

Towards the Incremental Model Checking of Complex Real-Time UML Models

Martin Hirsch and Holger Giese*
Software Engineering Group
University of Paderborn
Warburger Str. 100
D-33098 Paderborn, Germany
[mahirsch|hg]@upb.de

ABSTRACT

Today, the verification of complex distributed embedded real-time systems employing model checking is largely limited by the state explosion problem. We first report on the current tool support for an approach which addresses this problem by means of a compositional model checking approach for a pattern and component based UML 2.0 designs. However, the current checking covers only an abstraction of the employed Realtime Statechart semantics (cf. [4, 9]), and the compositional approach only works for properties which refer either to a single pattern or a single component. We then present plans for an improved tool support which supports the full Realtime Statechart semantics, enables the compositional checking of non-local properties, and a model checker integration which triggers required checks after a model update automatically in the background.

Keywords

Model Checking, UML, Real-Time, Embedded Systems, Mechatronic Systems, Compositional Verification, Fujaba.

1. INTRODUCTION

Mechatronic components [6], which beside their local control of equipment are also interconnected with each other, result in a complex distributed embedded real-time system. As such mechatronic systems often contain real-time and safety-critical requirements, a proper approach for the real-time and safety analysis is mandatory (cf. [11]). The worst-case real-time characteristics w.r.t. deadlines must be predictable, and appropriate means for the validation and/or verification are required. Thus, the engineer can judge whether the resulting system still contains safety hazards. However, the verification of these systems by means of model checking is often not possible today due to the state explosion problem.

In this paper we first report on the current tool support for an approach which is addressing this problem. It is restricted to the specific case of software controlling mechatronic systems (cf. [10]). It uses a domain specific pattern and component based approach that employs a subset of the UML 2.0 component model. The complex software systems are composed of domain-specific patterns. These patterns differ to some extent patterns introduced in standard literature e.g.[7]. In [7] patterns are each identified by classes, associa-

*This work was developed in the course of the Special Research Initiative 614 - Self-optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and founded by the Deutsche Forschungsgemeinschaft.

tions etc. In our context the considered patterns are also described by their ports resp. port-roles. In this manner using/instantiating patterns we have to deal especially with the port behaviour. Each of these patterns can be verified individually. The complete component behavior is derived by a composition of these patterns. For a syntactically correct composition the verified pattern and component properties are also guaranteed for the resulting overall system behavior. However, currently only an abstraction of the Realtime Statechart behavior can be checked and properties, which can be compositionally verified, must be locally defined either for a single pattern or a component.

A first required step is to transform the Realtime Statecharts into the format of the model checker in a way that all aspects of the semantics are covered. Then, the result of the model checking will result in less false negatives, because a less abstract model, which reflects the real behavior better, can be checked.

To address non-local properties, the approach can currently employ the compositional nature of the original approach and check a proper subset of the overall system. We propose to automatically determine such proper subsystems to incrementally check whether a property holds by using a series of subsystems with increasing model size. If from the verification follows that the property holds, the compositional nature ensures that the property also holds for the complete system.

Finally, we can improve the model checking support by integrating the automatic checking of properties if any update has happened in the related UML model. Due to the compositional nature each single checking can usually be done in the background in an incremental fashion.

This paper is organized as follows. In Section 2 the employed pattern-based approach for the modelling of real-time systems with UML is sketched, and the related compositional approach for model checking the resulting UML models is described afterwards in the remainder of this section. Then, we describe the currently in Fujaba realized tool support (Section 3). To support checking non-local properties, we review in Section 4 the shortcomings of the current tool support as well as ideas which permit to incrementally extend the checked model until the property holds. In Section 5, the resulting requirements for realizing these concepts are summarized. The paper finishes with a final conclusion.

2. REAL-TIME UML MODELLING

In this section, we first describe the main elements such as patterns, connectors and components, which are used when modelling real-time systems with UML. In the second subsection, we explain the design steps which are necessary to get a safe real-time system.

2.1 Basis Elements

In our approach a pattern comprises of a set of roles that interact only via a connector. Every connector connects the related component ports in the final system. In the instantiated system the ports refine the roles and the components synchronize the ports. We further have the restriction that for each pattern we have to specify a protocol automaton and invariants for each role. An overall constraint in form of e.g. a TCTL formula is also possible. While usually in untimed models the connector behavior is omitted, channel delay is of crucial importance for real-time systems and thus has to be addressed explicitly in form of an additional connector automaton.

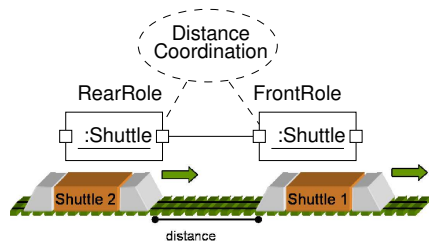


Figure 1: Modelling Example

Figure 1 shows an cut-out [10] of the convoy coordination between shuttles required for the software for the railcab research project¹. The railcab system employs a passive track system and intelligent shuttles that operate autonomously and make independent and decentralized operational decisions. The vision of the railcab project is to combine the comforts of individual traffic, such as flexible scheduling, on-demand availability of transportation and individually equipped cars, with the cost and resource effectiveness of public transportation. The control infrastructure of the shuttle-based transportation system is based on satellite positioning and a wireless communication network that enables communication between shuttles and stationary installations.

In our example the "DistanceCoordination" pattern realizes the two roles "FrontRole" and "RearRole" which denote the positions in a convoy. The two roles are connected via a connector representing the communication network in the shuttle system. Components like the "Shuttle" component are designed by coordinating and refining each role automaton. The refinement has to respect the role automaton and additionally has to respect the guaranteed behavior of the roles in form of its invariants. An additional internal Realtime Statechart [9] for coordination is used to describe the required coordination. In the context of the example in Figure 1 this means that the shuttle must confirm to the "DistanceCoordination" pattern and has to operate as both "RearRole" and "FrontRole".

2.2 Design Steps

Because of the results and semantic definitions from [8, 10] we have the following sequence of integrated design and verification activities organized into the following 5 steps:

1. design the pattern and their roles,
2. verify each pattern,
3. design the composition,
4. verify each component, and
5. compose the system using the components and patterns.

Note that steps 1 and 2 have to be repeated for each required pattern. When step 3 and step 4 have already been performed with incomplete sets of patterns, the additional roles have to be added

¹<http://www-nbp.upb.de/en/index.html>

to the component automata. Step 5 finally ensures correct semantical composition by a correct syntactical composition. An additional 6th step to perform verification for the overall system after the composition made in step 5 is thus not necessary. The latter result is proven in [10]. This theorem is only valid for local properties for patterns or components.

3. CURRENT TOOL SUPPORT

So far there exists a version of the model checking plugin called UMLRTModelchecking for the CASE tool Fujaba [1] developed in the context of SHUTTLE PG project [2] at the University of Paderborn. This section describes its architecture.

3.1 Architecture

In SHUTTLE PG already some plugins helping us to model real-time systems with UML-RT were developed. Figure 2 shows an

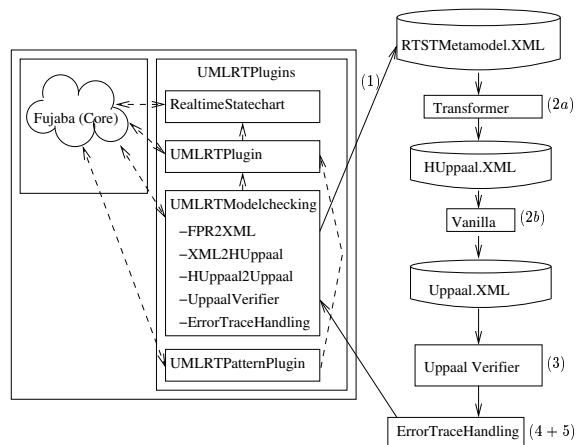


Figure 2: Plugin Architecture

overview of the plugins that are currently available. The UMLRT-Plugin, which allows us to model Realtime Component Diagrams and generate JavaRT code, provides an own UML-RT metamodel. The Realtime Statechart plugin [4] is also integrated. It is used to model the role protocols and the component behavior. All other plugins depend on the UMLRTPlugin. The pattern plugin, named UMLRTPattern, enables us to identify patterns within the UML-RT model and to manage them in an own repository. It is further possible to select patterns from the repository and add them to the UML-RT model. Finally, there is a plugin named UMLRTModelchecking which enables the user e.g. to prove properties specified in the UML model as TCTL-formulas. In the following, this plugin is described in more detail.

3.2 Model Checking PlugIn

As seen in Figure 2 the UMLRTModelchecking plugin requires the UMLRTPlugin and Realtime Statechart plugin. Further the plugin employs the real-time model checker UPPAAL [3] to verify whether the properties, specified in the UML model, are fulfilled. It is not necessary to add the whole model checker; only the verifier engine is required. Since the plugin execution is arranged in 5 steps, it is easy to use another real-time model checker like RAVEN [12] instead of UPPAAL by only making some changes.

The following paragraph describes the steps the existing plugin performs to check a UML model. (1) In the first step the meta data of a UML-RT model are exported and written to an separated XML-file. If any constraint in form of a TCTL formula has been

added to the UML-RT model, it is also written to the XML-file. If no TCTL-formula has been specified, the constraint "A[] not deadlock", which means checking the model for deadlock freedom, is automatically added to the file. (2a) Having generated this file, the plugin enables the user to transform the UML-RT model to Hierarchical Timed Automata [5]. Since there is no real-time model checker which works directly on the Hierarchical Timed Automata model we have again to perform a transformation. (2b) The generated Hierarchical Timed Automata model is transformed to the flat Timed Automata model by using the tool Vanilla [5]. The steps (2a + 2b) are executed in one step. Together with the TCTL-constraints defined in the original UML-RT model, the flat Timed Automata model can be used as input for the model checker tool UPPAAL [3]. (3) The UPPAAL verifier is automatically started after step (2). (4) The result of the verification and perhaps an error trace, if a property fails, is finally transformed back to UML view. (5) In order to make the result more descriptive, there is a visualization in Fujaba. Step (4) and (5) are currently only rudimentary solved by a text window showing the UPPAAL tool output. At present the transformation from step (2) works only for Realtime Statecharts. Our aim in this first version is to achieve a pessimistic abstraction of the original model. At present, only the hierarchy and the basic structures like transitions, guards and states are supported, where as complex features like priorities are currently not supported. Because of the latter fact, we have much non-determinism in our Hierarchical Timed Automata model and this enormously enlarges the state space. As mentioned before the basic structures are mapped to the basic structures of an Hierarchical Timed Automata as defined in [5]. To realize the do-methods, exit-methods and entry-methods every state in a Realtime Statechart can have, we use a similar mapping as described in [9], where the semantic of Realtime Statecharts is introduced. Another problem is to transform the asynchronous communication model Realtime Statecharts have. In the Realtime Statechart model it is possible to send messages in an asynchronous way, if a transition turns (cf. [9]). We realize this by using the synchronous communication model of Hierarchical Timed Automata and extend it by adding a separate automaton which manages a queue.

4. INCREMENTAL CHECKING

From [8] we know that local properties of patterns and components are preserved by composition and can be checked separately.

It is to be noted that if only the basic components and pattern that depend on the required property are checked, this can result in a false negative. If the verification of properties which involve more than one component is also required (non local properties), we have to slightly adjust the approach. In this case we can exploit the fact that proper subsets of the interconnected components and patterns can be seen as components and patterns again, if the local checks are not sufficient (cf. [8]).

Therefore, we can check a property by the following incremental procedure, shown in Figure 3. In the first step (1) we select a minimal composed pattern/component that contains all in the property referenced elements. After that we check whether the property holds (2). If so, we are done. Otherwise the set of considered elements has to be extended in a way, that the composed pattern/component contains the former one before continuing with step 2). Extending the model we embark on the following strategy. First we allocate all components specified in the selected requirements as mentioned before. If it is necessary to extend this model we first add thus components which interconnect the components selected in the last step, because often those components are involved in the considered context. Otherwise if there is no more

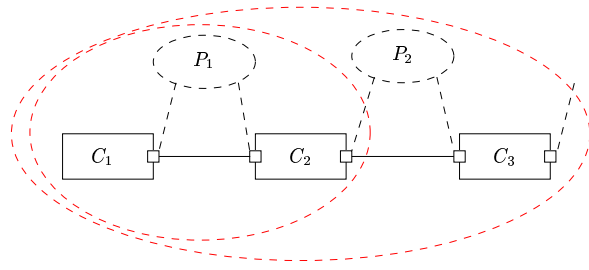


Figure 3: Example for incremental checking

component interconnecting another we select stepwise all neighbours and add them (cf. Figure 3).

5. PLANNED TOOL SUPPORT

In Section 3 we described the current tool support. The planned tool support is described in the following subsections.

5.1 Extended Transformation

Since there is no support for priorities which can be added to transitions in the Realtime Statechart at the moment, we extend the transformation by this concept. This is indeed not trivial, e.g. when transitions consist of synchronization elements, because it is not possible to invert them like guards. But if priorities are supported, we have a more precisely manner to specify our UML model, hence in the Hierarchical Timed Automata model we have less non-determinism. Regarding to the state space using on-the-fly model checking, there is no enlargement as we have without priorities.

5.2 Improved Model Checking

In the current tool support there is no reasonable checking procedure for large models. Based on the idea of the incremental checking presented in Section 4 and the results from [10], we improve our checking procedure.

First, it should be possible to select a subsystem and check whether the specified proper TCTL-formulas are satisfied. It should also be possible to select TCTL-constraints in the original model and then start the verification process. To be sure, that the TCTL-formulas are syntactically correct, we will implement a syntax-checker. These two check procedures are finally combined with the incremental checking.

5.3 Background Checking

At present there is only one checking mode, namely "on request". For the extension we plan to add a "background mode". In detail this means that there will be something like a consistency mechanism. If there is any update in the UML model, the user will be informed and the verifier starts in the background.

6. CONCLUSION

The presented concepts, which the first author will realize within his master thesis, outline how to extend the compositional approach of [10] as well as its current tool support in several ways: (1) The foundation to model checking taking the full Realtime Statecharts semantics into account is sketched, (2) an incremental approach for checking non-local properties with the smallest sufficient subsystem model is presented, and (3) a plugin for the automatic checking of properties in the case of relevant model updates in the background is described.

We expect, that the compositional nature of the employed approach ensures that for such an improved tool support holds that the

size of the models which have to be checked due to model updates is usually rather small. Thus, the proposed background checking will usually be a feasible and convenient solution. However, currently no empirical evidence for this thesis can be presented and larger case studies are required to underpinn it by experimental data.

Acknowledgements

We thank Daniela Schilling for her comments on earlier versions of the paper.

7. REFERENCES

- [1] <http://www.fujaba.de>.
- [2] <http://www.upb.de/cs/ag-schaefer/Lehre/PG/SHUTTLE>.
- [3] <http://www.uppaal.com>.
- [4] S. Burmester. Generierung von Java Real-Time Code für zeitbehafte UML Modelle. Master's thesis, Universität Paderborn, Deutschland, Fachbereich Informatik – Mathematik, Sept. 2002.
- [5] A. David and M. O. Möller. From HUPPAAL to UPPAAL: A transformation from Hierarchical Timed Automata to Flat Timed Automata. BRICS Report Series RS-01-11, University of Aarhus, Denmark, Department of Computer Science, BRICS, Mar. 2001.
- [6] D. Dawson, D. Seward, D. Bradley, and S. Burge. *Mechatronics and the Design of Intelligent Machines and Systems*. Stanley Thorne, Nov. 2000.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [8] H. Giese. A Formal Calculus for the Compositional Pattern-Based Design of Correct Real-Time Systems. Technical Report tr-ri-03-240, University of Paderborn, Germany, Department of Computer Science, July 2003.
- [9] H. Giese and S. Burmester. Real-Time Statechart Semantics. Technical Report tr-ri-03-239, University of Paderborn, Germany, Department of Computer Science, June 2003.
- [10] H. Giese, M. Tichy, S. Burmester, W. Schäfer, and S. Flake. Towards the Compositional Verification of Real-Time UML Designs. In *Proc. of the European Software Engineering Conference (ESEC), Helsinki, Finland*, Sept. 2003.
- [11] D. S. Herrmann. *Software Safety and Reliability : Techniques, Approaches, and Standards of Key Industrial Sectors*. IEEE Computer Press, Nov. 1999.
- [12] J. Ruf. Raven:Real-Time Analyzing and Verification Environment. *Journal on Universal Computer Science (J UCS)*, (1):89–104, Feb. 2001.