Vooduu: Verification of Object-Oriented Designs Using UPPAAL

Karsten Diethers* and Michaela Huhn

Technical University of Braunschweig, D-38106 Braunschweig, Germany
{diethers,huhn}@ips.cs.tu-bs.de,
http://www.cs.tu-bs.de/ips

Abstract. The Unified Modeling Language (UML) provides sequence diagrams to specify inter-object communication in terms of scenarios. The intra-object behavior is modelled by statechart diagrams. Our tool *Vooduu* performs an automated consistency check on both views, i.e., it verifies automatically whether a family of UML statecharts modelling a system satisfies a set of communication and timing constraints given as UML sequence diagrams. The front-end of the tool is implemented as a plug-in for a commercial UML tool. For verifying, statecharts and sequence diagrams are translated to the formalism of timed automata. The tool generates temporal logic queries, which depend on an interpretation status for each sequence diagram. The verification is performed by the model checker UPPAAL. The results are retranslated into sequence diagrams. Thus the formal verification machinery is mainly hidden from the user. The tool was applied to a model of the control software of a robot prototype.

1 Introduction

Model checking has been proved to be a useful technique for the verification of complex system behavior. In a variety of case studies relevant errors were discovered and safety and correctness were improved. A number of advanced model checking tools, optimized for different tasks like the analysis of timing constraints, allow to verify systems of size and complexity that are far out of scope of manual exploration. For all success, model checking is not integrated in many system development processes so far, because it is based on a formal description for the system and the requirements, which is proprietary to the employed model checker. Although many verification tools like UPPAAL[1] provide a user friendly graphical interface, the development and maintenance of two different sets of models, one for system design and one for the verification, are considered not practicable in many industrial projects. Thus we will consequently hide the application of formal verification from the user behind a well accepted, standardized modelling language like the UML (Unified Modeling Language)[2].

* The work of this author was funded by the DFG

[1] http://www.uppaal.com

[2] http://www.omg.org/technology/documents/formal/uml.htm

K. Jensen and A. Podelski (Eds.): TACAS 2004, LNCS 2988, pp. 139–143, 2004.

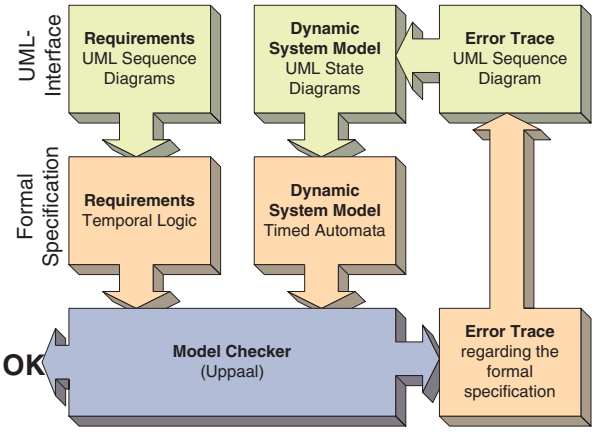

Fig. 1. Tool overview

Our tool *Vooduu* enhances a commercial UML tool by verification functionality. Within the UML, we concentrate on the dynamic view on software because unexpected error configurations caused by complex dynamic runs with timing constraints are one of the most serious problems in the field of real-time applications. These errors are not likely to be found by testing or manual exploration but can be detected by automated exhaustive search of the state space. We use UML sequence diagrams, extended by timing constraints [1], to model requirements and statecharts to model the system behavior.

We support time related constructs in sequence diagrams as well as in statecharts because the timing is crucial in our application domain, which is high speed control processes as used in robotics[3]. Consequently, we employ UPPAAL as model checker because it is tailored to the verification of time dependent systems and supports as well clock as integer variables.

The key functionalities of our tool are the proper transformation of a set of statecharts forming the system model and a set of sequence diagrams as requirements to a model checker and the reinterpretation of the verification results to the UML level in terms of sequence diagrams (Fig. 1).

Related work: Similar to us, [3] uses UPPAAL, too, to verify dynamic UML models but the semantic interpretation of several model elements (e.g. triggerless transitions) differs. In [4], an efficient transformation of statecharts to UPPAAL is implemented but it does not treat the specification of requirements by sequence diagrams. Objects do not communicate via the UML statechart event mechanism but synchronize on actions.

[3] SFB562 *Robotic systems for handling and assembly* (http://www.tu-braunschweig.de/sfb562/)

2 Tool Overview

We implemented our tool as a plug-in for the UML tool *Poseidon for UML*[4] called *Vooduu*[5] (Fig. 2). As a back-end, the model checker UPPAAL is employed. Poseidon provides a graphical user interface for all UML diagrams. UPPAAL is a model checker that provides timed automata for modelling systems and temporal logic expressions for queries [5]. *Vooduu* builds the interface between them.

Figure 1 gives an overview over the tool chain which consists of four components:

- A set of UML statecharts, imported as XML-files, is interpreted as a system model. It is translated independently from the rest of the UML model into a network of timed automata [2].
- Requirements can either be added directly in the UPPAAL environment or - if one prefers to stay on the UML level - in terms of UML sequence diagrams. In the second case, a set of observer automata is generated, which register if and when relevant communication takes place. Erroneous messages and timing violations cause a transition to an error state of an observer automaton. Then the reachability of error states is checked by corresponding automatically generated queries.
- The result of the verification is visualized in the context of UML because the error traces on the UPPAAL level are hardly readable for the designer working on the UML level. If sequence diagrams are used we mark the first position at which a violation of the expected behavior occurs. Additional information for error diagnosis is provided.
- Control of the tool chain, execution of programs and translation of the internal model representation of Poseidon into XML-files are integrated as a plug-in in Poseidon.

Vooduu can handle a restricted subset of UML statecharts modelling object behavior as input: Concurrency is only permitted on the object level. A statechart consists of a hierarchy of composite and basic states with arbitrary transitions. Transitions can be triggered by explicit or timed events or they can be triggerless. For timed events, an interval for the emission can be specified. This construct is useful to model bounded execution time, e.g. if for an action not the exact but a worst case execution time is given. The statechart semantics implemented in *Vooduu* conforms to the UML standard. Most relevant for the translation to UPPAAL is the time model: time may only pass if all event queues of the statecharts, which have to be modelled explicitly at the UPPAAL level, are empty. If an event[6] occurs, it is put in the corresponding queue and the statechart reacts, possibly by emitting some additional events. As long as the system is instable, i.e. some event queues are still not empty, no time elapses. The details on the subset of UML statecharts, its semantics and the transformation to UPPAAL can be found in [2].

[4] For more information see http://www.gentleware.de

[5] Verification of object-oriented designs using UPPAAL

[6] which may be a time event if some timeout is reached

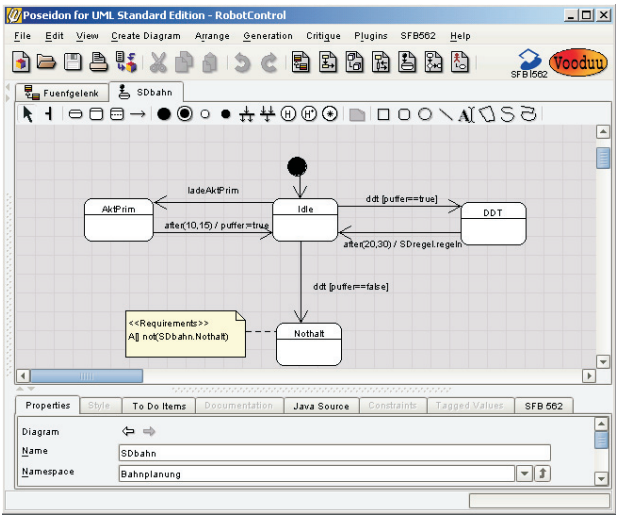

Fig. 2. *Vooduu* as plug-in for Poseidon for UML

To model requirements, we use UML sequence diagrams with some extensions necessary to specify the desired behavior precisely: each sequence diagram has a status, namely *obligatory*, which means that messages have to obey to the given interaction pattern. If the status is set to *optional* the system should allow a run, which is consistent to the sequence diagram. Moreover, a pre-chart (*if*) can be specified. If the pre-chart occurs the post-chart (*then*) becomes obligatory. Within a sequence diagram, loops can be used. A loop condition describes a set of possible iterations [1]. For messages, different modes are possible: Using synchronous messages, sending and receiving a message happen immediately. Using the asynchronous mode, the message is enqueued.

The mapping from the objects of the system model to instances in sequence diagrams can be defined in a flexible way: A set of statecharts can be mapped to one instance. The treatment of internal communication and messages not occurring in the sequence diagrams can be specified by parameters. The sending and receiving of messages can be labelled by time stamps. These time stamps can be used in timing expressions, e.g. to bound the maximal sending time of a message or the reaction time to a request. We use UML stereotypes and tagged values to model these constructs. Mainly, we verify violations of the requirements specification like:

- Incorrect message, sender, receiver
- Violation of timing conditions
- Violation of loop conditions

After modelling the dynamics of a software design using Poseidon, invoking *Vooduu* generates the necessary XML-files, starts the tool chain, performs the verification by UPPAAL and reveals the verification results.

Practical results: Our approach is limited due to the state explosion problem. In the evaluation of the tool we found that e.g. the number of states or messages in the UML diagrams does not give a good guess for the size of the state space, but crucial are the number and the interdependencies of clocks and variables and the degree of nondeterminism within the model. I.e. we could handle a medium sized rather detailed and deterministic model containing 6 processes but failed a much smaller but highly nondeterministic example.

3 Future Work

We implemented a tool that verifies whether requirements defined in terms of UML sequence diagrams are consistent with a system model in terms of UML statecharts. For future work, we intend to specialize our approach to certain application areas. E.g. for a given real-time operation system it would be useful to define a set of stereotypes which refer to its particular elements like processes, threads, channels, scheduling and communication mechanisms. This will support a more specific use of the modelling language, facilitates modelling and optimization of the translation to a model checker because non-relevant elements of the general UML can be omitted. We will integrate structural UML diagrams in the approach to support views on communication structures of software systems.

References

1. Firley, Th., Huhn, M., Diethers, K., Gehrke, Th., Goltz, U.: Timed Sequence Diagrams and Tool-Based Analysis - A Case Study. In Proc. of UML'99 - Beyond the Standard, USA, Springer, (1999) 645–660
2. Diethers, K., Goltz, U., Huhn, M.: Model Checking UML Statecharts with Time. In Proc. of UML'02, Workshop on Critical Systems Development with UML, September 2002
3. Knapp, A., Merz, S., Rauh, Ch.: Model Checking Timed UML State Machines and Collaborations. 7th Int. Symp. Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT 2002), pp. 395–414, 2002, LNCS 2469, Springer
4. David, A., Möller, O, Yi, W.: Formal Verification of UML Statecharts with Real-Time Extensions. Fundamental Approaches to Software Engineering (FASE'2002), 2002, LNCS 2306, pp. 218–232, Springer
5. Behrman, G., David, A., Larsen, K., Möller, O., Petterson, P., Yi, W.: Uppaal - Present and Future, Proc. of the 40th IEEE Conference on Decision and Control, 2001, 2281–2286