# Correct Development of Embedded Systems*

Susanne Graf[1] and Jozef Hooman[2]

[1] VERIMAG, Grenoble, France
Susanne.Graf@imag.fr
[2] University of Nijmegen and Embedded Systems Institute, Eindhoven, The Netherlands
jozef.hooman@embeddedsystems.nl

**Abstract.** This paper provides an overview on the approach of the IST OMEGA project for the development of correct software for embedded systems based on the use of UML as modelling language. The main contributions of the project are the definition of a useful subset of UML and some extensions, a formal dynamic semantics integrating all notations and a tool set for the validation of models based on this semantics.

## 1 Introduction

Building embedded real-time software systems of guaranteed quality, in a cost-effective manner, is an important technological challenge. In many industrial sectors, such as automotive and telecommunications, a proper development process supported by validation and formal verification tools is requested. Furthermore, the relations between suppliers and manufacturers are changing; the suppliers provide components which are integrated by manufacturers to produce goods and services of guaranteed quality. This requires new software engineering environments supporting architecture-based engineering practice for building complex systems by composing available components with known properties and evaluating the impact of local design choices on their global behaviour. There is now a general agreement that a means to achieve this is a *Model based approach* which has the following characteristics:

- This approach is based on the existence of a global model of a software system, consisting of possibly heterogeneous components. This model should address different aspects of the system - functional, architectural, non-functional, etc. Changes may be made for some aspects during the development from very high level requirements down to code; nevertheless the consistency of the global model must be maintained throughout the development.
- At any level of abstraction, models should be executable. In the context of embedded systems a model of the environment is also needed in order to allow "testing at model level"; this is interesting as in a model there exists a better controllability of the system and the explicit modelling of non-functional aspects (especially time and memory) allows to avoid the "probe effect" due to the presence of a "tester".

Such a model-based development approach is only useful if it is accompanied by tool support for the validation of design choices. This is particularly true in the context

* This work has been supported by the IST-2002-33522 OMEGA project – http://www-omega.imag.fr

F. Oquendo et al. (Eds.): EWSA 2004, LNCS 3047, pp. 241–249, 2004.

of real-time systems, where non-functional properties, such as reactivity of the system, are as important as its functionality. In order to detect design errors early, it is necessary to take non-functional aspects into account, in particular time-related ones, in high-level abstract models. Early decisions on the ordering of independent activities may later need important redesign when it turns out that time constraints cannot be met. Resolving non-determinism when timing constraints are already taken into account allows avoiding this problem.

Formal validation of different aspects (functional, non-functional) is usually done on specialised analysis models. In order to guarantee consistency, these models should be obtained by tool-supported extraction of the relevant information from the global model, and results of the analysis must be fed back into the global model. To avoid divergence between model and code, which would make the model useless, it is important to have automatic generation of code, depending on the target platform. Moreover, to avoid that "bugs" are eliminated at code level only, also support for round-trip-engineering is needed, where changes in the code are reflected in the model automatically.

In order to be able to implement an environment for such a model-based approach, one needs (1) notations for representing all aspects of heterogeneous systems and their environment, by separating as much as possible different aspects, (2) a formal semantics integrating all notations into a global model of both static constraints and dynamic behaviour, and (3) tools and methods for simulation and formal validation for both functional and non-functional properties of the system. The Unified Modelling Language (UML), which has been developed with the goal to enable model-based development, has become a de facto standard in the domain of software development and is imposing itself also in the domain of real-time and embedded systems.

In this paper we report on work done in the EU-IST project OMEGA. The goal of this project is to provide a framework the development of embedded software systems based on UML and formal verification. The project has 6 academic partners, Verimag, CWI, OFFIS, Weizmann Institute and the universities of Kiel and Nijmegen and 4 industrial users, EADS, France Telecom R&D, Israeli Aircraft Industries and National Aerospace Lab of the Netherlands. The academic partners provide the formal semantics and validation tools based an requirements and feedback from industrial users. The approach is being validated and continuously improved on 4 industrial case studies.

Section 2 gives a brief overview on UML and the state-of-the-art of UML tools as well as formal validation techniques. Section 3 presents the approach chosen by the Omega project and section 4 contains some feedback and lessons learned from the progress achieved within the first two years of the project.

## 2 Towards UML Based Development: State-of-the-Art and Problems to Be Solved

This section has 3 parts, giving a critical overview on the three main ingredients of the problem the Omega project wants to solve, UML and its semantics, formal validation methods and tools and UML-based CASE tools.

*UML, its aims and deficiencies:* UML aims at providing an integrated modelling framework encompassing architecture descriptions, as supported by the various Architecture Description Languages (ADL), and behaviour descriptions, as supported by various behavioural specification languages such as languages based on the concept of communicating state machines. Nevertheless, in UML some aspects of model based design are not sufficiently addressed:

- Semantic issues are hardly addressed. The UML meta-model solves a part of the static semantics, but most of the dynamic semantics is left to the CASE tools. Not fixing the dynamic semantics is intentional to provide a unified framework for various domains with different needs. Nevertheless, this means that validation tools are dependent on the semantic choices of each CASE tool. Notions for distinguishing successive refinements of models as well as an appropriate notion of refinement are lacking. In particular, there is no means to distinguish within a single refinement step between the "model" of the system under development and "properties" which can be used as a consistency check of this model and should be implied by it. All properties expressed - in an operational or declarative manner - have a priori the same status.
- Some of the UML notations are not expressive enough or have no appropriate semantics.
  - UML Sequence Diagrams are not meant for fully characterising the set of possible scenarios of a system. We want however use them for this purpose as an alternative to temporal logic which is less well accepted by users than scenario based specifications.
  - The notions for expressing architecture and components were very weak in the initial versions of UML. UML 2.0 has improved the situation, at least at the syntactic level.
  - UML has not been developed in the context of safety or performance critical systems, and initially, time and performance related features have not been considered otherwise than in the form of informal annotations. The Profile for scheduling performance and real-time (SPT) [OMG02] has brought some additional notation, but no concrete syntactic framework, and no semantics.

In the Omega project, we address the above mentioned issues by defining a UML Kernel model with extensions and a formal semantics. We provide also a notation for an explicit distinction between diagrams being part of the model definition and those representing requirements which must be implied by the model and represent the properties to be verified.

*Formal verification:* Little tool support exists so far for the formal verification of all aspects of this kind of systems. There exists many model checking tools [QS82,CES83] for verifying properties on finite state models. Similarly, there are tools for validating properties of timed systems [Yov97,JLS00] based on timed automata [AD94] and frameworks for scheduling analysis and performance evaluation. These tools suppose that models of a particular form are provided. Some tools claim to handle UML models (e.g. [LP99,CHS00]), but a closer look shows that they validate state charts [Har87] or UML activity diagrams. There are also many tools for the verification of properties of some form of scenarios. Nevertheless, it is impossible to use the different existing tools

together for a more complete analysis on a common model which is, amongst others, due to incompatibilities of semantics between tools.

Besides obvious problems of syntax compatibility, there are two main fundamental problems to be addressed when building validation tools that are smoothly integratable into a UML based software development process:

- The problem of adapting the verification techniques and making them scalable. In particular, the use of UML poses several challenges:
  - Verification of systems defined using object-oriented features, such as dynamic object creation and destruction, inheritance and parameterisation.
  - Verification of complex requirements on systems with a complex structure, and including non-functional aspects, such as time related features.
- The problem of model extraction: in UML, different diagrams are used to represent different aspects of a system which all together represent a unique model.
  - To obtain a faithful semantic model (e.g., for interactive or guided simulation) all these must be combined in a consistent manner into a unique semantic model.
  - Different aspects are verified in different steps, possibly using different tools. It is important to extract for the validation of each aspect all the necessary information, but not more than that.

Within OMEGA, UML models are translated into the formats of several existing validation tools. In particular, we consider a tool for handling scenario based requirements [DH99,HKMP03], an untimed model-checking tool [BDW00] and a model-checking tool for timed and untimed specifications [BFG+99,GM02]. We also provide a mapping, dealing with general OCL constraints, to PVS [SOR93], an interactive theorem prover allowing general reasoning about models, and thus potentially allowing to overcome some of the problems occurring with object-orientation.

The problem of scalability is addressed in several ways. General compositionality results are applied, and in particular, two aspect-depending abstract models are extracted: an untimed model dealing only with the functional aspects, and a timed model taking into account only control, interaction and timing and scheduling related aspects. In the future, both analysis methods should profit from each other. Presently each of these models is simplified using abstraction and static analysis algorithms implemented in the individual tools.

Finally, model transformation approaches are considered, in the form of scheduler synthesis and the synthesis of a state chart model from a complete set of scenario specifications. Nevertheless, the underlying synthesis problems have a prohibitive complexity, except when applied in restricted contexts.

*UML CASE tools:* There is a large number of generic CASE tools for UML, which allow mainly to edit diagrams and to generate templates for code production. They only deal with the static structure of a software. For the object-oriented development of real-time systems, there exists a number of specialised CASE tools, such as Rhapsody of I-Logix [Ilo], Real-time Studio of ARTiSAN [Art01b], TAU Generation-2 of Telelogic [Tel02], Rose-RT of IBM/Rational [SR98]. Contrary to general purpose UML tools, they all implement some *dynamic semantics* of UML and allow the user to interactively simulate models, as well as to generate executable code. Nevertheless, most of them pose one or several of the following problems:

- They are visual programming tools rather than modelling tools; non-determinism which is not modelled explicitly - in the environment - is forbidden or eliminated in some way by the tool.
- Some timing features, such as timers, are in general available, but no tool implements a framework as sketched in the SPT profile.
- Some notations, in particular the Object constraint Language, OCL [WK98] which is very useful for constraining models, never really made their way into any CASE tool.
- Apart from some simple static checks, the only available validation method is model-based testing, i.e. interactive or guided simulation of the directly executable part of the model. Tools for formal validation of models are lacking.

In the Omega project, our intention is not to improve CASE tools, but to develop tools that can be used together with any CASE tool exporting models in the standard XMI format. We achieve inter-operability by using mainly the extension mechanisms provided by UML itself. Our tools, however, are not made to be compatible with the dynamic semantics of any particular tool, but propose a rather non-deterministic semantics in which a part of the non-determinism is to be eliminated by timing, scheduling and user defined priority constraints, and not only by predefined rules.

Many important topics are not addressed in Omega, such as code generation, test case generation, general automatic model transformation and refinement, how to get appropriate estimations on execution times and other durations used in the high level model, as well as dealing with other non-functional characteristics, such as memory or power constraints. Moreover, we do not address meta-tools for model and proof management, which is an issue that should typically be handled within a commercial CASE tool.

## 3 The Omega Approach and Initial Developments

Within Omega, we intend to build a basis for an environment for rigorous model based development of real-time and embedded systems addressing basic issues, such as an appropriate set of notations with common semantic foundations, tool supported verification methods for real-life systems, as well as real-time related aspects. This section gives an overview on how we have addressed these problems.

### 3.1 UML Notations and Semantics

Concerning the problems of **expressiveness**, we mention here the most crucial issues in the context of real-time systems:

- In a given UML specification, we distinguish between the model and requirements to be verified on this model. Class and architecture diagrams, as well as state-machines are used for the definition of the model. To strengthen the model or to define requirements, (1) Scenarios in a formalism called Live Sequence Charts (LSC), which are more expressive than the standard UML sequence diagram and have a defined semantics[], (2) a subset of OCL extended with history depending constraints [KdB02,KdBJvdZ03] and (3) particular state machines, stereotyped as “observers” [OGO04] can be used.

- In typical embedded systems several execution and communication modes are used[1]. We consider a system as a hierarchically structured set of components. *Activity groups*, which may be part of and contain components, define a mono threaded behaviour, interacting with the environment and revealing their state only at well defined points in between so called run-to-completion steps; no implicit choice concerning the execution order of concurrently active activity groups is made. Communication is either by synchronous method calls or asynchronous signals. This semantics is defined in the form of a unique symbolic transition system in [DJPV03]. A more abstract version by means of an interpretation in PVS is defined in [vdZH03].
- A concrete timing framework, consistent with the SPT profile, has been developed, allowing to define a timed extension of any model. This framework is based on the notion of *timed event*, which can also be used to define a user-defined notion of *observability*, where durations between occurrences of events and constraints on them can be expressed by particular OCL expressions. The semantics of time extensions are orthogonal to the operational semantics.
- We consider architecture diagrams as static constraints and use components in order to define an appropriate notion of encapsulation for compositional reasoning and property preserving refinement. The interface between a component and its environment is defined by ports, where the communication between a component and its environment is only via these ports.

Thus, in order to achieve *semantic integration*, we consider a quite small, but powerful, subset of notations. Presently, activity diagrams are not considered but later they could be integrated easily as an alternative way to define "tasks", as needed in the context of timing and scheduling analysis.

### 3.2 Tool Support for Verification

The aim of the Omega project is to not impose a particular development methodology, but to provide methodological support for the use of the modelling language and tools in combination, as validation is only feasible for models respecting some structure. To be able to provide useful verification results in the context of software development in the Omega framework, we propose to extend and adapt a number of existing verification tools integrating state-of-the-art technology where each one solves a particular verification problem. The work has two parts, adapting tools to the UML technology and extending the verification technology, as described below.

*Adapting tools to the* UML *technology:* We have chosen to build upon the existence of the UML exchange format XMI which includes a standard extension mechanism useful to adapt UML to a particular framework. All the validation tools will rely on the same XMI format, and the common semantic framework allows to ensure consistent interpretation of a model amongst the different tools. The problems we had to face are the weakness of the XMI standard (in particular, there exists no structured representation of OCL or the action language, and the representation of sequence diagrams is too

[1] E.g. so-called GALS - globally asynchronous, locally synchronous systems are often considered. Our model is close to this view.

poor to represent the more powerful LSC) and the fact that XMI is still not sufficiently adopted, or used in different ways, by the different CASE tool builders.

*Extending the verification technology:* The main techniques for making formal verification scalable consist in exploiting the principles of compositionality (composing properties of subsystems to derive system properties) and abstraction (extracting just the necessary information from the model of a system or a component for the verification of a given aspect or property). It is well-known that time related aspects are by their nature not very compositional and the object-oriented setting makes the static analysis used for model extraction hard to apply. The methodology used - and its support by the notational framework - plays an important role for obtaining models in which the relevant component can be composed to system properties. This kind of methodological support is out of the scope of the project[2].

Our aim is to provide model-checking tools for establishing properties of components, where the notion of component interface plays an important role for establishing the notion of externally observable behaviour. We use composition theories and support of interactive theorem provers and composability results for timing properties to deduce system properties.

*Overview on the Omega tool set:* There are four main validation tools in the tool set:

- The play-in/play-out tool [HKMP03] allows user friendly editing of LSC (called play-in), interactive simulation and verification of consistency of LSC (called play-out) and an extension for state machine synthesis.
- A model-checking tool [BDW00] allows the verification of functional properties on an untimed model (time is counted in terms of run-to-completion steps). Properties can either be described by LSC or by a set of temporal logic patterns, integrated in the user interface of the verification tool.
- The IF verification platform [BFG+99,OGO04] allows timed and untimed verification. Nevertheless, it is more appropriate for the verification of coordination and timing properties and scheduling analysis. It takes into account the Omega real-time extensions and represents the UML model by a set of extended timed automata by abstracting a variable amount of attributes. The consistency of a model can be validated by interactive or exhaustive simulation of such a more or less abstract model. Properties, expressed by Omega time constraints or observers can be verified. In some cases, automata representing the externally observable behaviour can be generated. Also schedulability analysis is formulated as a consistency problem and validated in the same way on a particular abstraction.
- A set of tools [KdBJvdZ03,KdB03,vdZH03] built upon the interactive theorem prover PVS. They rely all on the same translation from UML, including OCL constraints, into a PVS expression. The aim is to verify systems which may have configurations of unbounded size and unbounded message queues and data. The tools aims at the verification of type-checking conditions, consistency checks, and proving properties of systems expressed either as temporal logic formulas or as OCL constraints.

[2] There is presently much effort devoted to this subject, e.g., in the Artist Network of excellence, see http://www.artist-embedded.org/

An important aspect of the Omega tool set is that all tools are based on a common reference semantics of UML, the only way to ensure that all tools analyse the same model. An effort will be made to provide feedback to the user in a user-friendly manner (e.g. error traces are provided in the form of scenarios), but beyond the already mentioned limited synthesis approaches no feedback in the form of a corrected or refined UML models in XMI format can be provided within the duration of the project. The tools will only provide relevant information, helping the user to manually update or refine the model.

## 4 Some Lessons Learned

A very preliminary analysis of the feedback from the work with the case studies allowed us to identify some critical points from which we mention only the most important ones:

- Object orientation makes static analysis and constraint propagation, e.g., methods which are important to make the model-checking approach feasible, are very hard to apply due to potential aliasing. Moreover, in the context of embedded systems, the only object oriented feature frequently used is static inheritance which can be compiled away for validation.
- Presently, different tools handle somewhat different subsets of the Kernel UML, for example, only one tool handles OCL, and each tool has its own internal format. A common semantic level format, which keeps the structure and concepts useful for validation and maps all others into more primitive ones, would be interesting for exchanging some effort (e.g. translation of OCL constraints, operational meaning of LSC, ...)
- Users are very satisfied with LSC for the expression of requirements, as long as they are not required to provide complete specifications. This means that we might have to revise our approach to synthesis of state charts from LSC.
- Interactive verification using PVS based on the general semantic model is rather complex and requires many user interactions. This can be improved by restricting the semantics to the features that occur in the model under investigation and by using the powerful strategies of TLPVS, which mechanises proof rules for temporal logic. For the verification of large models, the use of compositionality is essential.
- concerning scalability, there is still quite some effort to be made.

## References

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Art01b] *ARTiSAN*, 2001.

[BDW00] T. Bienmüller, W. Damm, and H. Wittke. The STATEMATE Verification Environment – Making it real. In *International Conference on Computer Aided Verification, CAV*, number 1855 in LNCS, 2000.

[BFG+99] M. Bozga, J.C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, and L. Mounier. IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems. In *Formal Methods'99, Toulouse, France*, number 1708 of LNCS, 1999.

[CES83] E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic verification of finite state concurrent systems using temporal logic specification: a practical approach. In *ACM Symposium on Principles of Programming Languages (POPL)*, 1983.

[CHS00] K. Compton, J. Huggins, and W. Shen. A semantic model for the state machine in the unified modelling language. In *Dynamic Behaviour in UML Models: Semantic Questions, UML2000 Workshop*, 2000.

[DH99] W. Damm and D. Harel. LSCs: Breathing life into Message Sequence Charts. In *FMOODS'99, Int. Conf. on Formal Methods for Open Object-Based Distributed Systems*. Kluwer, 1999.

[DJPV03] W. Damm, B. Josko, A. Pnueli, and A. Votintseva. A formal semantics for a UML kernel language. In F. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *1st Symp. on Formal Methods for Components and Objects, revised lectures*, volume 2852 of *LNCS Tutorials*, 2003.

[GM02] M. Bozga S. Graf and L. Mounier. IF-2.0: A validation environment for component-based real-time systems. In *Conference on Computer Aided Verification, CAV*, LNCS 2404, 2002.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Programming 8, 231-274*, 1987.

[HKMP03] D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart play-out. In *Companion of the ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 2003.

[Ilo] Ilogix. Rhapsody development environment.

[JLS00] H. Jensen, K.G. Larsen, and A. Skou. Scaling up UPPAAL: Automatic verification of real-time systems using compositionality and abstraction. In *FTRTFT*, 2000.

[KdB02] M. Kyas and F. de Boer. D1.2.1: Assertion Languages for Object Structures in UML. Deliverable of the IST-2001-33522 OMEGA project, 2002.

[KdB03] M. Kyas and F. de Boer. On message specification in OCL. In *Compositional Verification in UML*, Workshop associated with UML 2003.

[KdBJvdZ03] M. Kyas, F.. de Boer, J. Jacob, and M. v. d. Zwaag. Translating UML and OCL to PVS. In *Submitted*, 2003.

[LP99] J. Lilus and I. Porres Paltor. vUML: a tool for verifying UML models. Technical Report No 272, Turku Centre for Computer Science, 1999.

[OGO04] I. Ober, S. Graf, and I. Ober. Model checking of UML models via a mapping to communicating extended timed automata. In *SPIN Wshop on Model Checking of Software*, LNCS 2989, 2004.

[OMG02] OMG. Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0. OMG document ad/2002-03-04, March 2002.

[QS82] J-P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Int. Symp. on Programming, LNCS 137*, 1982.

[SOR93] N. Shankar, S. Owre, and J.M. Rushby. The PVS proof checker: A reference manual (draft). Tech. report, Comp. Sci.,Laboratory, SRI International, 1993.

[SR98] B. Selic and J. Rumbaugh. Using UML for Modeling Complex Real-Time Systems. Whitepaper, Rational Software Corp., March 1998.

[Tel02] Telelogic. *TAU Generation 2 Reference Manual*, 2002.

[vdZH03] M. v. d. Zwaag and J. Hooman. A semantics of communicating reactive objects with timing. In *Proc. of Workshop on Specification and Validation of UML models for Real-Time Embedded Systems (SVERTS)*, 2003.

[WK98] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1998.

[Yov97] S. Yovine. KRONOS: A verification tool for real-time systems. *Journal of Software Tools for Technology Transfer*, 1(1-2), 1997.