Formal Verification for Embedded Systems Design Based on MDE

Francisco Assis Moreira do Nascimento[1], Marcio Ferreira da Silva Oliveira[2], and Flávio Rech Wagner[1]

[1] Institute of Informatics - Federal University of Rio Grande do Sul - UFRGS
Av. Bento Gonçalves, 9500, Porto Alegre/RS, Brazil
fanascimento@inf.ufrgs.br, flavio@inf.ufrgs.br
[2] C-Lab, University of Paderborn
Fürstenallee, 11, Paderborn/NRW, Germany
marcio@c-lab.de

Abstract. This work presents a Model Driven Engineering (MDE) approach for the automatic generation of a network of timed automata from the functional specification of an embedded application described using UML class and sequence diagrams. By means of transformations on the UML model of the embedded system, a MOF-based representation for the network of timed automata is automatically obtained, which can be used as input to formal verification tools, as the Uppaal model checker, in order to validate desired functional and temporal properties of the embedded system specification. Since the network of timed automata is automatically generated, the methodology can be very useful for the designer, making easier the debugging and formal validation of the system specification. The paper describes the defined transformations between models, which generate the network of timed automata as well as the textual input to the Uppaal model checker, and illustrates the use of the methodology with a case study to show the effectiveness of the approach.

1 Introduction

Due mainly to severe design constraints and time-to-market urgency, embedded software applications are usually much more difficult to design than other types of applications. Furthermore, the complexity of embedded systems is increasing very fast. For example, in the automotive industry, nowdays 90 percent of the new features in recent models were related to electronics, making software be the main aspect in a car. In many recent car models, there are more than 200 functions the user interacts with, deployed over more than 60 independent embedded ECU (Electronic Control Unit).

One of the important aspects in the embedded system design is to ensure that a given system really does what it is intended to do. Nowadays, with the growing complexity of embedded systems, an exhaustive test of all possible system executions, or of at least a set of representative ones, is an impractical or even impossible approach. An alternative to testing is mathematically proving

A. Rettberg et al. (Eds.): IESS 2009, IFIP AICT 310, pp. 159–170, 2009.

correctness, by specifying precise models of the embedded system and formally verifying logical properties over these models.

1.1 Model Driven Engineering

To cope with the growing complexity of embedded systems design, several approaches based on MDE (Model Driven Engineering) have been proposed [1]. In MDE the main artifacts to be constructed and maintained are models, which are represented using a common modeling language. In the MDE context, software development consists of transforming a model into another one until a final model is obtained that is ready to be executed.

One variant of MDE is the Model Driven Architecture (MDA) ([2]), which is a framework proposed by OMG (Object Management Group) for the software development, driven by models at different abstraction levels and specified using UML ([3]). UML adopts the object oriented paradigm and includes different diagrams for the modeling of structure and behavior. In order to be used as input representation for formal verification and co-synthesis tools ([4]), a UML model must be translated into some formalism that can expose the control and data flow of the specified application in a concise and efficient way, since this information is essential to the algorithms used in the existing design automation tools.

In a UML model one can use Activity diagrams to specify such kind of information, but the internal representation defined in conformance to OMG's MOF (Meta Object Facility) ([5]) is not adequate to implement formal verification and co-synthesis algorithms, since the information is dispersed in different parts of the MOF based internal representation for UML. This makes very difficult to perform some basic operations on this representation, which are necessary for the design automation algorithms ([4]).

1.2 Formal Verification Approach Based on MDE

Differently from all other approaches oriented to MDE for embedded system design, which translate UML models to some specific internal representation format, we use only MOF concepts to define our internal design representation metamodel, and so, as a MOF-based metamodel, our internal design representation can take advantage of the concept of transformation between models to implement formal verification and co-synthesis tasks. This paper presents a formal verification methodology which adopts concepts from MDE for the automatic generation of a network of timed automata ([6]) from the functional specification of an embedded application described using UML class and sequence diagrams.

By means of transformations on the UML model of the embedded system, a MOF-based representation for the network of timed automata is automatically obtained, which can be used as input to model checking tools, as, for example, UPPAAL ([7]), in order to validate desired functional and temporal properties of the embedded system specification. Since the network of timed automata is automatically generated, the methodology can be very useful for the designers,

making easier the debugging and formal validation of the system specification. Moreover, the formal verification methodology is part of a complete MDE-based co-synthesis approach, and thus, after the formal validation of the desired properties, this same validated system specification can be directly used as input to a set of MDE-based co-synthesis tools.

1.3 Outline

The paper is organized as follows. A comparison of our methodology with other related MDE-based approaches to design validaton is given in Section 2. Section 3 introduces our MDE-based approach for formal verification, Section 4 presents the Internal Application Meta-Model, Section 5 describes the Labeled Timed Automata Meta-Model, and Section 6 presents the transformations between models that generate a Labeled Timed Automata model from our Internal Application model. Section 7 describes a case study, which illustrates our approach. Section 8 presents main conclusions and future research directions.

2 Related Work

There are many recent research efforts on embedded systems design based on MDE. The adoption of platform-independent design and executable UML has been vastly investigated. For example, xtUML ([8]) defines an executable and translatable UML subset for embedded real-time systems, allowing the simulation of UML models and the code generation for C oriented to different microcontroller platforms. However, there is no support to formal verification tools in xtUML.

The model checking based approach to formal verification of an executable UML subset, described in ([9]), can generate a S/R model for the COSPAN model checking tool. But differently from our approach, the supported UML subset does not include sequence diagrams.

The Internal Format (IF) from the OMEGA project ([10]) associates a process to each class and captures the behavior as state machines that represent the interactions between these processes. There is no concept of module to group processes and so to take into account the different forms of communications according to the partitioning of the processes. This missing information would be essential for the functional validation and co-synthesis tasks.

In the approach presented in ([11]), UML Sequence Diagrams are translated into a communication dependency graph in order to implement a specific performance analysis technique. This approach does not consider the structure and hierarchy of a UML model, as our approach does.

The co-synthesis tool POLIS [12] has an internal design representation, called CFSM (Co-Design Finite State Machine), which allows the implementation of efficient co-synthesis and formal verification strategies. However, it is not possible to use UML as input modeling language for POLIS, neither to implement the co-synthesis and formal verification tasks using MDE concepts.

3 ModSyn and Its Formal Verification Approach

Our MDE-based approach to embedded systems design automation [13][14] adopts meta-models to represent *applications*, capturing functionality by means of processes communicating by ports and channels; *platforms*, indicating available hardware/software resources; *mappings* from application into platform; and *implementations*, oriented to code generation and hardware synthesis. Figure 1 shows our MDE-based design flow, called ModSyn (Model-driven co-Synthesis for embedded systems design). In our approach, the *application* is specified independently from the *platform*, using UML as modeling language, but any other DSL (Domain Specific Language) ([1]) could also be used. A *mapping* defines how application functionality is partitioned among architectural components in order to produce an *implementation* for the specified system. Accordingly, four internal meta-models allow the independent application and platform modeling: Internal Application Meta-model (IAMM), Internal Platform Meta-Model (IPMM), Mapping Meta-model (MMM), and Implementation Meta-Model (IMM). Each meta-model provides the abstract syntax for the adopted design concepts in ModSyn. These meta-models are described in ([14]) and will not be detailed here. This paper will present the generation of a MOF-based internal design representation model conforming to IAMM from UML class and sequence diagrams and, from this internal representation, the generation of a network of timed automata. This task is performed by the Application Manager and System Designer, respectively, which are shown in Figure 1.

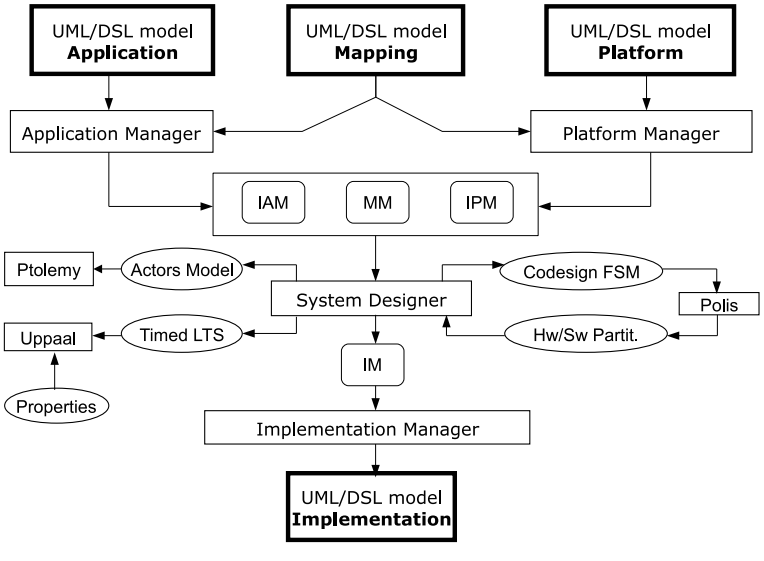

Fig. 1. ModSyn Design Flow

The ModSyn framework provides transformations between models that can generate a timed Labeled Transition System (LTS) ([6]) from IAM, which can be used in the Uppaal model checker ([7]), and also provides the generation of Co-Design Finite State Machines (CFSM) to be used by the Polis framework ([12]) in the Hardware/Software partitioning task, and an actor-based model for functional simulation using Ptolemy ([15]).

4 Internal Application Meta-model

In ModSyn, for the system structure, UML class diagrams indicate the components of the system under design, and the system behavior can be specified using UML Sequence diagrams that indicate the allowed execution scenarios. In order to represent an application in a standard way, a model that is captured using UML is translated into a common application model defined by the Internal Application Meta-model (IAMM) (illustrated by Figure 2 and Figure 3). This translation is implemented in ModSyn by means of transformations between models. As shown in Figure 2, in an application model conforming to IAMM, a system specification captures the functionality of the application in terms of a set of modules (`Module` class). Each module has module declarations (`Module Declaration class`) and a module body (`ModuleBody` class).

The control and data flow of an application model is represented by an `InteractionGraph`, presented in Figure 3. In the definition of our `Interaction Graph`, we adopt an approach similar to the proposed in ([16]), which takes MOF concepts from the UML Activity diagram meta-model. As illustrated in Figure 3, an `InteractionGraph` consists of a set of nodes (`IGNode` class) and edges (`IGEdge` class). Each node can represent different kinds of control

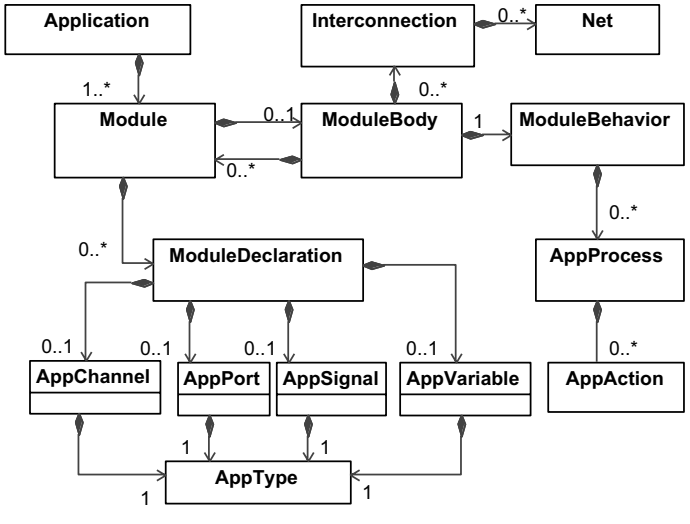

Fig. 2. Internal Application Meta-model (part 1)

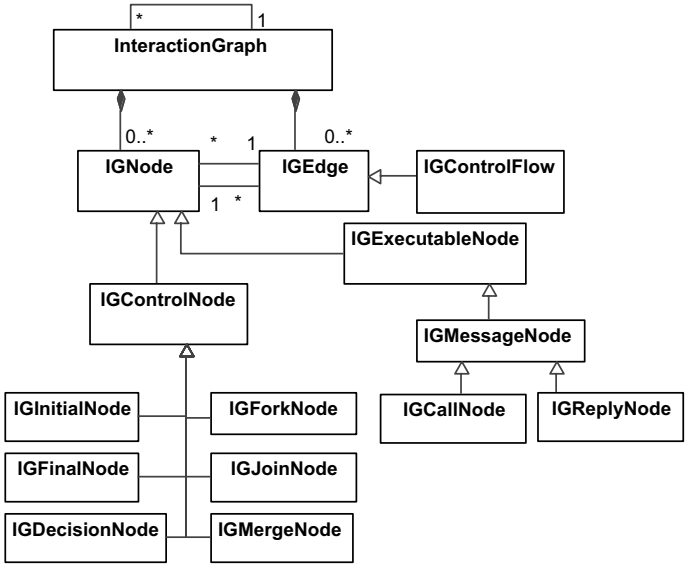

Fig. 3. Internal Application Meta-model (part 2)

flow (`IGInitialNode`, `IGFinalNode`, `IGForkNode`, `IGJoinNode`, `IGMergeNode`, `IGDecisionNode` classes) and two kinds of executable nodes (`IG CallNode` and `IGReplyNode`, sub-classes of the `IGMessageNode` class), which represent the possible actions of sending and replying messages in the UML Sequence diagram.

5 Labeled Timed Automata Meta-model

In order to represent the functional behavior of a UML model, the corresponding Internal Application Model is translated into a network of timed automata model conforming to the Labeled Timed Automata Meta-model (LTAMM) (illustrated by Figure 4), which is part of our IAM and captures all concepts introduced by the UPPAAL model checking tool ([7]). This translation is also implemented in ModSyn by means of transformations between models. As Figure 4 shows, conforming to the LTA Meta-Model, a system consists of `ltaDeclaration`s, which can be used to declare variables, functions, and channels, and `ltaProcesse`s, which are instances of `lta Template`s. Each `ltaTemplate` corresponds to a timed automaton, which can also have `ltaDecla ration`s of local variables and functions. Each timed automaton is represented by a set of `ltaLocation`s and `lta Transition`s, which have source and target locations. Each transition may have attributes: `ltaSelection`s (non-deterministically bind a given identifier to a value in a given range when transition is taken), `ltaGuard`s (transition is enabled in a state if and only if the guard evaluates to true), `ltaSyncronization`s (transitions labelled with complementary synchronization actions - send and

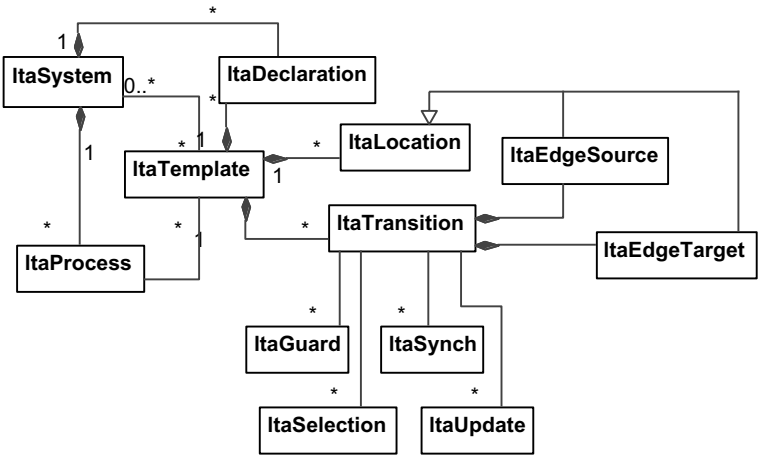

Fig. 4. LTA Meta-model

receive - over a common channel synchronise), and `ltaUpdate`s (when transition is taken, its update expression is evaluated and the side effect of this expression changes the state of the system).

6 Generating the LTA Model from UML

The transformation from a UML model into our Internal Application Model consists of a set of transformations between models, which are implemented using the Xtend language from the OpenArchitectureWare framework ([17]). The main transformation consists of traversing the UML model, where the sub-modules are identified, according to the aggregation and composition between the classes; the processes are built from the Sequence diagrams, one process for each sequence diagram; and, finally, the `InteractionGraph`s are also built from the Sequence diagrams. Each `Package` in the UML model is traversed recursively and each existing UML Class in a package is transformed into a `Module` class. Each UML Attribute of each UML class is transformed into a `Module Declaration` class. The associations between the UML classes will determine the sub-modules of each module: Each UML Class, which is part of an aggregation or composition of another UML Class, will be transformed into a `Submodule`. Derived classes are transformed into modules, and all the inherited attributes are replicated inside each such module.

In the UML/MOF, each `Lifeline` in a UML Sequence diagram is transformed into a process, which has its actions determined by the `Message` classes covered by the corresponding `Lifeline` class. For each sequence diagram, a model transformation rule in Xtend initializes and creates an `IGInitialNode` and an `IGFinalNode` for a corresponding `InteractionGraph`. After that, for each synchronous message call or signal call in the Sequence diagram a `IGCallNode` is created, and for each reply message a `IGReplyNode` is created.

IGCallNodes and IGReplyNodes are labeled with "cn-" and "rn-", respectively, followed by the name of the corresponding Message class. In the current implementation of the model transformations in Xtend, we do not yet handle asynchronous message calls, which will be one of our concerns as future work.

The Labeled Timed Automata model is also obtained from the Internal Application model by means of transformations between models implemented using the Xtend language of the openArchitectureWare framework ([17]). For each InteractionGraph we have a ltaProcess, where the ltaLocations will correspond to the IGNodes and the ltaTransitions will represent the IGEdges. The ltaSelection, ltaGuard, ltaSynchronization, and ltaUpdate attributes will capture the control flow represented in the InteractionGraph.

7 Case Study

The case study consists of a real-time embedded system dedicated to the automation and control of an intelligent wheelchair, which has several functions, such as movement control, collision avoidance, navigation, target pursuit, battery control, system supervision, task scheduling, and automatic movement. In order to illustrate the generation of an internal application model from a UML model, we focus only on the wheelchair movement control, whose simplified UML class diagram is shown in Figure 5. The UML class diagram in Figure 5 shows the MoveCtrl class, which represents the wheelchair movement controller with sensor and actuator drivers (represented by the Driver class), and a navigation mechanism (represented by the Navigator class with a Joystick component). There are two types of movement controllers (represented by MoveS and MoveC classes) that have different functions to determine each move for the wheelchair. In Figure 6(a), the UML sequence diagram defines how the possible execution scenarios for the application are composed. As shown in Figure 6(a), we have a parallel composition of the UML sequence diagrams, which are shown in Figure 6(b), and Figures 7(a), and (b). In Figure 8(a), we have a graphical representation of the CDFG corresponding to the generated InteractionGraph

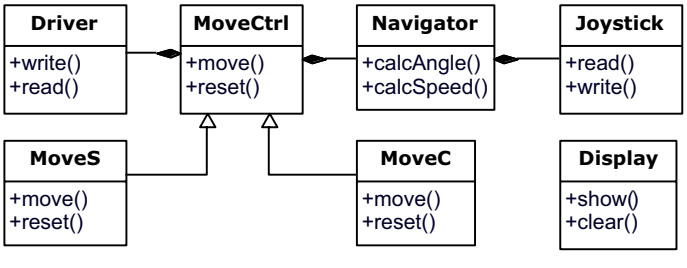

Fig. 5. Application Model: UML class diagram

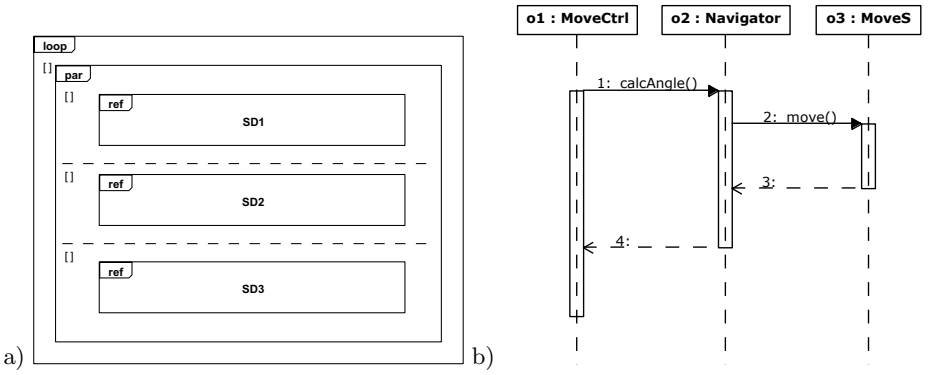

Fig. 6. UML Sequence Diagrams: a) Main b) SD1

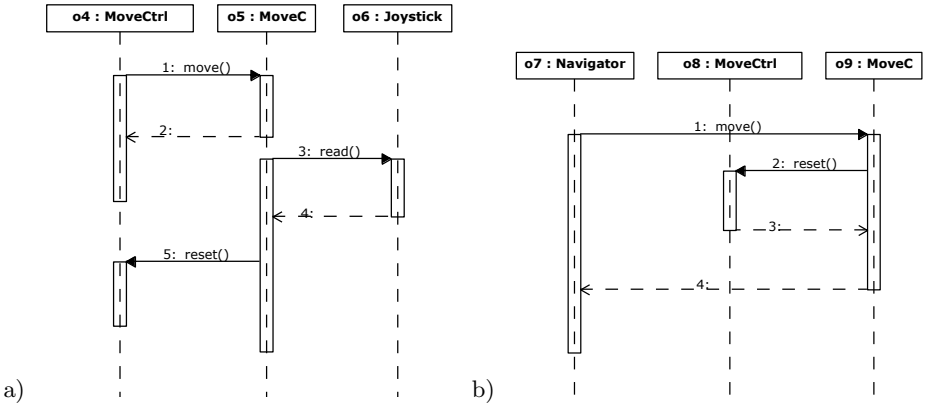

Fig. 7. UML Sequence Diagrams: a) SD2 b) SD3

for the Sequence diagram $SD1$ from Figure 6(a). The `IGCallNode`s cn-m1 and cn-m2 represent the message calls for `calc Angle()` and `move()` in the $SD1$, respectively. The `IGReplyNode`s rn-m1 and rn-m2 represent the corresponding reply messages for `calcAngle()` and `move()` in the same $SD1$, respectively. The `InteractionGraph` for the entire application is shown in Figure 8(b), where we have three `IGExecutableNode`s cn-ig1, cn-ig2, and cn-ig3, which are associated by the relation L to the corresponding `InteractionGraph`s of the sequence diagrams $SD1$, $SD2$, and $SD3$, respectivelly.

From the `InteractionGraph` in Figure 8, we obtain the network of timed automata shown in Figure 9. For the sequence diagram SD1, we have a `ltaProcess` PSD1 with six `Location`s (corresponding to the `IGnode`s labeled `Start-IG-SD1`, `cn-m1`, `cn-m2`, `rn-m1`, `rn-m2`, and `cn-Final-IG-SD1`) and five `ltaTransition`s (corresponding to the `IG Edge`s labeled `e1`, `e2`, `e3`, `e4`, and `e5`). We also have a `lpqProcess` PWheelchair for the entire application. By using the Xpand

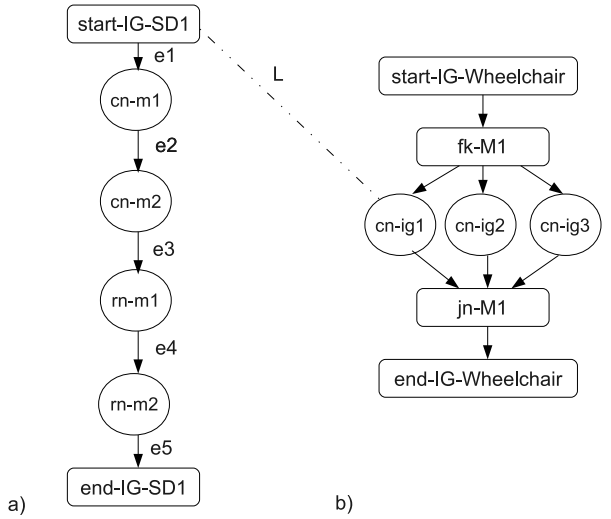

Fig. 8. InteractionGraph: a) CDFG for SD1 b) CDFG for application

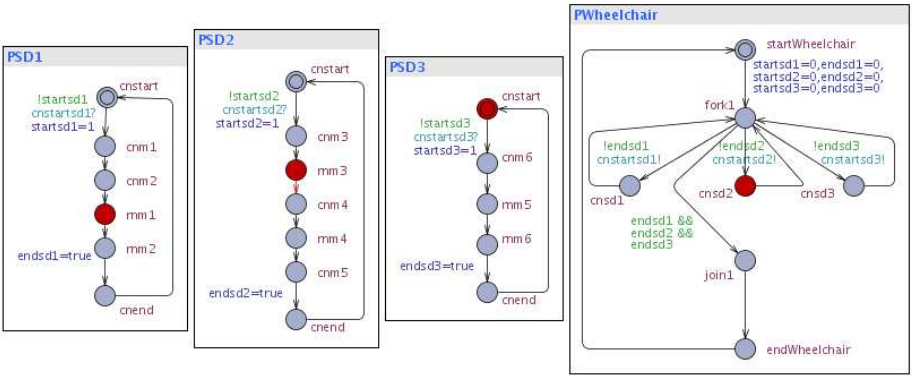

Fig. 9. Network of LTA in Uppaal for InteractionGraph

language of the openArchitectureWare framework ([17]), we implemented model-to-code transformations that generate, from the LTA model, the textual input for the Uppaal model checker. At this point, the designer can specify logical properties using CTL formulae and use Uppaal to verify them, as illustrated by Figure 10.

As shown in Figure 10, we have specified a property to check if the application model is deadlock-free (using the Uppaal macro `A[] not deadlock`) and if eventually the processes corresponding to the sequence diagrams will be executed all in parallel (using the CTL formula `E<> startsd1 and startsd2 and startsd3`).

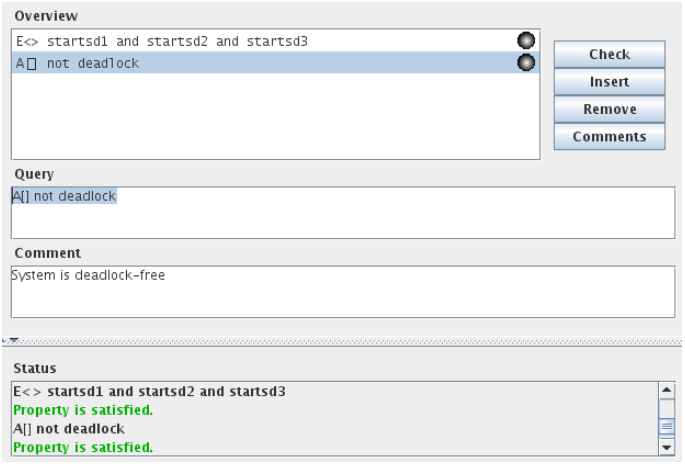

Fig. 10. Proving properties in Uppaal

8 Conclusions and Future Work

In this paper, the MDE fundamental notion of transformation between models is used to generate an internal representation model to be used by formal verification and co-synthesis tools, from a UML model of an application consisting of Class and Sequence diagrams. The obtained model captures structural aspects of an application model by using a hierarchy of modules and processes, as well as behavioral aspects by means of a control/data flow graph model.

We are currently implementing co-synthesis algorithms based on this internal representation model conforming to the Internal Application Meta-Model (IAMM) and using the concept of transformations between models from MDE to perform the co-synthesis tasks, as, for example, the task of hardware/software partitioning applied on the processes represented by the InteractionGraphs of an application. Some types of message calls and combined fragments in the sequence diagrams of UML 2.0 are not yet handled by our current implementation and will be one of our topics for future work.

Acknowledgements

This work described herein was partly supported by the German Ministry for Education and Research (BMBF) through the ITEA2 project TIMMO (01IS07002).

References

1. Schmidt, D.C.: Model driven engineering. IEEE Computer 23(2), 25–31 (2004)
2. OMG: MDA guide version 1.0.1 (2003), `http://www.omg.org`
3. OMG: UML - unified modeling language (2009), `http://www.omg.org`

4. Edwards, S., et al.: Design of embedded systems: Formal models, validation, and synthesis. Proc. of IEEE 15(3), 366–390 (1997)
5. OMG: Meta object facility 2.0 core specification (2009), http://www.omg.org
6. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Sciences 126(2), 183–235 (1994)
7. Larsen, K.G., et al.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfers 1(1-2), 134–152 (1997)
8. Mellor, S., Balcer, M.: Executable UML: A foundation for Model Driven Architecture. Addison-Wesley, Boston (2002)
9. Xie, F., Levin, V., Browne, J.C.: Model checking for an executable subset of UML. In: Int. Conf. on Automated Software Engineering - ASE, pp. 333–336 (2001)
10. Hooman, J., et al.: Supporting UML-based development of embedded systems by formal techniques. International Journal of Software and Systems Modeling (SoSym) 7, 131–155 (2008)
11. Viehl, A., et al.: Performance analysis of sequence diagrams for soc design. In: UMLSoC 2005 - Workshop on UML for SoC (June 2005)
12. Balarin, F., et al.: Hardware-Software Co-Design of Embedded Systems: The Polis Approach. Kluwer Academic Publishers, Boston (1997)
13. do Nascimento, F.A.M., da Oliveira, M.F.S., Wehrmeister, M.A., Pereira, C.E., Wagner, F.R.: MDA-based approach for embedded software generation from a UML/MOF repository. In: Brazilian Symposium on Integrated Circuits (SBCCI), pp. 143–148 (2006)
14. do Nascimento, F.A.M., Oliveira, M.F.S., Wagner, F.R.: ModES: Embedded systems design methodology and tools based on MDE. In: International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES), pp. 67–76 (2007)
15. Lee, E.A.: Overview of the ptolemy project - technical memorandum UCB/ERL m03/25. Technical report, University of California, Berkeley, CA, USA (July 2003)
16. Garousi, V., et al.: Control flow analysis of UML 2.0 sequence diagrams. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 160–174. Springer, Heidelberg (2005)
17. openArchitectureWare: openarchitectureware portal (2009), http://www.openarchitectureware.org