Universiteit
Leiden
The Netherlands

Modelling and analysis of real-time coordination patterns

Kemper, S.

Citation
Kemper, S. (2011, December 20). *Modelling and analysis of real-time coordination patterns*. *IPA Dissertation Series*. BOXPress BV, 2011-24. Retrieved from https://hdl.handle.net/1887/18260

Version:	Corrected Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/18260

Note: To cite this publication please use the final published version (if applicable).

Chapter 1

Introduction

Embedded software systems are nowadays found everywhere: in smartphones, in cars and aeroplanes, even in coffee machines. Naturally, with an increased number of software systems comes an increased number of observable system failures. While for some systems, a system failure is not dangerous at all but merely annoying—for example, if no coffee can be prepared—failure of other systems has safety critical or even life threatening consequences—for example, an airbag failure or an aeroplane crash. For this reason, it is important to verify that the system works correctly and safely before it is being put into operation.

Verification of such systems is a difficult task by itself already, but two features of present-day software systems complicate this task even more. First, due to the increasing size of software systems, they are developed in a modular, *component-based* way, such that the final system can be distributed over several (logical and/or physical) locations. Second, to be able to truthfully express real-life situations, and to specify lower and upper safety bounds on the occurrence of events—to ensure that things happen at the right moment—systems involve handling of *real-time*. We now look at these two aspects in a little more detail.

Component-based software engineering and system design amounts to constructing large systems by composing individual components. That is, (typically at beginning of the design phase) the behaviour of the system to be developed is split into logical units, each encapsulating a certain behaviour or aspect of the system. These units can then be developed in parallel and independently of each other. In the end, the final system is obtained by composing these individual components. Naturally, the correctness and safety of concurrent systems developed in this way not only depends on the correctness and safety of the individual components, but also (even more) on correct inter-component communication actions: even if the components are correct—that means, they correctly implement the intended behaviour of the respective logical unit—the final system cannot work as intended if the components exchange inappropriate information or communicate with the wrong partner. For example, consider an aeroplane controller that intends to request the status of

the fuel level sensor, but instead communicates with the cabin pressure sensor. In case the cabin pressure is fine, but the fuel level is low, this wrong connection of communication partners can have lethal consequences.

Another problem arises from the fact that when designing a new system, not all components are/need to be developed from scratch. Instead, component-based software engineering allows for reusing components from other systems, and for simple or recurring tasks, even off-the-shelf components exist. Unfortunately, such reused components are often only available as black boxes. As a consequence, to correctly interlink the components, component connectors are required that provide exogenous coordination, i.e., coordination from without [Arb98]. As such, these component connectors require true concurrency in time, which combines synchrony and asynchrony, to express complex coordination patterns.

The major advantages of the component-based approach are the possibility to develop the system parts in parallel and deal with larger systems (scalability), and the increased flexibility to exchange or modify components of the system, without having to modify or even inform/touch the other components (modularity). The major challenge for the verification of component-based systems is to adapt and scale the verification techniques to large and distributed systems.

To guarantee correct behaviour of the system, in addition to the above, the communication between components not only needs to happen correctly with respect to exchanged information and involved components, but equally important needs to happen *at the right time*. For example, a warning message indicating a low fuel level is useless if it is displayed only after the fuel tank is completely empty. Yet, "at the right time" does typically not mean "as fast as possible", but involves both upper and lower time bounds. For example, an airbag must not deploy immediately, but with the correct delay (a couple of milliseconds), to properly slow down the forward movement of the occupant at the right moment.

To develop component-based real-time systems which are correct—that means, behave as expected—and safe—that means, nothing bad can ever happen—essentially two things are needed. First, a formal model to describe the system components. This formal model should be powerful enough to faithfully describe all aspects of the system, and yet simple enough to allow being used in industrial practice by non-experts in formal methods . Second, formal methods are needed to analyse the system model before it is being put into operation, and to verify that it satisfies certain requirements.

In this thesis, we aim at solving the problem described above. To this end, we present formal models to specify real-time coordination patterns, and define formal methods to analyse and verify these formal models, as well as formalisms to further increase the manageable system size. We present our tool implementation featuring a graphical editor, that supports users in the development and design process.

1.1 Contents and Structure of this Thesis

The contents of this thesis are structured as follows.

Chapter 2 (System Models) We start with the formal models. In this Chapter, we define a series of three automata-based system models. We start with the (among the three) simplest model of Timed Automata [AD94, Alu99], followed by Timed Constraint Automata [ABdBR07, Kem11] and Timed Network Automata [Kem10]. The formal models are presented based on increasing modelling power with respect to communication. For each model, we discuss advantages and disadvantages, including for example what kind of constraints on (inter-component) communication can be imposed with the respective model, and what kind of constraints cannot be imposed. Moreover, for each of the formal models, we define a formal semantics, and present the product construction needed to compose larger systems.

Chapter 3 (SAT-based Verification) In this Chapter, we show how to use formal methods to analyse and verify properties of the system models from Chapter 2. We start by defining a translation into propositional logic with linear arithmetic for each of the system models. This formula representation allows to apply well-established model checkers to examine the systems and verify certain properties. In particular, we show how to apply the technique of Bounded Model Checking [CBRZ01, BCC$^+$03] to our representation, show how to express certain common-type properties, and we discuss correctness and completeness issues for Bounded Model Checking. In the end of Chapter 3, we provide an in-depth discussion about other possible translations, what influence these (would) have on the efficiency of verification, and in this way motivate and justify our design decisions.

Chapter 4 (Abstraction Refinement) In this Chapter, we adapt the principles of Abstraction Refinement [CGJ$^+$03, HJMM04], such that they can be used in combination with the formal methods presented in Chapter 3. These techniques allow to increase the size of systems to be analysed, by removing system parts which are considered irrelevant for the current examination or property under test. In particular, we consider a variant of Counterexample Guided Abstraction Refinement [CGJ$^+$03], where information from so-called spurious counterexamples (counterexamples to the property under test that only exist in the reduced system) is used to identify the reason why verification has failed. Such information is extracted from spurious counterexamples with the help of Craig interpolants [Cra57]. We briefly refresh the principles of Craig interpolants, show what kind of information can be obtained from these, how to modify the input to maximise the information, and in the end of the Chapter present techniques to prevent the spurious counterexample from happening again.

Chapter 5 (Tool Development and Application to Case Studies) The theoretical results obtained in this thesis have been implemented and integrated into an existing tool suit for component-based systems. Chapter 5 is devoted to tool development. We start by presenting the details of our implementation in the context of the existing tool suite. We then show how to actually use the tool, by giving a little workflow example, which can also be seen as a "getting started" tutorial. Finally, we show the applicability and performance of our approach and tool on two small case studies.

Chapter 6 (Conclusions) In this Chapter, we wrap up the results, and discuss some directions for future research.

Appendix In the Appendix, we give correctness results for the formula representation presented in Chapter 3, and the abstraction function presented in Chapter 4.

1.2 Origin of Material and Main Contributions

Some parts of the research presented in this thesis have been published previously, or have been included here for completeness. Other parts contain new, original research which has not been presented elsewhere. We now give an overview for each of the Chapters, and sum up the main contributions.

Chapter 2 (System Models)

Timed Automata have first been introduced in [AD94, Alu99]. Since they come in many different variants (most of them with only subtle differences), in Section 2.2 we have outlined the exact nature of Timed Automata that we use in this thesis, without changing the formal model itself.

In Section 2.3, we extend the formal model of Timed Constraint Automata from [ABdBR07] (which has also been used in [Kem11]) with location memory. Location memory for (Untimed) Constraint Automata [ABRS04] has been defined in [PSHA09], but this is the first work on defining a formal syntax (including a product construction) and semantics for the combination of Constraint Automata with time *and* data. The new Timed Constraint Automaton model is an extension of the model in [ABdBR07] and therefore extends the set of coordination patterns that can be expressed, in that coordination patterns can not only reason about data values received in the current step, but also about data values received in previous steps. Timed Constraint Automata can—to a certain extend—be seen as an extension of Timed Automata [AD94, Alu99] with data.

In Section 2.4, we extend the formal model of Timed Network Automata, as presented in [Kem10], with memory cells and concrete data values, and define a formal syntax and semantics. In particular, we define a product construction that uses data variables to preserve information about data values contained in memory cells. Timed Network Automata were inspired by the three-colouring idea for $\mathcal{R}$eo networks [CCA07], but lift the idea of considering environmental constraints to the automaton level. Since the extended Timed Network Automata defined in this thesis features the same concepts for data handling as Timed Constraint Automata (that is, ports and memory cells), they can be seen as an extension of Timed Constraint Automata with environmental constraints.

Summarising, this Chapter contains the following main contributions:

Contribution 1: memory cell extension of Timed Constraint Automata, with corresponding product construction and formal semantics

Contribution 2: definition of Timed Network Automata, with formal syntax, semantics and product construction

extension of the basic Timed Network Automaton model with memory cells and data-awareness, with corresponding formal syntax, semantics and product construction

Chapter 3 (SAT-based Verification)

The formula representation of Timed Automata presented in Section 3.1.2 has been published in [KP07]. Subsequent to the extensions of Timed Constraint Automata in Section 2.3 and Timed Network Automata in Section 2.4, in Sections 3.1.3 and 3.1.4 we extend the formula representations (in propositional logic with linear arithmetic) of Timed Constraint Automata (presented in [Kem11]) and Timed Network Automata (presented in [Kem10]) accordingly. After that, we discuss completeness issues of Bounded Model Checking in Section 3.2, which have only been briefly sketched in [Kem11], and extend the correctness proof to the extended representation (note that the actual proofs are contained in the appendix). Finally, we give an in-depth discussion about the quality of the representation with respect to speed of verification, which has not been published before.

Summarising, this Chapter contains the following main contributions:

Contribution 1: application of SAT-based verification in the process of component-based software engineering and to formal models of component-based software

Contribution 2: adaptation of the formula representation of Timed Constraint Automata and Timed Network Automata to the extended models of Chapter 2

Contribution 3: in-depth discussion of the quality of representation, including advantages and disadvantages of different translation options

Chapter 4 (Abstraction Refinement)

The abstraction function we present in Section 4.1 is essentially equivalent to the abstraction function presented in [Kem11], but we show that it works uniformly for both Timed Automata and Timed Constraint Automata, and that correctness is preserved for both formal models (again, the actual proof is contained in the appendix).

We restate the principles of Craig interpolation [Cra57] for the sake of completeness. After that, we provide a detailed discussion about the expressiveness of interpolants, the possibilities to influence expressiveness, and refinement options and heuristics, all of which has only been briefly and vaguely sketched in [KP07, Kem11].

Summarising, this Chapter contains the following main contributions:

Contribution 1: application of abstraction refinement in the process of component-based software engineering and to formal models of component-based software

Contribution 2: generalisation of the abstraction function and correctness results

Contribution 3: in-depth discussion of interpolation and refinement, to provide a deep understanding of the underlying mechanisms

Chapter 5 (Tool Development and Application to Case Studies)

We present the implementation that, in the context of this thesis, has been developed as part of the Extensible Coordination Tools [ECT],[1] a tool suit for component-based systems. In Section 5.1, we present the details of the implementation in a comprehensible way, which makes it easy to understand the overall structure and functioning, and provides the basis for future extensions of the tool. Section 5.2 demonstrates the applicability of the approach, by giving a workflow example/"getting started" tutorial. We finish with two little case studies.

Apart from Section 5.1.1 (which provides the overall context for the description of our implementation) and Section 5.3.1 (which has been published as part of [Kem11]), all results presented in this Chapter describe original work that has not been published before.

This Chapter contains the following main contribution:

Contribution: extensive tool-support for modelling and analysis of real-time coordination patterns implemented with Timed Constraint Automata

[1] See `http://reo.project.cwi.nl/`