Modelling and analysis of real-time coordination patterns

Kemper, S.

Chapter 6

Conclusions

Component connectors that implement real-time coordination patterns are an essential ingredient of component-based software engineering. They are needed to connect, coordinate and orchestrate the distributed components of large real-time systems, and in this way guarantee correct and safe behaviour of the entire system. Adaptation (and extension) of such systems to different needs is facilitated by the encapsulated modular nature of the component connectors, which allows to replace a coordination pattern by another, unnoticed by the other parts of the system. In this thesis, we have established a formal framework for exhaustive modelling and analysis of real-time coordination patterns, with a focus on the formal model of Timed Constraint Automata (TCA) with Data.

In Chapter 2, we have started with the formal definition of the system models, intended to be used to model the real-time coordination patterns. Each of the models is suited to model certain classes of real-time coordination patterns, depending on whether the pattern takes into account data, true concurrency of actions or environmental constraints. We present the models with increasing complexity and modelling power based on these features: Timed Automata (TA) [AD94, Alu99] are well-known (and well-studied), but lack the ability to handle data and true concurrency. As a consequence, we have extended the formal model of TCA [ABdBR07, Kem11] (handling true concurrency) with memory cells to handle data. To incorporate environmental constraints (for example, whether the environment in which a system operates is ready to communicate), we have finally introduced the formal model of Timed Network Automata (TNA) with memory cells and data (which allow for true concurrency as well) as an extension of the model presented in [Kem10]. The intuitive visual representation of each of the models (corresponding to a certain extend to mental models of such systems) allows to quickly and easily sketch and develop systems with the formalism.

In Chapter 3, we define a representation in propositional logic with linear rational arithmetic for each of the system models from Chapter 2. We try to keep the representation as general as possible, by using only those variable types and oper-

ations that are compulsory to faithfully represent the underlying formal model. In this way, the representation can serve as an intermediate format and can be translated into the input language of many common SMT-solvers, which in turn can be used to analyse the underlying system. We have discussed how to apply the technique of Bounded Model Checking to the formula representation, and have presented correctness and completeness results.

A major challenge in component-based software engineering is the fact that the systems to be analysed are getting bigger and more complex, while correctness and safety still need to be guaranteed. In Chapter 4, we have presented an approach to (partially) solve this problem: we have defined an abstraction function that can be used to reduce the size of the system representation. The abstraction function works on the formula representation from Chapter 3, and removes parts that are considered irrelevant to the verification of a particular property. We have shown how to undo part of the abstraction in case it has turned out to be too coarse, based on information obtained from so-called spurious counterexamples (counterexamples to the property under test that only exist in the abstract system). We have provided correctness results proving the abstract system to be an over-approximation of the original system.

Theoretical results can only be beneficial if they can be used and applied in practice. In Chapter 5, we have presented in detail the available tool support for the theoretical framework developed in the preceding Chapters (as of now, the implementation only supports the formal model of TCA). Our implementation of the TCA plugin is part of the Extensible Coordination Tools [ECT], an integrated graphical development environment for the Eclipse [Ecl] platform that supports modelling and analysis of component-based systems. We have provided detailed information about the implementation structure of both our plugin and the ECT in general, and we have sketched the typical workflow when using the tool. In this way, Chapter 5 can be seen as a "getting started" tutorial for users, and at the same time as a reference manual for developers. In the end of the Chapter, we have presented two case studies (including experimental results), to show how TCA can be used to model protocol coordination in a concise and understandable way, and we have discussed the benefits of using TCA, for the case studies in particular and for real-time coordination patterns in general.

6.1 Future Directions

In parts of this thesis, some questions have remained unaddressed. For other parts, we see obvious directions for future research and extensions of the work. We now give an overview of the most important points.

In Section 5.3.3, we have discussed the disadvantages of TA when it comes to concurrent execution of actions, and the advantages of TCA regarding this aspect. However, there are situations where a notion similar to committed locations in TA (remember that a committed location in a TA must be left immediately, without delay or interleaving of other actions, cf. Footnote 14 on Page 120) could be beneficial also for TCA: suppose a committed location in a TA, with n ingoing transitions

and m outgoing transitions, all with disjoint transition labels. To model the same behaviour with a TCA, we would need $n*m$ transitions (while the TA only has $n+m$ transitions and one location). We consider it interesting to investigate how a notion similar to committed locations can be defined for TCA, and whether there is a noticeable improvement on performance when it comes to verification.

The model checker Vereofy (`http://www.vereofy.de`) provides tools for model checking (untimed) Constraint Automata (CA). As input, it accepts amongst others CARML (Constraint Automata Reactive Module Language [BBKK09]), which is a textual description language for CA. Extending CARML with the notion of time would yield a textual description language for TCA, which for example could be used as an exchange format for TCA specifications between different tools.

Untimed CA where originally defined as a semantical model for the channel-based coordination language $\mathcal{R}$eo. Consequently, TCA were defined to serve as a semantical model for timed $\mathcal{R}$eo, yet TCA and timed $\mathcal{R}$eo have by now to a certain degree evolved independently of each other. In particular, timed $\mathcal{R}$eo is still restricted to the definitions presented in [ABdBR07]: timed behaviour is incorporated in $\mathcal{R}$eo through special timer channels. These accept any (data) type of input, delay for a certain amount of time, and then emit a special "timeout" signal. In addition, some timer channels can be stopped and/or restarted. It is obvious to see that TCA are more powerful. However, now that syntax and semantics for TCA with data have been defined formally, it should be straightforwardly possible to establish the link between TCA and timed $\mathcal{R}$eo. This of course includes extending the implementation as well: in ECT, timer channels for $\mathcal{R}$eo are already supported, and a translation from $\mathcal{R}$eo to CA already exists. Similar to this translation, a translation from timed $\mathcal{R}$eo to TCA can be implemented once the formal basis for this has been established.

The time that is needed for verification may be a bottleneck for very large systems. For this reason, we consider it worth to investigate how other SAT and SMT solvers perform on our formula generation. A drawback however is the fact that at the moment, there exist only very few SMT solvers that can generate interpolants for unsatisfiable SMT problems. For this reason, it might be worth to decouple interpolant generation from SMT solving. A tailored (re)implementation of interpolant generation would allow to optimise the generation algorithm such that it always generates the strongest interpolant (the notion of *the* strongest interpolant is well-defined, and the strongest interpolant can be computed iteratively, cf. for example [EKS06]). Another approach to decrease verification times is to investigate how the formula order of the input problem influences the performance of different SMT solvers (for example, whether variable valuations of the witness run occur at the beginning or at the end of the input sequence, or are distributed throughout the sequence).

We expect the largest potential for future research in the field of abstraction refinement. A straightforward extension of the work presented in this thesis is to extend the abstraction function to TNA. Assuming that ports in TNA can be merged just like ports in TCA, questions that need to be solved before an abstraction function for TNA can be defined include (1) how to define the colouring of merged ports, and (2) whether read and write ports can be merged, or (if this should not be possible) how to avoid this. From a more practical point of view, for most parts

of the formula representation we expect the abstraction function for TNA to work in the same way as it does for TCA. The only exception to this are port colour variables: the explanations after Definition 4.1.5 (positive propositional variables are used to describe the behaviour, negative propositional variables are used to ensure consistency) do not apply to port colour variables, which means they cannot be handled in the same, uniform way (only based on syntactical categories) as other variables. A possibility to solve this is to impose additional constraints on the set of omission and map of merging for TNA, in a similar way as has been done for data constraints involving merged ports in TCA (cf. Definition 4.1.3).

Apart from extending the abstraction function to other system types (like TNA), it is also possible to extend it with other concepts of abstraction. By concept, we mean the way how information is removed from the system. The abstraction function currently features two concepts: merging and omission. Examples of other concepts include weakening, and (variants of) abstract interpretation. By weakening, we mean to replace a constraint by a weaker variant instead of completely removing it. For example, replace a constraint $x<5$ by $x<10$ (note that convexity is required for this to work for compound constraints). We expect this concept to be applicable to clock constraints. In abstract interpretation (cf. for example [CC77, CC92]), a large (possibly infinite) set of concrete values is restricted to a smaller, finite set of abstract values, using a suitable abstraction function. For example, the set $\mathbb{Z}$ of integer numbers could be abstracted to the set $\{-1, 0, 1\}$, by mapping all negative integers to -1, all positive integers to 1, and 0 to 0. Abstract interpretation resembles our idea of abstraction by merging, and we expect it to be applicable to the domain of data values of TCA and TNA.

Finally, a question that has not been addressed in this thesis is how to determine the initial abstraction automatically, and how to automate the refinement step. We rely on human experts to provide the initial abstraction and decide which parameter to refine. As an intermediate step towards automatic abstraction refinement, the initial abstraction could be determined automatically based on the property to be verified. A related approach can be found in [CGKS02, CCK$^+$02], where the initial abstraction leaves the parameters contained in the property in the system, and removes all other parameters. In [MA03], the initial abstraction is obtained from a proof of unsatisfiability (in the original system, for some small unfolding depth). We believe these approaches can be adapted to the work presented in this thesis, but it should be one of the major goals for future research to aim towards a framework for fully automatic abstraction refinement.