasks are allocated. The tasks `task5` and `task6` on resource `Resource3` depend on `task2` on resource `Resource1` and `task3` on resource `Resource2` respectively. Tasks `task1`,...,`task4` are activated by event streams, thus the inputs for both `Resource1` and `Resource2` are directly given and their state spaces can be computed (illustrated in the left part of the figure). Next, the input of `Resource3` has to be determined, which depends on both the state space of `Resource1` and `Resource2`. As timing information for the tasks `task1` and `task4` is not relevant for `Resource3`, the corresponding STSs can be reduced by abstracting from states encapsulating information about these tasks. After this minimization, the product of both STSs is computed (indicated by the right part of the figure).

The details of our timing analysis including the composition operation, the minimization, and the resource construction can be found in [1].

4 Impact Analysis Methodology

During the design process changes affecting the architecture of a system occur, such as adding a new task on an existing resource, the merge of two tasks in a single one, or even the change of the complete implementation. If such changes occur, already performed analyses have to be repeated, increasing the time needed to verify the functionality and properties of the design, and thus increasing the time to market.

To minimize the effort of a re-validation, it is desirable to reuse the previous results of the analysis that did not change. With this, only the parts are re-validated, which were affected by the architectural changes. It is required to perform an impact analysis, when changing or maintaining software because it allows to judge the amount of work required to implement a change, proposes software artifacts which should be changed, and helps to identify test cases which should be re-executed to ensure that the change was implemented correctly [16].

As our timing analysis approach works in an iterative manner (and not holistically), we are able to determine whether the interface of dependent resources is affected through the concept of our refinement analysis: we are able to check if the new interface between dependent resources refines the old interface. In such a case a re-validation of dependent resources can be omitted. The definition of an appropriate refinement relation was the topic of our previous work [2].

In the next section, we illustrate our implementation approach of the impact analysis. We demonstrate in which cases a complete re-verification of a component is necessary, in which cases a refinement check is performed, and when verification steps can be omitted. Thereafter we discuss the advantages of our approach when using further abstraction techniques on the interfaces of resources.

4.1 Concept

The concept of the implementation of our impact analysis is illustrated in Figure 8 in terms of an UML activity diagram.

Each resource has a status flag for its resource state space called *outputIsConsistent*, initially set to *false*. The idea of this flag is to inform dependent resources whether some non-refinement changes concerning the resource state space occurred (and thus the resource state space has to be recomputed).

First, it is checked whether some inputs of the resource has changed (*checkInputStatus*). If changes occurred, the check evaluates to *false* and the input STS (*computeInputSTS*) followed by the computation of the resource state space itself (*computeResourceSTS*)is performed as usual. As the resource STS is newly computed, the flag *outputIsConsistent* is set to *false* to signalize dependent resources that this input has changed. Last, the resource STS is stored appropriately. If on the other hand the output STSs of all resources, from which the current resource depends on, did not change, *checkInputStatus* evaluates to *true*. Then it is checked whether an already computed resource STS of this resource exists (from previous verification steps, where the resource STS was saved). If not, it has to be computed as described above. Else, it is checked whether structural changes have occurred, i.e. changes concerning the scheduling policy of the resource, the number of allocated tasks and their properties like priorities and execution times. If these properties did not change, the resource STS will also be not affected. Thus, the existing STS can be restored (loaded from file system). The flag *outputIsConsistent* is set to *true* indicating that nothing changed on the output.

If else some changes on the resource occurred *checkInputStatus* will evaluate to *false*. In this case, we have to re-compute the resource STS, load the previously computed resource STS and do a refinement check between both STSs. If the refinement check evaluates to *true*, *outputIsConsistent* is also set to *true* indicating that the resource STS changed in a good manner. Else it is set to *false*. Note that before the re-computation of the resource STS the input STS of the resource has also to be re-computed because if properties of independent tasks change the input STS is also affected.

4.2 Combination with Abstractions

Generally an impact analysis is useful in combination with analysis techniques that involve abstractions. This is also a typical scenario for analytic techniques such as in [17]. These techniques are based on the assumption that every interface

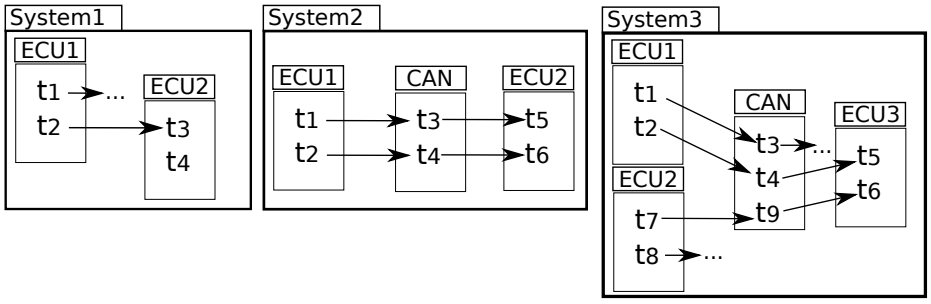

Fig. 4. Test systems.

behavior can be characterized by event streams. To obtain event streams for the outputs of a resource, the actual task behavior is generally over-approximated.

Hence changes in the behavior of a particular resource might indeed have an impact on the already computed *exact* state-space representing its output behavior, but might not have an impact on the over-approximated output behavior of the resource. This can be exploited by our impact analysis.

We consider event streams as the maximal abstraction of the timing behavior of a task, as these only contain information about best- and worst-case response-times, without any information, in which cases the corresponding response times occurs. For example a task could have a large response time when it is interrupted by an high priority task which is allocated on the same resource, and an small response time, when no interrupts occur.

Though our analysis approach is an exact analysis in general, it can be combined with abstraction techniques in order to reduce the state space of the interface transition systems. Such abstractions were the topic of our previous works [18].

An abstraction indeed might affect the schedulability of a depending resource, and hence may cause false negative results. On the other hand, suitable abstraction techniques may pave the way to omit re-validations.

5 Evaluation

In this chapter we will evaluate our methodology by the usage of the three test systems illustrated in Figure 4. Tasks with no input edge are considered as to be independent, i.e. triggered by event streams. The scheduling policies of each ECU is fixed priority with interruption, and the policy of the CAN bus is also fixed priority but (of course) without interruption. The parameters of the tasks are detailed in the table of Figure 5, where p is the period of a task, *ecec.* is the execution time which may be a single value or an interval, if $bcet \neq wcet$, and pr is the priority of a task.

In our evaluation we compare the time needed for an analysis of each resource and the times needed to store and load a corresponding state space, and check

	System 1			System 2			System 3		
	p	*exec.*	*pr*	*p*	*exec.*	*pr*	*p*	*exec.*	*pr*
t1	60	35	1	55	[4,10]	1	50	[4,10]	1
t2	5	2	2	50	[3,7]	2	50	[3,7]	2
t3	-	4	2	-	[2,6]	2	-	[2,6]	2
t4	60	12	1	-	[4,5]	1	-	[4,5]	1
t5				-	[3,5]	2	-	[3,5]	2
t6					[2,3]	1	-	[2,3]	1
t7							50	[3,5]	1
t8							50	[3,5]	2
t9							-	[4,5]	3

Fig. 5. Task parameters.

the refinement of the state spaces of the resources. The idea is to demonstrate that the analyses times of the resources is always much larger than the times needed to store and to load the state spaces, and to check whether – if a change occurred – the old state space is a refinement of the new one.

Note that all times were measured on the same machine to preserve comparability. Each check has been performed five times. The times illustrated here are the average times of all measurements.

The measured times are illustrated in the table of Figure 6. As an example: To analyze the timings of $ECU2$ of $System2$ we need $6,75$ seconds. In contrast to this, the refinement check of the of state spaces (new and old) of $ECU2$ only takes $0,015$ seconds. The cell *Sum* is the sum of the cells *Refinement, Load* and *Store* and is used to compare the times needed to perform these three steps against the plain verification time.

As an example we illustrated some cases graphically in Figure 7.

The result of our evaluation is that the larger the state space of a resource is (and therefore the verification time of that resource), the larger the difference between the verification time and the computation times needed to load, save, and check the refinement of the old and new state spaces is. Thus, for larger systems our refinement methodology is a real gain for our analysis approach. Note that of course, if the refinement check fails, i.e. the new state space of a resource is not a refinement, than we have extra analysis times which we would not have if we always perform the plain verification directly. But fortunately these refinement checking times are not that large. Actually the complexity of the refinement check is $n(n-1)$ where n is the number of states.

6 Conclusion and Outlook

We illustrated the implementation of our impact analysis approach which is applied when architectural changes occur during the design state of a system. We evaluated our approach by measuring the computation times needed to perform

		ECU1	ECU2	CAN	ECU3
System 1	Analysis	0,16	6,33		
	Refinement	0,00016	0,0138		
	Load	0,06	1,29		
	Save	0,04	1,4		
	Sum	0,10016	2,7038		
System 2	Analysis	0,09	6,75	0,58	
	Refinement	0,0001	0,015	0,013	
	Load	0,04	0,92	0,17	
	Save	0,03	1,07	0,17	
	Sum	0,0701	2,005	0,353	
System 3	Analysis	0,01	0,03	15,31	77,52
	Refinement	0,0001	0,0001	0,12	0,31
	Load	0,0001	0,009	3,61	10,81
	Save	0,0001	0,0001	3,78	11,4
	Sum	0,0003	0,0092	7,51	22,52

Fig. 6. Measured average computation times.

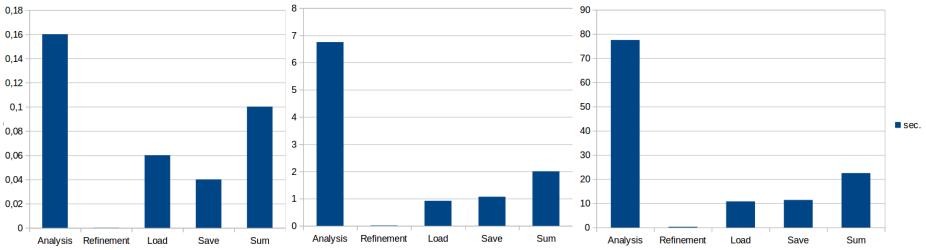

Fig. 7. Computation times for a) $ECU1$ in $System1$ (left), b) $ECU2$ on $System2$ (center), and c) $ECU3$ on $System3$.

the full verification, the storage and load of state spaces, and the computation of the refinement check, and compared these times. The result is that for larger systems our refinement methodology is a real gain for our analysis approach. Currently, we investigate new abstraction techniques which will yield more accurate results than the classical analysis techniques and will boost the scalability of our approach.

References

1. Gezgin, T., Stierand, I., Henkler, S., Rettberg, A.: State-based scheduling analysis for distributed real-time systems. Design Automation for Embedded Systems (2013) 1–18

2. Gezgin, T., Henkler, S., Stierand, I., Rettberg, A.: Impact analysis for timing requirements on real-time systems. In: Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on. (Aug 2014) 1–10
3. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. Microprocess. Microprogram. **40** (April 1994) 117–134
4. Gutierrez, J., Gutierrez Garcia, J., Gonzalez Harbour, M.: On the schedulability analysis for distributed hard real-time systems. In: Proc. of the Euromicro Workshop on Real-Time Systems. (1997) 136–143
5. Thiele, L., Chakraborty, S., Gries, M., Maxiaguine, A., Greutert, J.: Embedded software in network processors - models and algorithms. In: Proc. of the First Workshop on Embedded Software. EMSOFT, London, UK, Springer (2001) 416–434
6. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R.: System level performance analysis - the symta/s approach. Computers and Digital Techniques, IEE Proceedings **152** (2005) 148–166
7. Wandeler, E.: Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems. PhD thesis, Swiss Federal Institute of Technology Zurich (2006)
8. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In: EMSOFT '09: Proc. of the seventh ACM international conference on Embedded software. (2009) 107–116
9. Phan, L.T.X., Lee, J., Easwaran, A., Ramaswamy, V., Chen, S., Lee, I., Sokolsky, O.: Carts: A tool for compositional analysis of real-time systems. SIGBED Rev. **8**(1) (March 2011) 62–63
10. David, A., Illum, J., Larsen, K.G., Skou, A.: Model-based framework for schedulability analysis using uppaal 4.1. In Nicolescu, G., Mosterman, P., eds.: Model-Based Design for Emb. Systems. (2009) 93–119
11. Hendriks, M., Verhoef, M.: Timed automata based analysis of embedded system architectures. In: Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. (April 2006)
12. Perathoner, S., Wandeler, E., Thiele, L., Hamann, A., Schliecker, S., Henia, R., Racu, R., Ernst, R., Harbour, M.: Influence of different system abstractions on the performance analysis of distributed real-time systems. In: Proceedings of the 7th ACM & IEEE int. conference on Embedded software. EMSOFT (2007) 193–202
13. Fersman, E., Pettersson, P., Yi, W.: Timed automata with asynchronous processes: schedulability and decidability. In: Proceedings of TACAS, Springer (2002)
14. Benveniste, A., Caillaud, B., Ferrari, A., Mangeruca, L., Passerone, R., Sofronis, C.: Multiple viewpoint contract-based specification and design. In: Formal Methods for Components and Objects. Volume 5382. Springer Berlin Heidelberg (2008)
15. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: IEEE International Symposium on Circuits and Systems (ISCAS). Volume 4. (2000) 101 –104 vol.4
16. Lehnert, S.: A review of software change impact analysis. Ilmenau University of Technology, Tech. Rep (2011)
17. Richter, K., Racu, R., R.Ernst: Scheduling Analysis Integration for Heterogeneous Multiprocessor SoC. In: Proc. RTSS. (2003)
18. Gezgin, T., Henkler, S., Stierand, I., Rettberg, A.: Evaluation of a state-based real-time scheduling analysis technique. In: Industrial Informatics (INDIN), 2014 12th IEEE International Conference on. (July 2014) 158–163

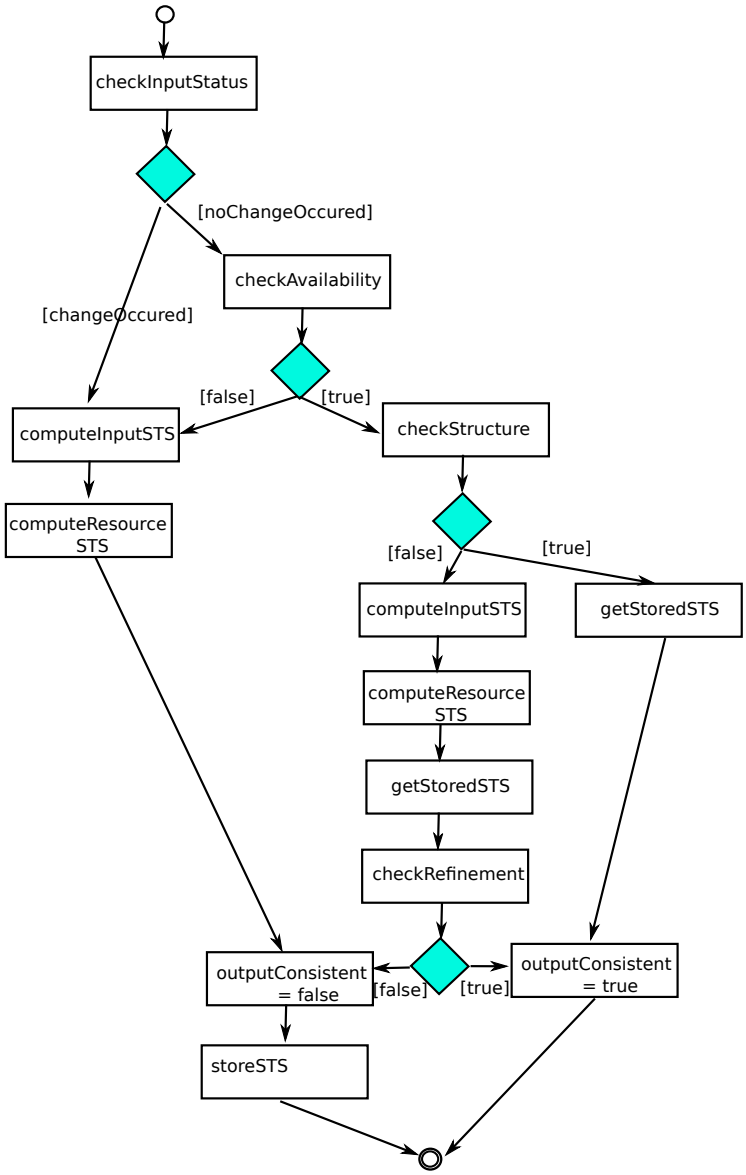

Fig. 8. Methodology of the Impact Analysis (timing analysis combined with refinement check).