Correct Development of Real-Time Systems

ΩMEGA

Deliverable D7: FINAL PROJECT REPORT

July 2005

Editor: Susanne Graf, Verimag

Contributors: Susanne Graf, Frank de Boer, Pierre Combes, Jozef Hooman, Hillel Kugler, Marcel Kyas, David Lesens, Iulian Ober, Angelika Votintseva, Yuri Yushtein, Meir Zenou

Document Version: 7

Status: Final version

Confidentiality: Public

Date of version: 06/07/05

Table of contents

1 Executive Summary

The Omega project has been a three year research and development effort on the part of a consortium of six academic bodies and four industrial users, partially funded by the European commission under the Fifth Framework Agreement. Moreover, the project got support and feedback expressing interest from three of the main UML tool providers, as well as concrete collaboration proposals of one of them.

The aim of the project is to increase the efficiency and competitiveness of the European Software industry by providing a framework for tighter integration of software validation into the software development process with the aim of contributing to the reduction of the cost and time of the software development process in the context of real-time and embedded systems.

All results of the project are available in some form to the general public:

- **Scientific results are provided in the form of largely distributed publications**
- **The UML profile for real-time and embedded systems developed in the project, which is available in the form of documents and libraries**
- **The tools developed in the project are maintained by their owners and accessible to external users,**
- **An overview on and pointers to project results are available on the Omega web page:** http://www-omega.imag.fr/

This is the final report of the project which focuses on the objectives, achievements and lessons learned. It provides also some discussion of future research directions. In the course of the past three years, the consortium has:

- Developed a UML profile adequate for the development of real-time embedded systems and usable with some major UML CASE tools[1], including means for the expression of functional and non functional specifications and requirements of such systems. The consortium has also provided a formal semantics allowing a consistent use of the different notations provided by the profile.

  This profile turned out to be useful for modelling real-time systems and their timing properties. It will continue to be used as is with the existing tools. Parts of it have already influenced the new version of the standard (in particular UML 2.0 sequence diagrams) and it will flow into the new real-time standard (in particular MARTE and the semantic profile). For

- Developed a set of tools and methods allowing the validation of different aspects of systems, in particular coordination and timing related issues, of models adhering to the constraints of the Omega profile. The tools address both checking of internal consistency of specifications and requirements and consistency of specification models with requirements.

  These tools continue to be maintained and further developed. In particular industry funded follow-up projects for tighter integration with existing UML

[1] In the project, we worked with Rhapsody and Rational Rose, comparisons with profiles of other tools remain still to be done

case tools, such as Rhapsody, and integration into Eclipse are being set up or already started.

- Completed four different experiments of the usage of the Omega profile and tool set on four different case studies addressing different application domains in collaboration between the tool providers and the users.

  These case studies were extremely useful for the success of the project. They successfully demonstrated the usefulness of the profiles and tools developed in Omega. Presently, they are used both by the academic partners who built the tools and the industrial users who provided the case studies to promote the project results in the community of embedded systems in general and internally in their respective companies. The exploitations planned for the profiles and the tools are to a large part due to the demonstrations provided by the case studies. Detailed accounts and analyses will be available shortly in the form of publications.

- Provided methodological support for a potential user of the OMEGA profile and tools providing support for optimal usage of the profile and the tools, independently of the general development methodology adopted by the user. The case studies and the obtained validation results are available as a part of the methodological support.

- Participated in many conferences and workshops for disseminating the project results and publishing 80 papers in proceedings of international conferences and journals. In addition, the project plans a special section in an international journal on results on the Omega project.

- Organised and initiated more than 10 workshops and conferences. In particular, the consortium created successful forums aiming at the exchange of results and experience reports from academia and industry related to the topics of OMEGA: the symposium on Formal Methods for Components and Objects, FMCO, and a workshop on the Specification and Validation of models for Real Time and Embedded Systems, SVERTS.

All the above mentioned results are discussed in more details in the following sections of this report and more detailed exploitation plans explained in Section 9.

2 Project Objectives

Building embedded real-time systems of guaranteed quality, in a cost-effective manner, is an important technological challenge. In many industrial sectors, a development process supported by validation and verification tools is requested.
Modelling plays a central role in software and systems engineering. The use of models can profitably replace experimentation on actual systems with incomparable advantages such as,

- enhanced modifiability of the model and of its parameters,
- ease of construction by integration of models of heterogeneous components,
- generality by using genericity mechanisms and behavioural non determinism,
- enhanced observability and controllability, in particular, avoidance of probe effect and of disturbances due to experimentation,
- possibility of abstraction and application of formal methods.

Building models which faithfully represent complex systems is a non trivial problem. Often, modelling techniques are applied at early phases of system development at a high level of abstraction. Nevertheless, there is a need for a unified view of the various activities in the life-cycle and of their interdependencies.

The so called model driven engineering methods rely on the existence of one or several models providing complimentary views of the system which are used for validation and code generation. The Unified Modelling Language UML has been defined for supporting such an approach. It includes notations for the description of structural and different behaviour views of an application, as well as platform dependent information. UML is a set of weakly integrated concepts, and when the project started, it had really weak coverage of the needs of real-time embedded systems which have been improved since with UML 2.0 which at the project start existed in the form of an early draft.

The aim of the Omega project was not to cover the whole process, but aimed at making possible the integration of the use of formal verification techniques in the development progress.

- Identify a reasonable and effective subset of UML which can be used for the development of real-time embedded systems, including both specification level and requirement level notations, and both implementation independent and implementation dependent real-time aspects of such systems.
- Provide a formal semantics of this profile in order to make possible consistent use of different notations, as well as the consistent mapping into the input languages of the formal verification tools.
- Provide a set of tools for the verification of real-time embedded systems described in this profile, where different tools may validate different aspects. For mastering complexity issues, develop methods and tools for compositional verification and experiment methods for synthesis of specifications from requirements.
- Provide sufficient methodology support so that the users can use the developed profile and the tools.
- Apply industrial case studies for evaluating the new real-time UML profile and the proposed verification methods and tools.

The work in OMEGA was based on the following hypotheses:

- Verification is only an aspect of the whole process, even if it is the only one considered in this project. This means that the chosen profile must be rich enough to fit the development process and if the validation process imposes restrictions, they are not allowed to hinder the development process.
- If we develop a profile rich enough for fitting the needs of the users coming with a well defined semantics and tool support for validation, it will be taken up by CASE tool providers
- Developing state-of-the-art verification tools is expensive, and as they concern only a subset of the users, CASE tool builders hesitate to invest into such technologies. This, amongst others, motivated our choice to build on the standard model interchange format XMI, adopted presently by some of the main UML CASE tools.

3 OMEGA general methodology and architecture

3.1 Problem statement

The project builds on the generally accepted assumption that the so called model based engineering will help to build better systems and to make their maintenance easier.. It presumes the existence of a model, or of a set of models, which is used for specification purposes, for validation purposes – including all kinds of validation techniques, as well as possibly also test case generation - and for generating code – where code generation is done in an automatic fashion or it provides skeletons, including the relevant information of the model in such a way that the code can be fed back into the model.

We have considered that UML is a good candidate for such an approach; it provides a number of notations allowing to represent a system model at a more or less detailed level, there are multiple CASE tools that can be used to define models, and there exist some examples demonstrating the intended process – for example the process proposed by Rhapsody [Ilo] or by the Accord model [Acc] and others. It's on those we wanted to improve by extending them with more powerful features for expressing requirements and time and by considering more abstract, and nevertheless formal, models. Having defined such a profile, the project considered two interesting problems in a framework of model based development, which is often neglected by commercial tools:

- Tool support for validation, whenever possible by reusing existing state-of-the-art tools.
- Explorative research on two issues related to model transformation:
    - Synthesis of a design level model (a state machine for each class) from requirements expressing a service point of view.
    - A generalisation of the scheduling problem, the transformation of a design level concurrency model into an implementation level concurrency model.

The validation tools existing before the project started, did not handle many of the concepts to be dealt with in UML, such as inheritance, dynamic object creation, time etc, and no existing tool handled all of them in combination.

Semantic issues are important, even if different tools handle different aspects of the system, there will always be some overlap between the concepts handled by different tools, and in order to verify dynamic properties, tools must somehow agree on a common semantic ground to provide valuable feedback to the users.

3.2 General tool set architecture and integration

A general overview on the tool set and on the possible flows between the different tools

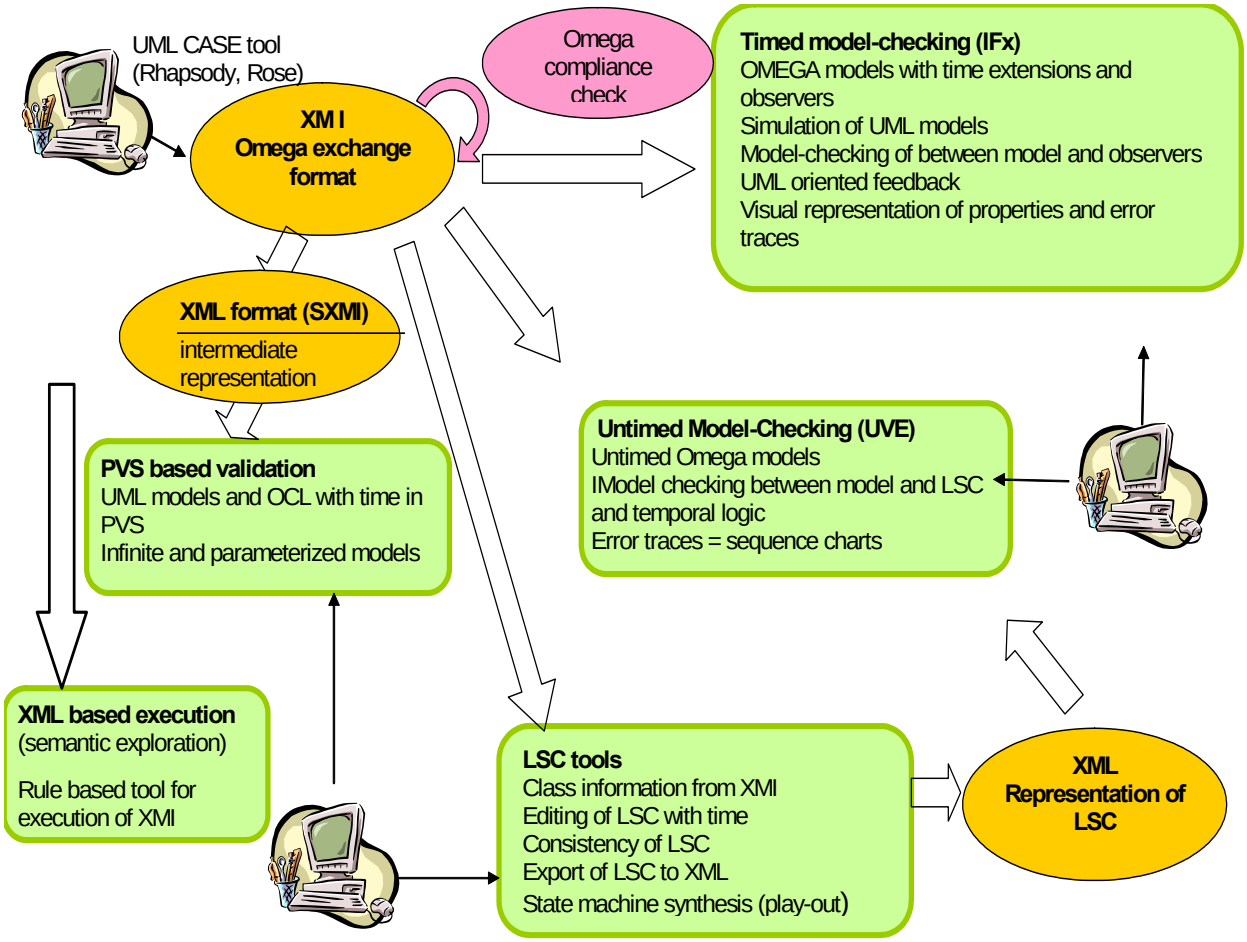


Figure 1: Overview on the toolset

is provided by **Figure 1**. It shows the possible flows and interactions of the user with the set of tools.

The tool integration is obtained by the use of common formats at model level. Each tool extracts some information from the common model for its particular analysis. We have chosen this type of integration based on a few common formats as earlier experiences showed that a tighter integration is often difficult to achieve and is fragile, in the sense that evolution will break the integration. In our setting, the different tools can be used even independently of each other if not all types of analysis are required. From the point of view of the user, the interaction with the tools is as follows:

- He elaborates a model with a UML case tool – in the project we have used and tested the tools Rose from Rational/IBM and Rhapsody from Ilogix – by respecting the Omega constraints and syntactic conventions, and he exports the model in XMI using the tool's exporter.

- The generated XMI model can be analysed for compliance with the Omega profile using a tool called *xmicheck* which can be used independently from the verification tools and which is invoked from shell.
- When using the untimed model-checker of an operational model, the user will open the UVE tool, import the XMI, edit some properties in one of the tool internal property editors or import LSC generated by the LSC Play Engine and verify the properties on the model. Methodological guidelines are provided in [D3.3A1].
- The IFx/IF tool is composed of several components which can be used in different ways. The most common workflow is the following: the XMI file containing the UML model **and** the properties to be verified (UML observers) is compiled into an IF specification (using *uml2if*). The IF specification may be simplified (e.g., for eliminating dead variables and code) using the static analysis tool *dfa*, which yields another, equivalent or more abstract, IF specification. The IF specification is further compiled into an executable (simulator) which can be used either for interactively simulating the model or for generating the whole system state space and searching for (observer's) error states. All these phases may be performed either in command line or using the IFx GUI (which is also the GUI for performing interactive simulation). The methodology for working with the tool is explained in [D3.3A2]
- For using the LSC Play Engine in combination with UML, the UML model needs only to provide a class diagram defining the object structure, signals and methods which can be imported by the tool. The Play Engine is an interactive graphical editing, simulation and validation tool for requirements in the form of Live Sequence Charts (LSC). Methodological guidelines are provided in [D3.3A31].
- Users of the PVS based tools also start by importing an XMI file of a concrete UML model. Via the intermediate SUML format, the XMI file is translated into a representation of the UML model in the typed higher-order logic of PVS. This model may include OCL specifications (as comments) which are translated separately into specifications in PVS. Alternatively, the user may express specifications directly in the specification language of PVS. The generated PVS files representing the concrete model are imported by general PVS theories that define the semantics of UML models in general. This defines the set of all runs of the UML model and the user can start proving properties about these runs. To prove a property, basically the *(x)emacs* interface of the PVS proof checker is used (although there are a few Tcl/Tk-style windows that are more user friendly). The user types in proof commands that may gradually reduce the proof goal until it is completed. Clearly, this requires experienced users. Some methodological guidelines on verification with PVS in OMEGA can be found in [D3.3A41].

All tools provide as feedback some evidence for property satisfaction or non satisfaction, simulators and model-checkers in terms of (un)successful traces and provers in terms of proofs or non proved verification conditions. The analysis of the model and the error traces and the appropriate modification of the properties and/or the UML model are done by the user using the appropriate tool.

A more detailed overview on the profile is given in section 4, and an overview on the functionalities and restrictions of the individual tools in section 5.

3.3 Workflow for the considered profile and tools

The work in Omega is based on a simple UML-based development process, which represents the basic process of most methods that are currently used in practice. As a starting point, we consider an iterative, incremental development process, which can be used to indicate how we intend to support UML-based development by formal methods and tools. The basic activities of this process are:

1. Specify and analyse *requirements*
2. Define an *architecture*

   where the first two points may go hand in hand, as shown for example by the fact that the formalisation of the requirements using LSC, OCL or observers, supposes the existence of a vocabulary provided by an initial architectural decomposition[2].

3. Iterate the following steps, for an increasing part of the system under development:
   3.1. *Design* a part of the system
   3.2. Refine this design until it is close to a concrete *implementation*
   3.3. Realize a version of the system on a concrete platform
   3.4. Evaluate the current version and select a new part.

Note also that this process is only used as a stepping-stone to illustrate the Omega techniques; it is certainly not the only process that can be supported by our techniques. But it provides a useful framework for a discussion about the place of the Omega tools in the development process and their relation.

Timing occurs early in the process. In real-time and embedded systems, timing constraints are often even a part of the requirements. In addition, especially as systems are rarely built totally from scratch, some assumptions or knowledge on platform and the physical architecture induced timing constraints can be considered at an early stage of design, allowing early architecture validation or, if needed, re-architecturing.

In the case studies, we have done only limited design space exploration; we have played with variations of the time constraints of the case studies which all have been obtained more or less as a posteriori models built by reotro engineering from existing systems.

[2] Indeed, requirements are expressed often in terms of some inner structure, inner components and interfaces.

Formal Support for the Development Process

Table 1 lists the concepts that are used to describe the results of the four workflows of our development process: requirements, architecture, design, and implementation. We mainly use UML-based concepts, which are extended or modified where appropriate. Moreover, we indicate the formal techniques developed within Omega to support these activities

Core Workflows	Omega/UML - concepts	Omega support for validation
Requirements	Use Case Diagrams (informal) Live Sequence Charts (LSC) OCL State Machines (including observers)	Play-out approach using LSC Internal consistency of LSC, OCL specs Refinements of specifications Deduction and verification of properties
Architecture	Components Diagrams (e.g., with required and provided interfaces), LSC, OCL, (Protocol) State Machines	Correctness wrt requirements Compositional verification Timing analysis Refinement of architecture properties
Design	Class Diagrams, OCL, State Machines, LSC	Correctness wrt (component) specs Synthesis of State Diagrams Correctness of refinement steps, within and between iterations.
Implementation	Deployment Diagram	Scheduler verification (and synthesis)

Table 1: Overview Omega development process and formal support

3.4 References concerning the general methodology

[Ome] The Omega web page, http://www-omega.imag.fr/

[Acc] **Agnes Lanusse, Sébastien Gérard, François Terrier**, "Real-time Modeling with UML: The ACCORD Approach", in UML'98, LNCS 1618

[D3.3] **Jozef Hooman** Editor, "Omega General Methodology", Feb 2005

[D3.3A1] **Angelika Votintseva**, "Deliverable D3.3 Annex1: General methodology, un-timed verification", Feb. 2005

[D3.3A2] **Iulian Ober**, "Deliverable D3.3 Annex 2: Specification and verification of real-time systems using the Omega real-time profile and the IF verification tool", Feb. 2005

[D3.2A31] **David Harel, Hillel Kugler and Gera Weiss** "Deliverable D3.3 Annex31: Some Methodological Observations Resulting from Experience Using LSCs and the Play-In/Play-Out Approach", Feb. 2005

[D3.3A41] **T. Arons, J. Hooman, H. Kugler, A. Pnueli, M. van der Zwaag** "Deliverable D3.3 Annex41: Deductive Verification of UML Models in TLPVS", Feb. 2005

[Ilo] **Ilogix**, The Rhapsody development environment, http://www.ilogix.com/

4 OMEGA UML profile for real-time and embedded systems and its semantics

This section discusses the UML profile chosen in the project and its semantics; it answers some questions, concerning the choice of UML and the particular profile chosen and about the problems concerning semantics.

4.1 UML profile

The choice of the subset of UML that we have made can be considered as relatively standard with respect to the choices of commercial UML tools for simulation and/or code generation for real-time and embedded systems, such as Rhapsody, Telelogic TAU and Rose Real-Time, emerging from ROOM. These tools mainly focus on what is called a "*platform independent*" description of the system, that is a model focussing on the software structure and on the functionality of an application, independently of the middleware, OS, hardware architecture,… it is going to be executed on.
A so-called "*platform dependent*" model should provide in addition, enough information to generate code or to analyze non-functional aspects for a given platform.
Notice that the UML *profile for Scheduling, Performance and Time* (SPT) [SPT03] – focuses like we do – on the analysis of time related properties; to this aim, it provides concepts for defining different kinds of "resources", "task" needing a resource to be executed and consuming some quantities, in particular time.
An aim of the Omega project is to handle these two aspects less independently as they are in most existing tools. The aim is to use less violent abstractions of one aspect when verifying the other, in cases where this is needed.

The profile has been defined in phases. First, a so called Kernel Model, has been defined, representing a useful operational subset, rich enough, to start the work on the tools and the case studies. In particular, the combined modelling synchronous and asynchronous parts of a system, is an important issue in some of the case studies. The criteria for the choice of this profile are the usefulness for the potential users, the availability of the chosen notations in the CASE tools, the semantic choices of existing UML tools and the possibility to provide rapidly some verification tool support for the chosen notations. In a second phase, notations for the expression of time constraints, of requirements and for structuring models are defined.

4.1.1 Operational profile and Kernel Model

For the OMEGA *Kernel Model*, we have chosen, like the considered CASE tools, a relatively complete subset of the operational part of UML, where

- The static structure of the system is described in terms of a class diagram with only a few restrictions, where associations between classes express inclusion or accessibility.
- In particular, like the standard profile, we distinguish between active and passive classes, but with a particular interpretation: the behaviour of an active class and all the classes owned or created by it, represent a mono-threaded behaviour, executing request in a run-to-completion fashion. This notion of activity group is also used in Rhapsody, and is similar to the notion of process in SDL or capsule in ROOM.
- Communication between objects is either via asynchronous signals or via synchronous operation calls, where we distinguish between primitive operations

which are executed by the calling thread and normal operations which are scheduled by the active object of the activity group.

- The behaviour of the system is described by means of an explicit imperative action language which can be used in combination with a form of state machine notation for describing transition systems extended with data, communication and object creation.

4.1.2 Real-time extensions and observers

UML 1.4 includes no support for real-time, but a profile for Scheduling Performance and Time (called SPT profile) had been defined, including some extensions for the expression of timing properties, mainly in the form of tag values. We have defined a real-time profile that respects the SPT profile which takes over most of its basic concepts, defines a concrete syntax where this is missing and specifies the usage. Also, contrary to the SPT where time constraints are mainly expressed at instance level, the Omega real-time profile enables time constraints at class level.

- We have introduced the notions of *timer* and *clock*, as they exist in other modelling formalism and as they have been introduced in UML 2.0. Timers can be set and deactivated and cause a "timeout event" after a specified duration and clocks can be set and deactivated, and when active, they count the duration since they have been set for the last time. This allows the definition of time dependent behaviour by means of new primitives in the action language.
- We provide syntactic access to semantic level events. A semantic level event is a state change in the underlying dynamic semantic. Each syntactic construct may have to 0, 1 or more semantic level events associated. With a state an *enter* and an *exit* event is associated, with a signal transmission, a *send*, a *receive* and a *consume* event. A syntax is defined for identifying all state changes. An event can be defined as a class stereotyped <<event>> with predefined parameters depending on their type (all events have an occurrence time, a send-signal event has a sender, a receiver, a signal type, signal type depending parameters) and possibly user defined attributes. Moreover, there are means to refer to different occurrences of an event in a given execution or prefix of it.
- Expressions of the type duration which can be used in time constraints can be defined simply using arithmetic expressions on clocks and the occurrence time attribute of different events (this is the access to time and duration used in OCL, see below and in observers, see next item), by a set of predefined duration expressions
- Time constraints may be arbitrary Boolean expressions depending on time and duration expressions, but we consider only linear constraints in all our tools. Time constraints can occur
    - In the form of *guards* in state machines and observers,
    - conditions in LSC,
    - OCL constraints
    - explicit time constraints or constraint patterns associated with different UML construct, as they are foreseen in SPT (WCET associated with methods, *minDelay*, *maxDelay* associated with channels,…) ; notice that such derived constraints are presently not implemented in the tools, they have to expressed using the basic means for expression of time constraints

- We have introduced ***observers*** mainly as a means for expressing complex time constraints using a UML operational syntax, accessible to the user. They are defined as stereotyped of state machines, where transitions are triggered by semantic level ***event occurrences*** (they can be identified using explicitly defined event instances or by using an event matching clause as in the definition of a corresponding event class). An observer allows expressing constraints on the order and/or timing of occurrence of semantic level events and is a means to define dynamic properties depending on time or not. Observers have proven to be well accepted by users. They express safety properties in the form of acceptors (an execution leading to an <<error>> state represents an *error trace*). In addition, observers can express assumptions (a sequence leading to an <<invalid>> state is "not to be considered").
- Finally, some minimal concepts have been introduced to define scheduling constraints and general scheduling policies.

4.1.3 OCL

OCL has been developed as a constraint and query language for static UML models and does therefore not adequately capture the dynamic behaviour of objects and systems. To overcome this short-coming, we distinguish between local constraints, which are used to specify the behaviour of an object without referring to the context in which it is used, and global constraints, which are used to specify the context in which object occur and how objects are connected. This distinction is achieved by using the additional stereotypes <<local>> and <<global>> which are attached to constraints. Additionally, we defined a trace logic in OCL which specifies an object's behaviour in terms of constraints of traces of events it can observe. The trace logic is a conservative extension of OCL 2.0, where we provide data types for events, very similar to the events presented in the preceding section, and a logical constant "trace" of type Sequence of Events which designates the local trace. For global specifications, we introduced a corresponding global assertion language, which is a generalisation of the local assertion language. Finally, these assertion languages include mechanisms for specifying global constraints by providing component and system contexts. This is necessary, because OCL 2.0 assumes that the context, in which a constraint is to be evaluated, is an object and not a collection of objects.
This extension of OCL allows one to capture all safety properties of an object in a way similar to LSC or sequence diagrams, without being bound to refer or provide an objects environment.

4.1.4 Component model

In OMEGA, we have anticipated the UML2.0 component model by using the notations provided by the profiles available presently in tools. Components encapsulate their internal description and interact only through a certain kind of objects which are called ports. Ports are instances of internal classes which are represented by roles. Roles export information about the required and provided operations of these classes by means of interfaces.

Another distinguishing feature of the OMEGA component model is that ports can dynamically instantiate their associated required interfaces which are used to represent external classes belonging to other components. Connectors wire roles of different

components together to form a component-based application, and a required interface ***I*** acts as placeholder for the external class realizing the role wired to ***I***.

Notice that Components in UML 2.0 do not encapsulate their internal structure. In OMEGA a component does encapsulate its internal class structure because in OMEGA we have defined the relation between the internal class structure of a component and its (provided and required) interfaces at the level of the action language for state machines. A characteristic feature of this relation is that the state machines describing the behaviour of ports in general contain actions for instantiating required interfaces. By the component connectors these required interfaces are associated to the classes describing the ports of another component. Consequently, component connectors in OMEGA allow for the inter-component dynamic creation of ports.

OMEGA components are used to structure sets of classes and to support a modelling discipline based on interfaces. Based on the connections provided by a component system diagram, we formalize the semantics in terms of the semantics of the underlying class structure. The behaviour is defined as the concurrent behaviour of the objects living in the component and renaming the required interfaces in the corresponding state-machines by their realizations as specified by the connections.

The CASE tools Rhapsody and Rational Rose do not support components yet. As a workaround, a component based design can be done within the Kernel Model Language by modelling components as classes. One can also associate LSCs and OCL assertions to components to specify the overall behaviour of their ports. One can also associate with each role of a component a state machine describing the externally observable behaviour of its instances. The resulting set of state machines describes the overall behaviour of the ports of a component. The interactions between the ports of different components then can be model-checked by the OMEGA tools.

OMEGA also started preliminary work on a compositional semantics of components. Such a semantics forms the basis for the further development of compositional verification techniques which allow separating the verification of the observable interactions between components from the verification of their implementation.

4.1.5 Live Sequence Charts

Sequence diagrams are widely used by UML users, but their UML 1.4 version is not expressive enough, as they can describe only a particular (desired or forbidden) execution of a particular set of instances up to the order of independent events.
For this reason, we have chosen Live Sequence Charts [DH99], a formalism extending the existing versions of Sequence charts (MSC, High Level MSC,…) by adding

- Quantifiers, stating that either there exist an execution or all executions with a certain prefix are compatible with a given chart (distinction of e*xistential- or possible* - and *universal – or mandatory* -charts)
- Liveness constraints by marking certain events “mandatory” (so called *hot* conditions) to distinguish which observations of prefixes are considered to satisfy the chart and which ones not.

These Live Sequence charts have been extended in Omega for taking into account Object Orientation and timing constraints:

- The Global time progress supposed in Omega is represented by an external event *tick*, representing time progress by one time unit. LSC specifications can store time in variables and conditions can contain constraints on such variables, similar as in timed automata. Due to the distinction between mandatory and non mandatory conditions, this allows to distinguish between reaction to external time progress and time dependent requirements.
- The extension to object orientation and dynamic systems is obtained by allowing the interpretation of "life lines" as classes (a set of potential instances) and the introduction of quantifiers on life lines. This allows distinguishing the case where the event specified by the LSC should be observed in all existing instances of a class at the instant of occurrence of the corresponding event, and the case where the behaviour must be observed in at least one such instance.

A set ***L*** of events traces and conditions *satisfies* a set of LSC charts if each trace in ***L*** satisfies all universal charts and for each existential chart there exists at least one trace satisfying it. A trace in ***L*** satisfies a universal chart, if in its projection on the set ***A*** of events and conditions observed by the chart is of the form

$$(A^*\text{-prechart})^\infty \cup ((\mathbf{A}^*\text{-prechart});\text{prechart};\text{mainchart})^\infty$$

A detailed description of the semantics of LSC can be found in [D.1.2.2bis].

4.1.6 Availability of the profile

The profile is defined by a set of documents describing a set of admitted constructs and other restrictions, a set of stereotypes and tag values that can be used and their meaning, as well as a library containing the definition of time-related data types (available in Rational Rose and in I-Logix Rhapsody formats). Pointers to all parts of the profile are available at http://www-omega.imag.fr/profile.php

1. Operational Kernel model: the kernel model is defined by some restrictions of the UML 1.4 profile and a few extensions with stereotypes and predefined tag values. Descriptions can be found in the publications [DJP*02, DJP*05]. All the tools developed in OMEGA are based on the kernel model and consider at least a subset of it. The document [Syntax] provides an overview on the accepted syntax, including the Omega Action Language OMAL which is accepted by the Omega tools.
2. Timing extensions: The time extensions consist mainly in the definition of stereotypes which are described in [GOO04] and in the syntax document [Syntax]. We have defined a library that contains the definition of time-related data types. The library is part of the IFx distribution.
3. OCL: The OCL extensions are defined as a conservative extension of OCL2.0, where the extensions are described in [KdB03a, KdB03b, KdB04]. It tries to subsume part of the expression language of the timing extensions, but uses a different syntax. Its syntax is defined in the document [Syntax].
4. Component model: The component model is defined mainly in terms of syntax conventions which are described in the document [Syntax].
5. LSC: LSC are a particular form of sequence diagrams, originally defined in [DH01] explaining graphical syntax. More recent descriptions with extensions are defined in [HM02, MHK02, LSCuser04, D1.2.2-b]. Graphical editors are implemented in Weizmann's PlayEngine and in OFFIS' UVE tool.

4.2 Semantics

Consistency of semantics

We have defined a formal semantics for all parts of the system in terms of sequences of the above defined semantic level events, where time extensions add an occurrence time to events. The problem solved by the tools, is answering a question of the form

$$\textit{environment assumption} + \textit{mode l} \models \textit{property} \; ?$$

where *environment*, *model* and *property* may be expressed using different formalisms, but for each one a single one is used. The question can be reduced to a question of the form

$$\mathbf{L}(\textit{environment assumption} + \textit{model}) \subseteq \mathbf{L}(\textit{property}) \; ?$$

and this is well defined if the (timed and untimed) semantics of the formalisms used for models and properties are defined independently. Ensuring consistency is now, as usual, the problem of the tool builder to correctly implement an algorithm solving this question.

Semantic choices

Notice that one of the aims of OMEGA was to provide a new profile for Real-time and embedded systems which extends the expressivity of the currently existing tools. For this reason, today, the user can not use the analysis tools together with the code generator of his CASE tool as none of them generates code in accordance with the Omega profile.
The questions about semantic choices were discussed mainly for the operational Kernel model and for the time extensions.

One motivation of the project was the definition of a profile and semantic framework appropriate for the description of mixed synchronous/asynchronous systems. In this context arose the question on how to handle the non determinism induced by concurrency and if and how to restrict non determinism of time progress.

- Synchronous approaches impose often deterministic time progress (in fact maximal system progress) whereas asynchronous models in general assume external time (non controllable time progress). Timed automata with urgency allow any kind of control over time progress, but earlier experience showed that explicit transition urgencies are not accepted by users. For time extended operational specification, this lead us to the proposal of some choices of urgency modes handling this issue implicitly, and it turned out that this was sufficient for the considered case studies. In LSC, one of these options is chosen, and time progress is determined by the environment, where the environment is only taken into account in stable states and only one environment event is available at a time. In OCL, this is somehow a non issue as it is used only in a declarative manner. When working only with OCL without the existence of a operational model, which is what we have mainly done, restrictions on non determinism and time progress can be imposed by a set of axioms[3].

[3] Such as those proposed in [GP05] in the context of abstract state machines

- As a result, using the Kernel model with the notion of activity group (extended with time), we can accommodate so-called GALS (Globally asynchronous, locally synchronous Systems). Nevertheless, this needs still some undesired workaround for those users used to the use of synchronous languages, such as Esterel, Lustre or Signal.
- We have looked into the interest of general coordination frameworks, and we have developed a component model based on a very general notion of interaction between a possibly variable number of components based on lattices of interactions complete interactions. We have showed that this framework allows a more direct integration of synchronous and asynchronous systems [GS02, GS03, GS04]. Nevertheless, it was not possible to integrate this work directly into our UML profile as it would have required in addition a different type of diagrams not accessible through existing UML tools.

The work on semantics generated several unexpected problems and lead to some reflections on how to deal with them. We have obtained some insights, but there are still many open questions:

1. It turned out that the initially provided version of the semantics, that everybody believed to understand in a first approximation, led to many misinterpretations and almost endless discussions amongst the partners. This motivated on one hand, the elaboration of several abstract semantics on more restricted subsets of the profile. It motivated also the work on the semantic exploration tool RML and some more long term reflections on how to represent semantic choices in an easy to understand way.
    - An implementation of the chosen semantics, such as they are provided by the RML and the IFx tool together with a well chosen set of examples are probably a good way to demonstrate by examples the user what the semantics of a tool is, but this can not be the only information given to the user – just like use cases, it will never be complete.
    - The usage of explicit priority rules makes semantic variations easily recognizable if these variations correspond to a particular elimination of non determinism.
    - Abstract semantics (of small sub-calculi) can also be helpful in this respect, but there strength is rather to provide insight about the essentials of the considered calculus or to help understanding the consequences of the choice between to options for a given variation point. In particular, we have developed an abstract class calculus [ABBS04].
2. The previous problem motivated us to try to identify all potential semantic variation points. It was already clear from the experience with state chart semantics that the number of variation points is very high in presence of parallel and deep history states and therefore we left these features aside in a first approach. A study provided in deliverable [D.1.1.4] shows that the possible variation points concerning method invocation is also very high; moreover, the number of variation points increases considerably with the granularity. This shows that it is probably not practicable to provide the user an explicit choice for all variation points.
3. As one of the objectives of the project was to enable a compositional approach to verification, some effort was spent to provide a compositional semantics instead of the initial global semantics. The difficulty to provide such a semantics comes from the aim to design a semantics which is both compositional and abstract; this

provides information on the minimal information needed for the verification of a set of relevant properties, and indeed, we obtained some interesting results here.
In OMEGA, a formal theory has been developed for reasoning compositionally about the behaviour of a system in terms of its class invariants. A class invariant describes in a generic manner the local communication traces of the instances of a class. The behaviour of the system is given as a set of global communication traces. In general, communication traces formalize message sequence charts in UML and abstract from the actual creation of objects.
The compositional proof theory provides an axiomatic characterisation of unbounded class instantiation at the level of abstraction provided by the communication traces. Compositional verification techniques based on communication traces have been applied successfully to the MARS example.

4.3 References concerning profile and semantics

[ABBS04] **Erika Ábrahám, Marcello M. Bonsangue, Frank S. de Boer, Martin Steffen** Object Connectivity and Full Abstraction for a Concurrent Calculus of Classes In *To appear in the LNCS Proceedings of the First International Colloquium on Theoretical Aspects of Computing, ICTAC 2004*

[D.1.1.4] **Harald Fecher, Marcel Kyas, Frank de Boer**, Variation Points of the Semantics concerning Methods, December 2003

[D1.2.2-b] **D. Harel, H. Kugler, R. Marelly, A. Votintseva, J. Klose, B. Westphal**, Deliverable D1.2.2-b, Live Sequence charts for UML and their semantics

[DJP*02] **Werner Damm, Bernhard Josko, Amir Pnueli, Angelika Votintseva** Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML In Frank de Boer, Marcello Bonsangue, Susanne Graf, Willem-Paul de Roever (Eds.) *Proceedings of the 1st Symposium on Formal Methods for Components and Objects (FMCO 2002)* LNCS Tutorials vol. 2852 2003

[DJP*05] **Werner Damm, Bernhard Josko, Amir Pnueli, Angelika Votintseva** A discrete-time UML semantics for concurrency and communication in safety-critical applications In *Science of Computer Programming* 2005

[DH01] **W. Damm and D. Harel**. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design, 19(1):45 -- 80, July 2001*

[GOO04] **Susanne Graf, Ileana Ober, Iulian Ober** Timed annotations in UML *accepted to STTT, Int. Journal on Software Tools for Technology Transfer* Springer Verl. 2004

[GS02] **Joseph Sifakis**. Scheduler Modelling Based on the Controller Synthesis Paradigm In *Journal of Real-Time Systems, special issue on Control Approaches to Real-Time Computing* vol. 23 2002

[GS03] **Gregor Gössler, Joseph Sifakis** Composition for Component-Based Modeling In *1st Symposium on Formal Methods for Components and Objects, revised lectures* LNCS Tutorials vol. 2852 2003

[GS04] **Gregor Gössler, Joseph Sifakis** Priority systems In *proceedings of FMCO'03* LNCS 3188 2004

[GP05] **Susanne Graf, Andreas Prinz** Time in ASMs - Some problems and solutions In *ASM 2005, to appear in LNCS,* 2005

[HM02] **D. Harel, R. Marelly** Playing with Time: On the Specification and Execution of Time-Enriched LSC In *Proc. 10th IEEE/ACM Int. Symp. on Modelling,*

Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002), Fort Worth, Texas 2002

[MHK02] **R. Marelly, D. Harel, H. Kugler** Multiple Instances and Symbolic Variables in Executable Sequence Charts In *Proc. 17th Ann. ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'02)* 2002

[LSCuser04] **D. Harel and R. Marelly**, "Play-Engine User's Guide" Play-Book

[KdB03a] **Marcel Kyas, Frank de Boer**, Assertion Languages for Object Structures in UML, Omega Deliverable D1.2.1, 9. January 2003

[KdB03b] **Marcel Kyas, Frank de Boer**, Addendum to Assertion Languages for Object Structures in UML, Omega Deliverable D1.2.1b, 18. July 2003

[KdB04] **Marcel Kyas, Frank de Boer**, On Message Specification in UML, In: de Boer, Frank S. and Bonsangue, Marcello (eds.) Compositional Verification in UML, ENTCS vol. 101, 2004

[Syntax] **Marcel Kyas, Joost Jacob, Ileana Ober, Iulian Ober, Angelika Votintseva**, OMEGA syntax for users, Omega Deliverable D2.2.3 Annex 1. January 2005.

External References

[SPT03] **OMG**, Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0 March 2002

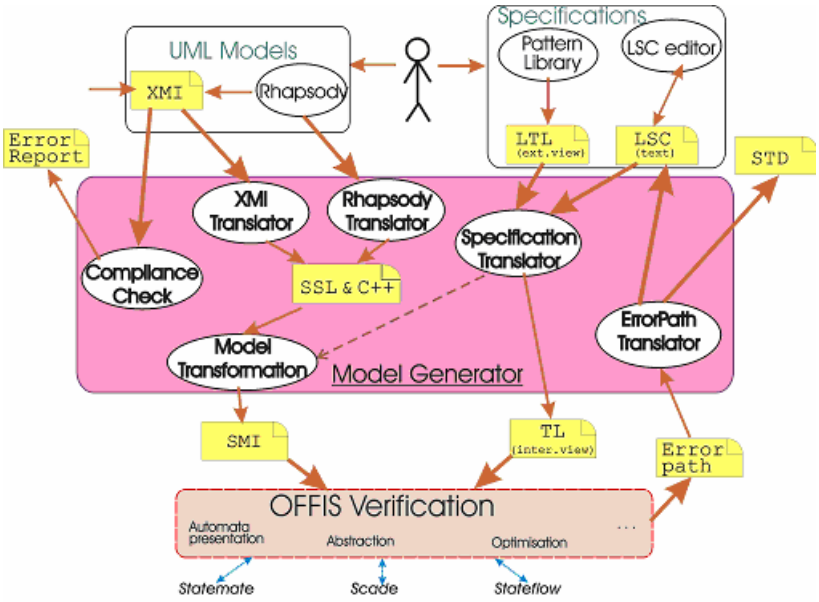
5 OMEGA Tool set for validation of UML specifications

5.1 Overview on the tool set

This section gives a short overview on the functionalities and interfaces of the individual tools, as well as the subset of the profile they are accepting. The way in which they are integrated can be seen from the global toolset picture given in section 3 and from the specification of interfaces in section 5.3. A methodology using the different tools in combination is presented in section 6.5. A short overview on all tools developed in OMEGA can be found on the OMEGA webpage as well as in an overview paper [GH04].

5.1.1 Untimed Verification tool UVE

The UVE tool (UML Verification Environment) serves to check functional and dynamic properties of the Omega kernel model - structure, behaviour and the order of the object communication - combining them into (temporal logic) formulas. It can be applied at the design and implementation phases for the component verification when real-time constraints are not yet specified. In cases where this makes sense, requirements can refer to the number of steps in the model execution, thus achieving a kind of discrete time. The most elements of the UML object-oriented features in class diagrams and state machines, a subset of C++, a subset of CTL, LSCs, parameterized environment, tuning verification parameters are covered by this tool set. A more detailed description is published in [STMW04].



The main functionality of UVE is the following:

- Verification of a set of temporal logic formulas (defined via the provided patterns): check of reachability, invariance, liveness, safety etc.
- Verification of LSCs: a compliance check between specifications and a design.
- Sequence diagrams generation:
  - (a) as witness-paths for properties reachability and existential LSCs;
  - (b) as counterexamples - error-paths - for so called *invariant* properties such as, e.g., universal LSCs.
- Results visualization with symbolic timing diagrams (STDs) and LSCs.
- Verification of requirements under different kinds of assumptions, restricting the non-determinism of the environment or of the system behaviour (e.g., not yet implemented parts).

UVE consists of two components:

- Rhapsody-based, RUVE: the development was started in the AIT-WOODDES project and has been extended within the OMEGA project with respect to several features: extending the supported UML set in particular regarding object-oriented elements, extending the formulization of properties (e.g. introducing LSC specifications) as well as improving the verification engine using optimization and abstraction techniques;
- XMI based - XUVE - developed in the OMEGA project. In addition to the features covered by RUVE, XUVE adds the following functionality:
  - the semantics defined in OMEGA with non-determinism between *concurrent regions* in statecharts and non-determinism between *enabled transitions*;
  - OMEGA Action Language (in addition to C++) with extended constructs for non-deterministic choice and concurrency;
  - Two possibilities of the fine-tuning and invocation of the verification process: using the Rhapsody graphical interface or from a command line without a UML tool.

The tool-set has been partially extended with the means to derive symmetry property of the whole model from the properties of its parts. This tool-extension is intended to be used to reduce verification complexity as well as for the verification of unbounded models.

The tool has been applied on two of the Omega case studies:

- MARS case study (NLR), verifying 4 main un-timed properties in different versions represented as logical patterns and some of them as LSCs.
- Sensor Voting an Monitoring case study (IAI), verifying 2 main algorithmic properties with different assumptions.

The integration of the UVE toolset in other tools was performed in the following directions:

- Translation of a UML model into the XMI exchange format and XMI-based verification
- LSC translation from and to the XML exchange format
- Integration into the commercial CASE-tool Rhapsody in C++

5.1.2 IF/IFx tool for verification of timing and dynamic properties

Within the Omega project, VERIMAG has developed the IFx toolset for timed verification, simulation and scheduling analysis of Omega-UML models. The approach that was chosen is to *reuse* the timed validation techniques that VERIMAG developed for dynamic communicating timed automata extended with data and actions, as well as the already existing IF toolbox which implements state-of-the-art validation and verification techniques. A more detailed description focusing on the IF language – including extensions made for the purpose of handling UML - and tool can be found in [BGM02, BGOOS04] and one focusing on the front-end for UML in [OGO05]. The main functionalities provided by IF/IFx are:

- ***Simulation*** allows the user to interactively explore a model's execution graph. The user may perform operations that are similar to those offered by advanced debuggers: step by step execution, inspection of the system state, conditional breakpoints, scenario rewind/replay, manual resolution of non-determinism, control of scheduling policy and time related parameters, etc.
- Verification of simple ***consistency*** conditions like deadlocks, timelocks and satisfaction of state invariants.
- **Verification of *dynamic and timing properties*** using the model-checkers provided by the IF tool. The properties may be expressed within the UML editor by means of the following notations provided by the Omega UML profile:
  - *observer classes* : classes with special state machines reacting to events and conditions occurring in the system execution
  - *timing constraints* : constraints on durations between system events

  Furthermore, all the property expression formalisms of the tools connected with IF can be used, in particular μ-calculus formulas, but they require knowledge about the entities generated by the translation of UML to IF and are reserved to specialists.

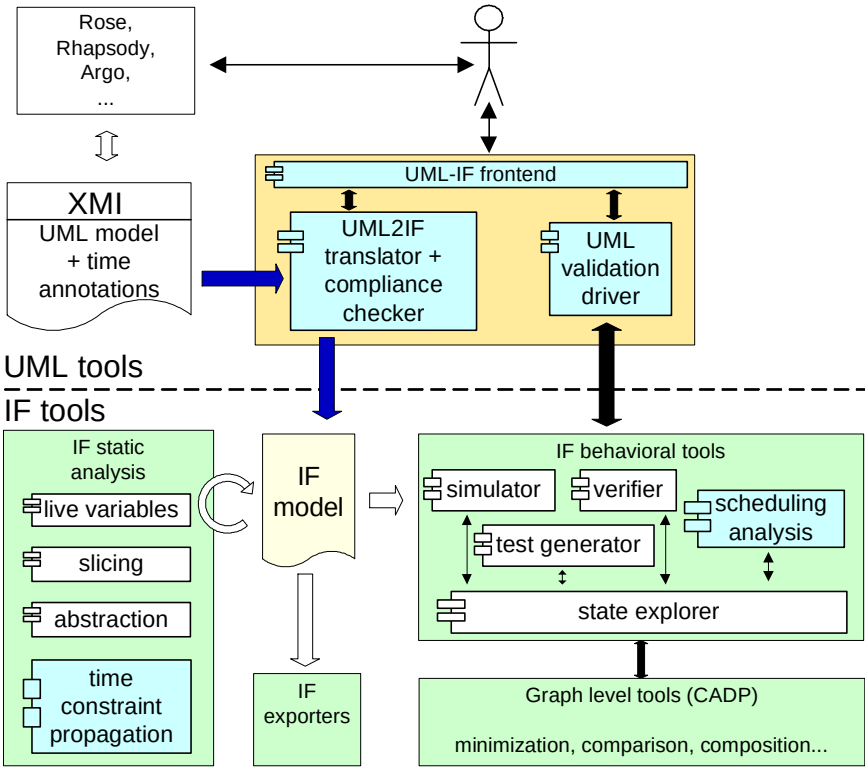


Figure 2. Architecture of the IFx toolset

Finally, verification techniques implemented in the tool allow performing ***scheduling analysis***, by specifying scheduling objectives (e.g., deadlines) as properties to be verified (observers). Other types of functionality, like comparing models modulo simulation and bisimulation relations, are available through the connections provided by the IF toolset to external tools.

The architecture of the IF/IFx toolset is depicted in Figure 2 above. The upper part shows the UML tools specific to IFx, while the lower part shows the components of the IF toolset, including some modules developed in the OMEGA project (in blue). The main components of IFx (in addition to the IF toolset), are:

- The UML-to-IF translator which takes as input a UML model stored in XMI format. The model may use standard UML constructs and extensions defined by the Omega profile: actions written in the Omega action language, timing annotations and observers expressing model properties. The translator generates an IF specification corresponding to the UML model, according to the Omega semantics.
- The UML front-end provides an interface specifically targeted at UML modellers for the IF validation tools. The interface hides IF and the details of translation and presents simulation and verification results in the vocabulary of the initial UML model. The interface supports all compilation and simulation features mentioned before, and offers customizable views on the analyzed system.

The tool has been applied on three of the Omega case studies:

- On the Ariane-5 case study (EADS), to statically validate the well-formedness of the model, to prove 9 safety properties of the flight regulation and configuration components, and to analyze the schedulability under the assumption of fixed priority pre-emptive scheduling policy.

- On the MARS case study (NLR), to prove 4 safety properties and to discover reactivity limits of some system components and fine-tune their behaviour in order to improve reactivity. On this case study, we have also applied compositional verification which is partly supported by the tool through the existence of simulation checkers, minimization with respect to bisimulation and abstraction.
- On the Sensor Voting an Monitoring case study (IAI), to prove 4 safety properties and timing properties

The IF/IFx tool is freely distributed on the web (either through the Omega webpage http://www-omega.imag.fr/tools.php or http://www-verimag.imag.fr/~async/IF/).

5.1.3 The LSC Play Engine

The LSC tools developed by WIS consist of a set of tool for specifying and executing behavioural requirements by means of LSC, verifying an Omega UML model with respect to LSC requirements and synthesizing statecharts from LSC. A more detailed description of the tool can be found in [LSCuser04] and at http://www.wisdom.weizmann.ac.il/%7Eplaybook/.

Play-In

The main idea of the *play-in* process is to allow requirements engineering at a high level of abstraction, and to work with a look-alike version of the system under development. This enables people who are unfamiliar with LSCs, or who do not want to work with a formal language directly, to specify the behavioural requirements of a system using a graphical interface and an interactive tool. These could include domain experts, application engineers, requirements engineers and potential users.
P*lay-in* means that the system developer first provides the static information of the system (class diagram) and either builds a GUI of the system or uses a predefined one. The user *plays* the GUI by clicking buttons, rotating knobs and sending messages (representing operation calls and signals) to objects, similar to a user of the final system. The user also describes the desired reactions of the system and the conditions that may or must hold. The play-engine records this behaviour in the form of an LSC. For this purpose, it queries the application GUI for its structure and interacts with it, thus manipulating the information entered by the user.

Play-Out

One way of validating or testing requirements is by constructing a prototype intra-object implementation and using model execution for this purpose. Instead, we provide direct simulation, which we call *play-out*: the user *plays* the GUI application as he/she would have done when executing a system model, or the final system, by restricting the interactions to user and external environment actions. While doing this, the play-engine keeps track of the actions and causes other actions and events to occur as dictated by the *universal charts*. The engine traces these actions at the GUI level and thus gives the user feedback on the system evolution. This process of the user operating the GUI application and the play-engine causing it to react according to the LSCs has the effect of working with an executable model, but with no intra-object model having to be built or synthesized.

This makes it easier to let also non software developers participate in the process of debugging the requirements, since they do not need to know anything about the implementation (or its specification). It yields requirements that are well tested, thus lower probability of errors in later phases, which are a lot more expensive to detect and eliminate. Notice that the behaviour played out is not only the one played in, but also *derived* behaviours. Universal charts are used to drive the model, whereas existential charts are used, similar to observers, to express properties or as examples of required interactions and to monitor the system by tracking the events in the chart as they occur.

Smart Play-Out

Play-out is an iterative process, where after each step taken by the user, the play-engine computes a *super-step*, which is a sequence of events carried out by the system as response to the event input by the user. Due to the inherent concurrency, there can be several sequences of events possible as a response to a user event, and some of these may not constitute a *correct* super-step, that is, they may lead to the violation of some active universal chart.
Smart play-out uses model-checking to find a correct super-step if one exists, or proves that there is *none*. We do this by formulating play-out as a verification problem, in such a way that a counter example resulting from the model-checking will constitute the desired super-step. The transition relation is defined so that it allows progress of active universal charts but prevents violations. The property to be checked is one that states that always at least one of the universal charts is active. In order to falsify it, the model-checker searches for a run in which eventually none of the universal charts is active; i.e., all active universal charts completed successfully, and by the definition of the transition relation no violations occurred. Such a counter-example is exactly the desired super-step. If the model-checker manages to verify the property, then no correct super-step exists.
Smart play-out can also verify the possibility to satisfy an existential chart. This cannot be done by exploring a single super-step, since the chart under scrutiny can contain external events, each of which triggers a super-step of the system. Nevertheless, the formulation as a model-checking problem can be used with slight modifications. In this case, we assume that we can choose the appropriate events of the environment.

Statechart Synthesis

A methodology for synthesizing statechart models from scenario-based requirements has been developed. The requirements are given as LSCs. We have implemented our algorithms as a part of the Play-Engine tool and the generated statechart model can then be executed using existing UML case tools.
Due to the intrinsic complexity of this synthesis, we suggest a methodology that is not fully automatic but relies on user interaction and expertise to make the synthesis more efficient. In particular, we ask the requirements engineer to provide enough detail to reduce the number of choices in the model to be synthesized. In fact, the algorithm tries to prove, using the above mentioned verification methods, that some synthesized model (by an extension of smart play-out) satisfies all requirements; if it manages to do so, the synthesized model is a correct one. A major obstacle that requires additional research efforts is the high computational complexity of the synthesis algorithms, preventing scaling of the synthesis approaches to large systems.

5.1.4 PVS based tools and methods

We have built a number of tools allowing the verification of UML/OCL specifications with the help of theorem provers, mainly PVS. An overview on parts of the tool can be found in [AHKPZ04, KFB*04]

SUML

We have implemented a tool that translates a subset of UML, in the XMI format, to the input language of the theorem prover PVS [PVS]. To simplify this process, we use a translation of two XMI *dialects*, namely the one of Rhapsody and the one of ArgoUML, to an intermediate format, called *SUML*.
There is also a translator from SUML to PVS. The tools translating XMI to SUML and PVS are available at http://homepages.cwi.nl/~jacob/uml2pvs.html.

OCL

The OCL Tool implements a translation of a subset of OCL constraints into the input language of the theorem prover PVS via the SUML format.
In order to avoid implementing a three-valued logic within the framework of PVS, we have defined a sound translation of OCL into a two-valued logic. The advantage is a more direct representation of constraints in PVS. The disadvantage is that a set of constraints, which are undefined in OCL, may be provable in PVS.
OCL is a three-valued logic, because the semantics was defined for an executable language and not a logic. If the evaluation of an expression raises an exception or diverges, then the constraint is considered to be undefined. However, such notions do not exist in the declarative semantics of a logic. Finally, we are mostly interested in a *proof* of a property, and for this an operational semantics of OCL is not relevant. As an alternative, constraints can be specified directly in PVS. This also enables the use of TLPVS (see below). The OCL tool is available at
http://www.informatik.uni-kiel.de/~mky/omega/suml.html.

TLPVS

TLPVS is a PVS implementation of a linear temporal logic verification system that has been further developed in Omega. The system includes a set of theories defining a temporal logic, a number of proof rules for proving soundness and response properties, and strategies which aid in conducting the proofs. In addition to implementing a framework for existing rules, we have derived new methods which are particularly useful in a deductive LTL system. A distributed rank rule for the verification of response properties in parameterized systems is presented, and a methodology is detailed for reducing strong fairness to weak fairness. Special attention has been paid to the verification of systems with unbounded number of processes.
TLPVS is available at http://www.wisdom.weizmann.ac.il/~verify/tlpvs/. We also have used it for some examples available at the same page.

Compositional verification

We have defined a general framework for supporting compositional verification using the interactive theorem prover PVS. The focus is on the level of components, and we have concentrated on parallel composition and hiding. The framework is based on timed traces that are an abstraction of the timed semantics of the Omega kernel language. So we abstract from all internal details such as internal objects and the values of their attributes, and only record the current time of the configurations and the external events of the labels. To be able to formalize intermediate stages during the top-

down design of a system, we have constructed a framework where specifications and programming constructs can be mixed freely. The semantics of parallel composition and hiding has been defined in the PVS specification language. Compositional proof rules for parallel composition and hiding have been formulated in PVS and the tool has also been used to prove the soundness of these rules.

OAS

The OAS tool has been designed for experimental analysis of the abstract OMEGA kernel model semantics.
The user provides a class diagram plus an object diagram representing the initial configuration of the model to analyze. Furthermore, the user provides scripts that define the operational semantics in the form of object diagram transformation rules. The scripting language provides also means to describe how the choice amongst several enabled rules is made.
An online version of the tool is available at
http://homepages.cwi.nl/~jacob/km/cgikm.html.

5.2 Overview on work on scheduling and coordination

In Omega, we have carried out a number of studies concerning scheduling and coordination related issues and built tools for handling some of them. Some results concern directly the developed UML profile, already described earlier, others are of more general nature and can be applied to other frameworks.

5.2.1 Fundamental results

We have achieved results on a general framework for high level component composition with the ultimate goal to achieve correctness by construction. We developed a general framework for component composition where individual components are composed using a high level concept of interaction and execution modes or scheduling is expressed by means of dynamic priority rules. The framework allows the definition of atomic interactions of any number of components, and contrary to process algebra like rendez-vous, it foresees to either forbid any partial interactions – as in process algebras – or to allow a subset of them which means that it can, amongst others, it can explicitly handle "*missed rendez-vous*" which no other framework does in a really satisfactory manner. The framework is compositional and allows incremental construction and verification of systems. We hope that this allows the description of system at a high level of abstraction, thus making verification a feasible goal. Moreover, the fact that interactions themselves can be obtained by exchanges of values amongst ports and the execution of actions of individual components in some order provides a basis for constructing implementations of high level models in terms of primitives available on usual platforms preserving liveness of the involved components. We have developed a framework for the construction of systems with guaranteed properties from components. Building a general framework for component composition by preserving properties of components was one of the motivations at the basis of the definition of the Omega. The results can be found in [GS03,GS04]. We have started to implement the composition concept in a small prototype tool.

In OMEGA, we have also shown how to use a (subset) of UML as it is as a coordination language that is based on binary interaction with a clear separation of

concerns between coordination and computation. The basic idea is to use UML as formalism to specify the "*glue code*" in terms of state-machines which are added to the classes of the underlying applications which only provide the implementation of the primitive operations. The added state machines describe the coordination of these primitive operations in terms of sending and receiving events. For example, a state machine can be used to describe the dependency of the execution of a particular primitive operation on the reception of a particular event (within a certain amount of time). The results can be found for example in [GABB04].

We have studied the problem of architectures for adaptive scheduling, in particular how to obtain safe schedulers taking into account *Quality of Service* attributes. The continuation of this work goes beyond Omega, as it focuses on power and memory management [KY03].

We have also studied particular frameworks in which the scheduling problem expressed in terms of difference equations is decidable. This is motivated by the fact that our validation tools for timed systems, Kronos [Yov97] and IF, use difference equations for representing time constraints.

We have worked on scheduling frameworks and scheduler synthesis in collaboration with Ametist IST project[4]. In fact, the partners implied in this work in Omega are also partners of Ametist, where this problematic is a central issue: "*the aim of Ametist is to develop a powerful modelling methodology supported by efficient computerised problem-solving tools for the modelling and analysis of complex, distributed real-time systems. In particular, the project will address problems in connection with time-dependent behaviour and dynamic resource allocation.*" We therefore decided to join forces.
Jointly, we have studied the problem of synthesis of optimal schedulers of acyclic systems under different assumptions [AM03, AAM04,BKM04]. This is motivated by the fact that many embedded systems are modelled as cyclic systems, where each cycle can be considered as cycle free and where scheduling amounts basically schedule each cycle. Under some conditions, these results can also be adapted to the case where there are several "overlapping cycles" at any time, which is the more interesting case in the context of asynchronous systems.

5.2.2 Applications

The main application handled in OMEGA was the schedulability analysis of the EADS case study. It is an instance of the scheduling problem where the abstraction of the system to a simple set of tasks with a worst case execution time is not sufficient. In order to show the schedulability of the system, we had to consider the internal workflow of each task as well as the evolution of the system over time, as over-approximations over all periods lead the conclusion that the system is not schedulable, whereas in fact, it is.

The work on the framework for composition has been used in the tools built in OMEGA; in fact, it strongly influenced the introduction of dynamic priority as a general means to express execution modes. In the context of a PhD thesis, we have

[4] http://ametist.cs.utwente.nl/

introduced an explicit notion of "resource" and means to express dynamic priority rules in the IF language and studied the feasibility of scheduling analysis based on code annotation including timing constraints and scheduling constraints expressed by dynamic priorities has been experimented also in the context of multi-threaded real-time Java applications. A tool, called Jedi [KNY03] has been implemented[5] .

Two applications were done jointly with Ametist. We have used an extension of the IF tool with (dynamic) cost functions associated with transitions to the problem of finding an optimal schedule for the monthly production of lacquers on a given physical production line. We have modelled individual workflows as processes with costs associated with basic tasks and the physical installations as resources and obtained very interesting results by using some optimality preserving heuristics [BM03].
Another case study was based on the use of LSC. Here the idea is to use a semi-automatic strategy based on the use of the play-out engine. If the engine cannot find a schedule automatically, based on output from the play-out engine, the user can refine the set of constraints in order to make the problem easier to solve [KW04].

5.3 Interfaces provided and used by the toolset

In the Omega project, we have defined and used several interfaces for the exchange of models. Moreover, the individual tools provide interfaces for connecting external tools and sometimes use existing connections through these interfaces.

5.3.1 Common format for model representation

- All tools handle in principle the same models as they can be obtained by XMI export from UML case tools using Omega extensions (tag values, stereotypes and libraries), where in the project we have used Rhapsody and Rational Rose mainly.
- Moreover, as XMI does not provide a structured representation of the action language, we have developed an Omega Action Language (OMAL) that is compatible with the UML action semantics.
- For the same reason, OCL is defined as a concrete syntax in accordance with OCL 2.0. For the subset of OCL corresponding also to Boolean expressions of the action language, also the syntax is the same. OCL is only used by the OCL to PVS translator
- As the Meta-model for sequence diagrams of UML 1.4 was too weak for being extended for the representation of Live Sequence Charts (LSC), an appropriate XML format has been defined allowing us to share LSC amongst several tools. This format is used to export LSC from the LSC tools to the UVE untimed model-checking tools. There is currently no transformation of timed LSC into the IFx tool for verification of time dependent models and properties.
- Moreover, a simplified XML format has been defined, back and forth translatable with XMI, used by an XML based interpreter - used mainly for semantic exploration - and as an intermediate format for the translation from XMI into PVS.
- Omega XMI based model representation:
    - XMI 1.0 for models developed in Rhapdosy, XMI 1.1 for models developed in Rational Rose
    - XMI includes textual actions in the Omega Action Langage

[5] see also http://www-verimag.imag.fr/nakhli/jeditool/

  - OCL expressions are placed in a separate file, using ***context*** clauses
- ***SUML***: The SUML (simple UML) format is a simplified XML representation of UML models. There exist tools for the translation from XMI to SUML and from SUML into PVS, using the package of PVS theories mentioned below.
- XML for LSC representation: An XML format has been jointly developed by OFFIS and WIS and appears in deliverable D1.2.2bis. It has been implemented in the LSC tool of OFFIS, whereas some additional implementation effort remains to be done in Weizmann's PlayEngine.

5.3.2 Additional interfaces provided

Interfaces of the IF/IFx tool. The IF tool provides three APIs:

- The Model API which gives access to the abstract syntax tree of an IF specification and allows to write tools that process such specifications. Examples of tools using this interface: the static analysis tools, the simulator generator.
  Using this API, IF is interfaced with other tools like: the TGV test generator, the AMETIST *mincost* path extraction tool, other model checking tools (e.g., for μ-calculus formulas).
- The Simulator API which gives access to the IF execution platform. It provides functions like: driving a specification to the initial state, selecting / executing transitions, dynamically inspecting IF objects (process instances, queues, etc.).
  Based on this API, have been built the IFx interface as well as some other connections to other high-level languages, such as SDL, and other UML profiles. This interface could also allow with a relatively small effort to handle heterogeneous models
- Finally a "*step*" API by which the simulator interacts with individual components. This API can be used to integrate other component based frameworks directly, without passing through IF as an action language. Today this API is exploited for integration of external C or C++ code.

The ***integration of the UVE toolset*** in other tools was performed in the following directions:

- Translation of a UML model into the XMI exchange format and XMI-based verification
- LSC translation from and to the XML exchange format
- Integration into the commercial CASE-tool Rhapsody

5.4 References concerning verification methods and tools

[AHKPZ04] **T. Arons, J. Hooman, H. Kugler, A. Pnueli, M. van der Zwaag** Deductive Verification of UML Models in TLPVS In *Proceedings UML 2004* Springer-Verlag 2004

[AM03] **Y. Abdeddaïm, E. Asarin, O. Maler** On optimal scheduling under uncertainty In *Proceedings of TACAS 2003, Warsaw* LNCS 2003

[AAM04] **Y. Abdeddaïm, E. Asarin, O. Maler** Scheduling with timed automata In *accepted to TCS* 2004

[BKM04] **M. Bozga, A. Kerbaa, O. Maler** Optimal Scheduling of Acyclic Branching Programs on Parallel Machines In *RTSS 2004*

[BM03] **M. Bozga, O. Maler** Timed Automata Approach for the AXXOM Case Study, Verimag, 2003

[BGM02] **Marius Bozga, Susanne Graf, L. Mounier** IF-2.0: A Validation Environment for Component-Based Real-Time Systems In *Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen* LNCS (2404) Springer Verlag June 2002

[BGOOS04] **Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, Joseph Sifakis** The IF toolset In *SFM-04:RT 4th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time* LNCS June 2004

[GS03] **Gregor Gössler, Joseph Sifakis** Component-based construction of deadlock-free systems In *proceedings of FSTTCS 2003, Mumbai, India* LNCS 2914 2003

[GS04] **Gregor Gössler, Joseph Sifakis** Priority systems In *proceedings of FMCO'03* LNCS 3188, 2004

[GH04] **Susanne Graf, Jozef Hooman** Correct Development of Embedded Systems In *European Workshop on Software Architecture: Languages, Styles, Models, Tools, and Applications (EWSA 2004), co-located with ICSE 2004, St Andrews, Scotland* LNCS 3047 Springer-Verlag May 2004

[GABB04] **J.V.Guillen Scholten, F. Arbab, F.S. de Boer, M. M. Bonsangue** Mocha-pi: an Exogenous Coordination Calculus based on Mobile Channels In *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC 2005)* ACM Press. Accepted for publication 2005

[KFB*04] **Kyas, Marcel, Fecher, Harald, de Boer, Frank S., van der Zwaag, Mark, Hooman, Jozef, Arons, Tamarah, Kugler, Hillel** Formalizing UML Models and OCL Constraints in PVS In *Workshop on Semantic Foundations of Engineering Design Languages* Electronic Notes in Computer Science Elsevier 2004

[KNY03] **Christos Kloukinas, Chaker Nakhli, Sergio Yovine** A Methodology and Tool Support for Generating Scheduled Native Code for Real-Time Java Applications In *EMSOFT 2003* LNCS vol. 2855 2003

[KY03] **Christos Kloukinas, Sergio Yovine** Synthesis of Safe, QoS Extendible, Application Specific Schedulers for Heterogeneous Real-Time Systems In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)* ISBN 0-7695-1936-9, 2003

[KW04] **H. Kugler, G. Weiss** Planning a production line with LSCs Weizmann Institute (MCS04-05) 2004

[LSCuser04] D. Harel and R. Marelly, "Play-Engine User's Guide" Play-Book

[OGO05] **Iulian Ober, Susanne Graf, Ileana Ober** Validating timed UML models by simulation and verification In *Accepted for publication in STTT, Int. Journal on Software Tools for Technology Transfer, 2004* Springer Verlag 2005

[STMW04] **Ingo Schinz, Tobe Toben, Christian Mrugalla, Bernd Westphal** The Rhapsody UML Verification Environment In *Proceedings of the 2nd International Conference on Software Engineering and Formal Methods (SEFM 2004)* IEEE September 2004

[Yov97] **S. Yovine** Kronos: A verification tool for real-time systems In *Springer International Journal of Software Tools for Technology Transfer* vol. 1 (1-2) December 1997

6 Experimental results: the OMEGA case studies

In order to evaluate the OMEGA project methods and techniques, as well as to provide feedback to their development, four case studies apply these in the industrial context. After specification of the case study UML models the OMEGA tools are being applied to these models in order to investigate applicability, usability, scalability, complementary use, and methodological aspects of the employment of these tools in the industrial embedded real-time software development. An important requirement in the Omega project was that industrial users do not only provide the case studies, but also use and evaluate the developed tools from the point of view of an industrial user. Thus, the main tool evaluation reported here has been driven by the users. After some initial help with the use of the tools, the tool providers provided mainly methodological guidelines whereas the validation activities were done by the users themselves.
In addition, all academic partners have cooperated in an evaluation experiment for one of the case studies, reported in section 6.5, with the aim to show several uses of all the developed tools in combination which included also some uses of the tools requiring a good expertise in the possibilities of the tools and verification technology, whereas the users mainly used the available push button features. Notice that in particular, the users have not evaluated all the PVS based tools as for their use, important expertise in theorem proving is needed.
Overall, the case studies played an important role for the success of the project in demonstrating well the usefulness of the profile and tools developed by the project. Presently, they play an important role for the promotion of the tools outside the project.
An overview on all case studies, providing a more detailed description can be found at the Omega web page http://www-omega.imag.fr/cs/cs.php. Also the Deliverables [M4.2] and [D4.5] contain more detailed descriptions of the experiments done by the industrial partners.

6.1 Case study 1: Ariane-5 Flight programme

The EADS ST case study presents the Flight Software of the European Ariane 5 Launcher and focuses on relevant real time behaviours. An overview on a part of the verification experiments can be found in [OGO05] and a report on earlier experiments using an SDL model of the asynchronous behaviour can be found in [BGOOS04]

The objective of this Ariane 5 Flight Software is to control the launcher mission from lift-off to payload release. This software operates in a completely automatic mode and has to handle both the external disturbances and the hardware different failures that may occur during the flight. This case study presents the most relevant points required for such an embedded application and focuses on the real time critical behaviour.

EADS ST software merges in the same processor asynchronous behaviours (stage ignition and release, failure isolation and recovery) and cyclical synchronous behaviours (control/command of the vehicle, failure detection). The validation of the asynchronous behaviour is mainly a vehicle system task, i.e., this task consists in proving that the software specification is correct with respect to the vehicle definition and that the real time constraints are coherent. The validation of the cyclic behaviour is mainly a software design task, i.e. this task consists in proving that the real time implementation of the software is correct, with respect to the multitasking architecture chosen.

In the development planning, the specification verification comes of course before the design verification. Moreover, the specification verification is easier to performed (because less complex) than the design verification.
The evaluation of the tools has followed the process used in an operational development:

- Development of the model (including environment assumption description).
- Syntax and semantics model checking.
- Model simulation on some nominal asynchronous selected scenarios.
- Model exhaustive proof on all the degraded and exotic asynchronous scenarios.
- Model simulation and proof on all the cyclical synchronous scenarios

Particularities and solutions

The main difficulty in this case study was the combination of cyclic and acyclic behaviour, which leads to an explosion of the state space (caused by the execution of several thousands of cycles along the lifetime of the acyclic behaviour which takes around 1 hour). This has called to the application of some abstraction techniques:

- Properties of the acyclic part have been initially verified by abstracting away the cyclic part manually.
- In order to verify properties which involved both the acyclic and the cyclic part, we had to artificially reduce the duration of the mission from around 1h to around 1 minute. However, the relevant behaviours of both parts are fully preserved by this abstraction.

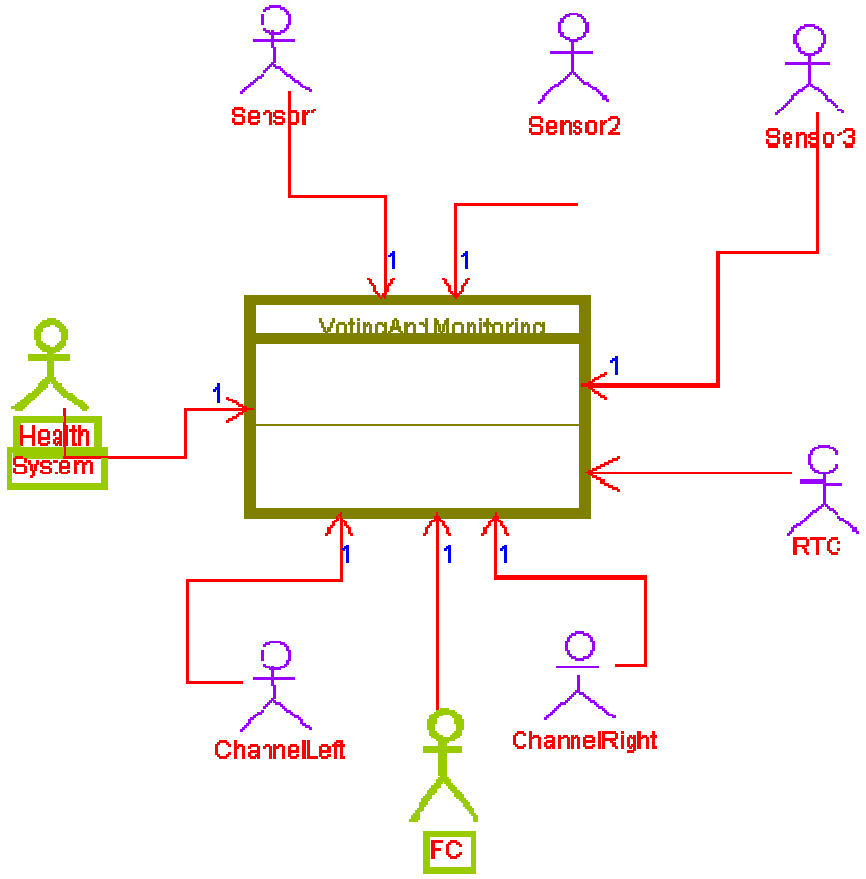
Another particular issue raised by the case study was the validation the scheduling policy used by the launcher software, which is based on a fixed-priority preemptive scheme. This has necessitated the construction of a model of the scheduler as well as the capturing of scheduling objectives by UML observers.

Evaluation summary

EADS ST has developed its UML model under the Rational Rose tool and has then used the IFx/IF tools (semantics checker, simulator, model-checker) for validating it.
The great strength of the used tools is their compatibility with the OMEGA semantics and the fact that they take into account the real time behaviour. The compiler has allowed us to obtain a model with a clean and well understood semantics, which is not possible with the Rational Rose tool (which provides neither a syntax nor a semantic checker). The simulator has powerful features (such as breakpoints, undo, redo,...) and provides a complete visibility on the model under simulation. At the end of the process, the model-checker has allowed us to verify exhaustively the properties which have been previously partially validated by using the interactive simulator.
The simulator has allowed correcting several errors in the model (mainly unexpected deadlocks) which have not been detected by manual revue. As the cost of a specification error during the validation phase is very high, these techniques have already proved their great interests.
All the properties have been exhaustively proven correct. The model-checker allows increasing our confidence in the model.

6.2 Case study 2: A Vote Monitor

The case study is a flight control mechanism that implements "sensor voting" and "sensors monitoring" operations in a typical flight control system.



The main role of a *Flight Control Computer* of an aircraft is to implement control loops based on computations of *Command* values to Servo actuators controlling the air vehicle surfaces. These computations are parameterized by the actual values provided periodically by different *sensors* installed in the air vehicle.
This system is critical and requires a very high reliability in presence of hardware faults. For achieving this reliability, we realize the avionics system using a triple *redundancy* of the different Sensors and Flight Control Computers.

The software installed in the Flight Control Computer has different modules. IAI case study focuses on one of them: *Voting And Monitoring*. The role of this module is to:

- Provide persistent sensors values resulting from the sampling of 3 sensors
- Monitor the healthiness of the sensors, based on successive comparison of sensor values
- Monitor the healthiness of the computers, based on successive comparison of the produced command values

The environment of the “VotingAndMonitoring” module is described in Figure 3.

The actors for *VotingAndMonitoring* are:

- Sensors: three (identical) elements providing specific information needed by the Flight Control loops: angles, velocities, accelerations, etc.

- FC: software module, which gets sensors values and servo-actuators status as input together with commands from the Pilot or the Controlling station. It implements control loops and outputs Commands to servo-actuators
- Channels: two other computers; exchange with present Computer the computed commands values
- Health System: software module which gather information on the good operation of the avionics system and authorizes/forbids usage of the system components especially sensors and computers
- RTC: Real-Time Clock; synchronizes the three computers and marks the beginning of a new computation cycle.

Evaluation summary

On the basis of this case study, IAI has used and done some evaluation of three tools developed in Omega. For this evaluation, the case study has been tailored:

Using Play Engine (Weizmann Institute)

The case study has been reduced and simplified for fitting the tool limitations and the case study version that was used on the LSC Play engine consists of one channel and three sensors.

Using RUVE (OFFIS)

In order to run the tool with our case study we needed to seriously simplify the model and reduce it to only 4 statecharts and 12 classes focusing in this way on the non real time issues in the model.

Using IF (Verimag)

With the IF tool, we wanted to verify time related properties of our case study. In order to do so, we have modelled the timing aspects of the system with Rational Rose, as the syntax checks of Rhapsody posed problems with the export of the action language part and also with the timed annotations. The model used for timed verification is based on the same state machines, but the functionality (in particular the voting mechanism, including the health monitor) has been omitted; on the other hand, all objects are active, and here we have taken into account two CPUs.

The model was extended with timing specifications: we defined time triggered actions and time consuming activities of variable duration.

Notice that the different models used with the three different tools could have been automatically extracted from a common global model, but presently such extraction functionalities are not (yet) implemented in the tools.

6.3 Case study 3: MARS system

The case study is the Medium Altitude Reconnaissance System (MARS), software that controls a photo camera embedded in a fighter aircraft. The camera is aimed at the ground surface and purpose of the system is to counteract the image quality degradation caused by the forward motion of the aircraft, by dynamically controlling film movement (during film exposure) and frame rate. The system controls these parameters based on the current aircraft altitude and ground speed, which are acquired from two

data streams providing *altitude* data and *navigation* data. Next to these exposure control functions, the system annotates every frame with the aircraft position (navigational co-ordinates) at the moment of the corresponding frame exposure.
The system also performs self health monitoring. It supervises the operational status of the various MARS components (e.g. camera status, serial communication status, data bus status, statuses of the hardware modules, etc.) and generates pilot alarms according to the alarm processing logic.
The experiments with the OMEGA tools have concentrated on the self-monitoring functions of the system.

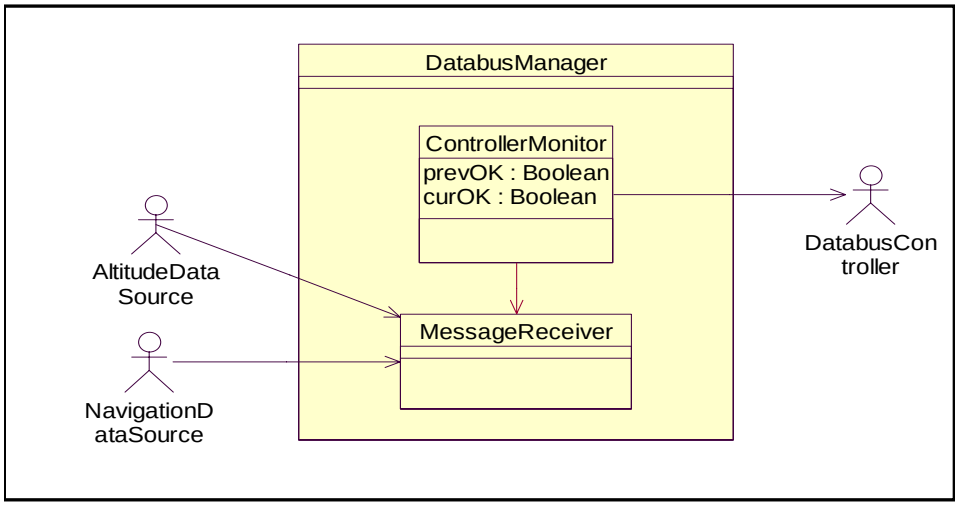


Figure 4: Databus manager

The system performs asynchronous data acquisition from the avionics data bus (altitude data and navigation data), while performing cyclical internal processing of the hardware failure detection (data bus controller Built-In-Test status). The data bus status monitoring functions involve functional as well as time-dependent system behaviour. The system environment (namely the data bus controller and data sources) exhibits non-deterministic functional and timing properties.

The case study objective with regard to the development approach was to evaluate the possibility of analysis of the different aspects of the behaviour and properties of the real-time software system model through application of several formal verification techniques to the common model.
The case study objective with regard to modelling was to evaluate the efficient way of high-level requirements modelling as well as use of the OMEGA UML for modelling of non-deterministic and timing aspects. For the high-level requirements modelling the use of LSCs was proposed. The case study evaluated the use of LSCs for the scenario-based modelling and perspective of further use of the LSC models in the UML-based development.

The verification and validation activities comprised application of the OMEGA tools to the case study model in the series of verification experiments. The objective of these experiments was to evaluate the following aspects of the application of the new OMEGA technologies in the industrial context:

- relevance to the case study domain;
- applicability to the issues of the industrial software development;

- usability in the industrial setting;
- OMEGA tools in the industrial software development lifecycle.

Three tools were employed in the scope of the case study. The LSC PlayEngine tool (Weizmann Institute of Science) was used for the high-level scenario-based requirements modelling and verification of the requirements model. The Rhapsody UML Verification Environment (RUVE) tool (OFFIS) was used for untimed verification of the case study UML model. The IF/IFx tool (VERIMAG) was used for the timed verification of the timed version of the case study UML model, which made use of the OMEGA UML time extensions.

Particularities and solutions

The most important issue of this case study was related to verification of the model with non-deterministic environment. The main source of non-determinism lied in the fact that the data sources were independent, unsynchronised, provided cyclical data with non-deterministic timing jitter (bounded to ±10 ms), and had a possibility of non-deterministic data loss. For the purpose of untimed verification the timing aspects were abstracted to purely functional behaviour with the use of system execution step as a relative discrete time measure, while data losses were made observable through explicitly modelling them as non-deterministic "no data" messages. For the purpose of timed verification explicit timed model with non-deterministic timed environment was specified. Abstraction techniques were used to allow verification of separate timed properties as enabling all the possible sources of non-determinism in the model led to the state space explosion.

Evaluation summary

The UVE tool

An untimed version of the case study UML model has been developed in the Rhapsody tool and verified with the UVE tool. The tool provides possibility to specify the non-deterministic external stimuli to drive the system model behaviour. The tool allows verification of safety and liveness properties of a UML model, and provides facilities to specify assumptions on the model behaviour as well as on the environment (external stimuli).

Several untimed properties, specified with the use of propositional logic expressions, temporal logic patterns and LSCs, have been verified. For specification of complex properties and assumptions combinations of several temporal logic patterns can be employed (if LSC specifications are used the number of LSCs is in principle unlimited). While the verification experiments could effectively use such specifications for several properties, the limits of the reasonably practical tool use were reached on several occasions.

During the high-level design phase design decisions impact all the subsequent development phases in terms of safety and liveness properties, required sequencing of internal and externally visible responses to the external environment stimuli. The RUVE tool proved to be reasonably effective for the verification of high-level models, or partial models of the critical system parts.

The IF/IFx tool:

A timed version of the UML model has been developed and verified with the IF/IFx tool. The tool allows a more realistic modelling of time dependent behaviour in the self-monitoring components, as well as a more explicit environment modelling. The

latter is possible as the tool provides support for non-deterministic behaviour in a closed UML model, including timing non-determinism.
Several timing properties have been specified and verified using observers. For example, a typical property is the following: "*If the DatabusController becomes non-operational at time T and stays non-operational for more than 10 ms then the MessageReceiver shall enter the state ControllerError by the time T+10 ms at the latest*."
Observers proved to be a convenient and intuitive way to express properties. For modelling complex properties, the use of parallel composition of several observers based on shared attributes is possible and convenient.
Verification experiments were performed in several configurations: synchronised vs. desynchronised data sources, deterministic vs. non-deterministic polling performed by the ControllerMonitor, etc. We have thus been able to verify the model in realistic conditions, but also to reach the limits of the verification tool (e.g. for desynchronized data sources and deterministic polling).
During the high-level design phase design decisions are very sensitive with respect to the system timing issues. The IF tool provided effective means for timed verification of the abstracted simplified models, dedicated to the respective timed properties.

6.4 Case study 4: A service component based depannage system

The FTR&D application is a telecommunication service built on top of embedded platform and service components. The complete application developed for Omega is a service called *Depannage*. The Depannage service is related to a specific user need (the subscriber): Medical and doctor, Fire brigade, car repairing, etc. It allows a user to call for a *Depannage* with a specific number. The service invocation firstly asks for authentication of the calling user, and then will search the calling location. Once the calling location found, the service will search in a data base different numbers corresponding to a provider of the requested *Depannage* as close as possible from the location of the calling user. Then, the service will try to connect the calling user to one of the numbers (in a sequential or parallel way). In no case, the calling user should be connected to a secretariat or to a vocal box. The platform and service components should be reusable for different service logics and therefore they are specified independently of any embedding system. The communication and all these components include time constraints.
FTR&D used during the project a set of techniques in order to build the application by a step-by-step approach.

- First, we describe a high level specification of the service and component behaviour, including the behaviour of the communication between these components. This description includes timed constraints. Then the consistency of this high level specification is validated with respect to end-to-end requirements. This analysis is made with LSC, the Play Engine tool and simulation/animation.
- In a second step, model checking techniques are used with the Play Engine tool in order to verify in a formal way some requirements. The Play Engine tool allows verifying model with timed constraints, but it implies restrictions on the model. Then, parts of the model are identified, focusing on complex and/or critical behaviours.
- Once these first steps done, a more complete model (with all the potential behaviours, including creation and destruction of objects) is elaborated using the

Rose CASE tool. This model is then exported to the IF/IFx tool and it is validated with respect to some requirements expressed by observers.

Evaluation Summary

Application of LSC and animation with the Play-Out Engine: The wish to specify components in a reusable way involves that the component specification should be done independently of any embedding architecture. Such specification should correspond in universal LSC describing how the component will react to events coming from its provided ports and how (and when) this component will act on its required ports. For the system, the complete application, the specification should be enhanced by universal LSCs describing the communications between these components. Such LSCs could include time constraints and delays on the communication. The end-to-end requirements are expressed by existential LSC and will be validated during the simulation/animation of the model. The Figure 6-1 represents the communication between two components.

Application of the Play Engine model checking to timed verification: The model checking tool allows formally verifying the expressed requirements. In order to use the model checking tool, some restrictions need to be made on the model: no symbolic instances, only one parameter for each signal. We have also to restrict verification on parts of the model in order to avoid state-explosion (explosion of the graph which is enforced with time constraints). It means that we have to focus our work on the more critical part of the behaviour. We want to verify that all possible executions of the model satisfy the specified requirements. The smart Play-Out approach allows executing all the execution paths and, during this execution, one searches for the satisfaction of a property (an existential LSC). Using the smart Play-Out tool we have to express a property violating the requirements, thus, the model is correct when the property is not satisfied by the model. The kinds of requirements we want to verify are: $d1 \leq Time\_Duration \leq d2$, were *Time_Duration* is the end-to-end execution-time of some service or sub-service.

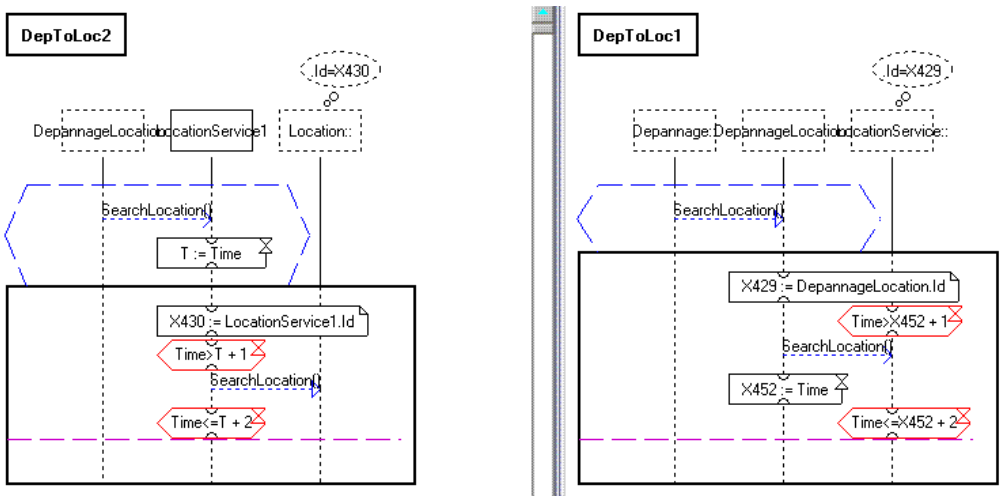


Figure 6-1: Connectors with Time Constraints

The current version of the Play Engine tool, including time constraints, gives good results. However, the tool should be improved for more practical use.

Application of the IF/IFx tool for formal specification and verification of complete systems. The objectives of our work with the IF/IFx tool was to make a formal verification of a more complete model. The model developed here includes more complex behaviours corresponding to the call termination. It involves the creation and destruction of objects and more complex message exchange patterns. In this model, we introduce also another behaviour that has not been completely described with the LSC tool: the fact that several calls can be initiated in parallel in order to search the called party. In this case, only the first party answering the call will be connected to the calling party, all other initiated calls will be killed. We expressed some properties expressing the correctness of the model with respect to the service requirements by means of observers:

- If a call succeeds, all the other initiated calls are aborted.
- A call never succeeds to call the vocal box of a mobile phone.

We were very satisfied by the use of the OMEGA/IF approach for the modelling and verification of our application. The main characteristics of telecommunication models and their properties could easily be modelled: non determinism, different kinds of message exchanges, time constraints and timers, etc. The verification techniques are complete and efficient: simulation, exhaustive simulation and observer verification. Also the user interface of the IF/IFx tool has good functionalities: it really allowed understanding the behaviour of our system.

6.5 Case study 5: Compositional verification of the MARS case study

Finally, all tool providers have applied advanced verification techniques on the MARS case study of section 6.3. The aim of this work was twofold. On one hand it was meant to demonstrate some of the possibilities of the tools which have not been exploited by the users – in particular the tools based on the use of the PVS theorem prover and the different compositional verification methods. On the other hand, it should demonstrate a development methodology taking into account the need of validation which moreover demonstrates a combined use of all the different OMEGA tools. Some more details can be found in the set of documents constituting deliverable D3.3, in particular the documents [D3.3-I, D3.3-A1, D3.3-A2, D3.3-A42].

Here, we have considered mainly the bus manager module which has initially been given as a single module and we have decomposed it in different ways so as to be able to apply compositional verification and verification based on abstraction.

In this work we have concentrated on the *MessageReceiver* module which is the main source of space explosion and a good candidate for further decomposition. The initial version of the *MessageReceiver* defined by NLR is described by a single state machine handling all incoming messages and changing status according the messages received during the last three periods. Also, the descriptions for timed and untimed verification were different.

We have experimented different decompositions:

- In all cases, we have introduced a “vertical” decomposition, leading to a separate *Receiver* for each data source and an *ErrorLogic* module taking the decision about the status change.
- In some cases, we have added a “horizontal” decomposition by further decomposing each Receiver into modules for each of the three basic functionalities of a receiver: detecting a valid period, detecting data / absence of date and counting the number of consecutive *data*/*nodata* periods.

In different experiments, we have used different decompositions.

The horizontal decomposition into (1) period detection (2) data / data-miss detection and (3) message counter was used for compositional verification with the untimed model-checker UVE. In this case the part (1) is always considered part of the environment, and different hypotheses are used about the sequences data and data-miss messages sent by the environment are used for verifying properties of the other modules. The possibilities of the UVE tool do assumption commitment style reasoning, by deriving properties by taking properties of input sequences as assumptions, has been exploited here. Due to the restrictions of the Rhapsody tool – which does not allow to specify an open system – this required quite some remodelling of the communication model in order to keep the overhead small for adapting the model to different local verifications.
The limitation of this untimed verification was mainly that it was hard to find untimed properties which do not depend on assumptions on the environment, whereas in principle, the system should work also in an unrestricted environment. In addition, the union of the used restricted environments, in the form of regular properties, does not cover the set of all reasonable environments.

Using the IF tool, we did not need any further horizontal decomposition as the main source of state explosion was the number of non synchronized clocks. The verification of a system with a single data source requires in all cases 2 such clocks and this verification could be carried out without further reduction[6]. Each receiver handles all issues mentioned above and sends in every period (at the point of time of data or data-miss detection) the status of the last three periods to the *ErrorLogic*, which changes the status depending on the information received from **all** *Receivers*.
The vertical decomposition has been used for compositional verification by abstracting the behaviour of the other (or the set of other) receiver(s) to the noise that it represents for the non abstracted part. Here, we could use a chaotic abstraction and still show the desired properties concerning the concrete *Receiver*.

The verification using PVS is similar to the one using the UVE tool, but extended to the timed case. Also the set of properties that can be used to express assumptions and requirements is much richer.

Especially when considering the timed models, some of the decompositions and optimizations, we came up with, turned out to be erroneous. These errors were always easily detected using the explorative model-checker of IF. In some cases, we did not need any further experiments to understand and fix the error, but in some cases, the model checker turned also out to be a good tool to understand some errors by experimenting with versions of the model where the time constraints in the model or in the requirements were slightly modified.

6.6 References concerning the Case studies

[BGOOS04] **Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, Joseph Sifakis** The IF toolset In *SFM-04:RT 4th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time* LNCS June 2004

[6] The size of the model is around 100 000 states which is unproblematic

[OGO05] **Iulian Ober, Susanne Graf, Ileana Ober** Validating timed UML models by simulation and verification In *Accepted for publication in STTT, Int. Journal on Software Tools for Technology Transfer,* Springer Verlag, 2005

[D3.3-I] **Jozef Hooman**, D3.3: General Methodology, Jan 2005

[D3.3-A1] **Angelika Votintseva**, D3.3, Annex 1: General methodology for untimed verification, Jan. 2005

[D3.3-A2] **Iulian Ober**, D.3.3 Annex 2: Specification and verification of real-time systems using the Omega real-time profile and the IFx verification tool, Jan. 2005

[D3.3-A42] **Jozef Hooman and Marcel Kyas**, D33 Annex42: Compositional Verification of Timed Components using PVS, Jan. 2005

[M4.2] **Pierre Combes, David Lesens, Yuri Yushtein, Meir Zenou**: Preliminary evaluation of the Case studies with the OMEGA tool set and future validation plans, June 2004

[D4.5] **Pierre Combes, David Lesens, Yuri Yushtein, Meir Zenou**: Final evaluation of the OMEGA tool set, Jan. 2005

7 Summary of results and achievements

The OMEGA project, which started in January 2002 and completed in February 2005 has achieved the following results:

- Defined a ***UML profile for real time***, adapted for a wide class of real-time and embedded systems and usable with the several main UML case tools used in this domain. The profile covers a large subset of UML and has been extended with missing features; it includes means for the modelling structure, behaviour and requirements with time using operational, declarative and mixed descriptions
- Adopted a ***common format for model representation*** based on the UML XMI standard exported by CASE tools, defining a common syntax for the action language and OCL and a particular XML format for the representation of LSC.
- Implemented a set of ***validation tools*** for models adopting the Omega profile and the Omega model exchange format. Different tools address different aspects of models; they cover requirements and design and architecture analysis, verification of functional, coordination and timing properties. The developed tools are either freely available for research activities or will made commercially available in the future
- Developed ***verification methods*** adapted to the expressivity of the defined profile combining different communication paradigms, object orientation, functionality and time.
- Carried out a series of ***industrial case studies*** for validating the developed profile, methods and tools.
- Contributed to the state-of-the art by developing several ***theories*** related to the problematic of modelling and validation of real-time embedded systems.
- Published over 60 ***papers*** in conference proceeding and journals. In addition, a special issue in the SoSym journal is planned which will present some of the theoretical results of that project and also an overview on all the results of Omega and presentations of the most interesting Omega case studies.

- Initiated and/or organised 13 international ***conferences and workshops***, mostly with published proceedings, in particular the new series of symposia FMCO, which has been organised 3 times, and the SVERTS workshop which has been organised twice as a satellite event of the UML conference. We intend to continue both series of events created as Omega initiatives, in slightly modified settings.

The ***language and tool developments*** have been done jointly by all the academic partners based on input and feedback from the users. Below, we provide for each result the contributors:

- The main contributors to the Omega Kernel model and its semantics are OFFIS and the Weizmann Institute. The final version has been obtained due to numerous contributions by all academic partners.
- The main contributor of the Real time profile is Verimag taking into account suggestions made by other partners.
- The main contributor for the definition of the Omega OCL profile is the university of Kiel in collaboration with CWI and taking into account suggestions by other academic partners
- The adaptation of LSC for the UML framework including the adaptation of the real-time profile has been done by the Weizmann Institute.
- The Omega component model has been developed by CWI based on numerous discussions amongst academic partners.
- The UVE tool for untimed verification, including the corresponding XMI import, has been developed by OFFIS as an extension of their pre-existing verification tool RUVE for the verification of Rhapsody models.
- The IFx tool for the verification timing properties, including the corresponding XMI import, has been developed by Verimag as a front-end of their pre-existing IF verification engine
- The LSC play Engine, including the corresponding XMI import, has been developed by the Weizmann Institute based on a pre-existing version of that tool.
- The package of tools enabling the verification by means of properties expressed in OCL and a subset of the operational profile have been developed jointly by the university of Kiel, the university of Nijmegen and CWI.
- The tools for manipulating and transforming XML documents have been developed by CWI.

The ***experiments*** were coordinated by the industrial partners, *EADS ST*, *NLR*[7], *Israeli Aircraft Industries* and *France Telecom R&D*. Each partner providing a case study used a subset of the tools in collaboration with the academic partners providing them. On one case study, all academic partners collaborated on an effort showing how the tools can be used in combination and how to provide a model more suited for compositional verification.

The ***academic results*** of the project as well as the proceedings of organised events have been published. The list of publications, providing also the information on the involved

[7] During the first two years of the project; then, the NLR expert has continued his work as an employee of the university of Kiel

contributors, is provided as an Annex A of this document. All papers are available at the Omega web page.

8 Lessons learned

The goal of the project was to allow a tighter integration of formal validation in a model based software development process of real-time and embedded systems. We have chosen UML and existing UML based CASE tools as the framework for providing such an integration.

The industrial experiments show that the outcome of the project is very positive, both, concerning the developed UML profile and the developed tools, encouraging us to continue the work started in Omega. Nevertheless, the case studies, as well as the difficulties encountered during the project, show also that there is still a long way to go for the existing tools to provide a model based development framework fully integrating validation techniques at all development steps. A lot remains to be done for our tools which lack still many of desirable and necessary features, but also UML or in general model based development in its generality, is still an emerging and heavily evolving, and not a mature technology.

Some things have changed during the duration of the project, and not always as we expected. We anticipated UML 2.0, and in fact, the finally adopted version corresponded quite well to our expectations, but it took more time than anticipated, and there exist still no “real” UML 2.0 tools today.

Concerning tool integration, we did not consider the integration of our tools into a particular CASE tool or development framework, as we aimed to be open for different frameworks, which motivated also the choice of a kernel UML accepted by several tools. We still think that this choice was the right one, even if it has some drawbacks, in particular the one that it cannot be used smoothly without further adaptation with any existing UML tool providing simulation facilities. The UVE tool by OFFIS provides a version for Rhapsody users, by nevertheless abandoning some of the features of the profile.

UML profile and semantics

The concepts covered by the profile turn out to be a good choice. We have privileged extensions of notations already familiar to the users: in the operational kernel model, the concepts covered are those chosen in the Rhapsody profile and existing in the same or a similar form also in other tools (such as Rose Real-time or Telelogic TAU tool) and formalisms (such as SDL). Similarly, for the interpretation of the concepts, we have chosen a *usual interpretation* or a more non deterministic one, more in line with a modelling language. Especially the users familiar with the Rhapsody tool did not always easily accept any changes with respect to the Rhapsody profile, even in cases where the Omega solutions had some objective advantages. Our experience should be taken into account by other projects, proposing new solutions and aiming for user acceptance.
Indeed, we had initially much more innovation in mind with respect to the concepts covered by UML 1.4, especially for the notations going beyond the operational kernel model. The main blocking factor were the concepts handled by the editors of the

existing UML tools as our aim was not to work on UML graphical editors, but to build on the existing ones.
For the ***extension of sequence diagrams*** in the form of LSC, we could rely on editor, already under development at Weizmann Institute which allowed the introduction of interesting concepts not covered by UML sequence diagrams. The existence of this editing tool together with a tool implementing several analysis techniques contributed to a good user acceptance, together with expressivity and intuitivity of the proposed formalism.
Concerning ***architecture and component related concepts***, we proposed the use of stereotyped class diagrams and additional OCL constraints. Unfortunately, it turned out that this kind of extensions were not really accepted by the users, partly because they preferred graphical notations, and partly because the only tool supporting these concepts required the use of the interactive theorem prover PVS.
With ***timing extensions*** we made a similar experience: it turned out that the best option was to propose a small set of basic constructs (in our case timers clocks and semantic level events) and integrating them into (extensions of) the existing notations rather than defining a small OCL like language for the expression of dynamic timing properties. The concepts provided in the form of tag values in the SPT profile, can then be obtained as derived concepts, and could even be user defined. This, and the existing tool support for simulation and model-checking lead to a good user acceptance.

An aim of the project was also to define an *unambiguous semantics* implemented by all tools. From earlier experience, we knew from the beginning of the project that this constitutes a real challenge for such a rich set of notations with many semantic variation points, which should be handled consistently by several tools. We succeeded in partially handling the issue by the fact that many, but the most central concepts were handled by a single tool. Nevertheless, handling semantic issues represented much more effort than initially anticipated, and yet we have not fully achieved the initial goal. A detailed discussion of the difficulties encountered and approaches followed in the project, which might be useful for future projects, can be found in section 4.2.
What became clear, is that the so called model based development approach, based on the existence of a meta-model, supports syntactic transformations, but does not presently provide any help for alleviating our problems with dynamic semantics; it rather handles this problem by just ignoring it.

Building tools for UML

Concerning the ***XMI exchange format***, the experience we had shows both positive and negative aspects.
On the positive side, XMI has, albeit with a lot of effort, allowed us in the end to use UML models edited with real industrial tools. If flexible enough, a tool is able to encompass the differences between versions of XMI generated by different tools (XMI 1.0 for Rhapsody, XMI 1.1 for Rational Rose).
On the negative side, the XMI format is very complex and contains a lot of useless information. Moreover, even for a same XML schema, interpretation about where (i.e. in what XML element) to put a certain information may vary from one tool to another. In the project, these negative aspects have triggered the work on a simplified version of XMI (SUML).

For these reasons, we think that using XMI as an exchange format was the right thing to do but there is a real need in the UML community for developing re-usable XMI parsers in order to share the big development effort which is implied.

Tool evaluation

We consider that the applied tool evaluation strategy, which is not a completely standard one, had many very positive effects. The fact that the users did not just bring the case study, but did the main work on them by using the tools developed in the project, gives a much stronger value to the final tool evaluation than in the usual situation, where the tool developers themselves demonstrate the potentialities of their tools. This was also much more profitable for the users as they got a much deeper personal involvement with the tools. Obviously, this lead to a much more critical evaluation, because this did not allow to hide all the details still to be worked out; it also showed clearly which kind of tools have a chance to be used in industry – in the context aimed by the all the users – and which kind of tools are limited to a much more specialized user community, which may also include specialists from industry. We believe that this procedure had also the very positive effect of increasing the chances of future collaborations and usages of the tools, and this despite the fact that the tools became fully available, relatively late in the project.
Finally, the relatively tight interactions between users and tool providers, was an important factor for the building of an "OMEGA team".

8.1 What would we do the same? What different?

If we had to do it again, with all the experience gained and taking into account the evolution outside the project into account, what would we repeat, what would we do differently? We have tried to summarize the main issues in the following table.

What would we repeat?	***What would we do differently?***
Modelling language: we based our approach on UML and defined a rich profile for real-time embedded systems compatible with the relevant CASE tools. This was certainly a wise decision in terms of industry acceptance.	**Modelling language**: obviously, starting today, we would chose UML 2.0 instead of UML 1.4 which would allow us also to conveniently treat aspects left aside in Omega.
UML profile: contrary to most approaches for verification in the context of UML, we have chosen a rich profile, including most of the concepts that the users expect in the modelling language used in the system development process. This decision made our live much harder, but it was the decision to be taken for being accepted by the users. The *Timing extensions* and the adaptation of *LSC* for requirements modelling are great achievements of the	**UML profile**: we have not taken into account all important user requirements, in particular those concerning the modelling of architecture constraints and the possibility to model designs with synchronous parts directly. Both, the evolutions of UML and the fundamental research we made during the project, would help us to address these issues in a more straight forward manner, when starting today.

project.	
Integration with CASE tools: we have made the choice to develop a framework and tools independently of a particular framework as such important tool development should be sharable by several CASE tools for a given profile. We think that this was a good choice as it gave as more freedom when defining the profile. Even if the general approach was to be compatible with the main tools, we could deviate where this seemed preferable to us.	**Integration with CASE tools**: the independence of the CASE tools has an obvious reverse side. Our verification results are not compatible with the simulators provided by the case tools (where they exist). Adapting our tools and profile to particular tools, is work which should start now in collaboration with the CASE tool providers. More generally, the industrialisation of the Omega tools remains a major issue to be improved on.
Tool integration: we have built a set of validation tools based on a light-weight integration through sharing the same UML profile and formats for models and properties. We still think that this is a sound approach as it allows different tools to evolve or even die independently without breaking the rest. But on the other hand, from a user point of view an integration in a commercial CASE-tool is required to be acceptable for industrial usage. The RUVE tool has demonstrated this approach.	**Tool integration**: we have not explored all possibilities to share model representations. In addition to the UML/LSC level models which can be shared today, it would be interesting to have an intermediate level format representing communicating automata allowing to handle state machines, LSC, temporal properties, and why not also activity diagrams, in a more uniform manner. In future projects, we plan to develop a semantic level tool exchange format.
Approach to methodology: we have identified the general activities during development, a basic development process, and investigated the possibilities to support this process by formal methods. Moreover, discussions about the methodology have led to pictures about the tool set and relations between tools, to an investigation of the problems addressed, a vision on the Omega solutions, and finally, scenarios of possible uses of the tools. To get more insight in the relation and combination of tools, we have defined a common case study. In general, these activities have contributed to the global coherence of the project and led to interesting insights.	**Approach to methodology:** in the early phases of the project most partners were quite busy with semantics, tool support and modelling case studies, so there was not so much attention to methodology. Looking back, it was also not so clear that the work on methodology could play an important role in clarifying the overall vision of the project and the relations between the tools. Hence, in a future project, the activities should be scheduled earlier. In particular, it would be beneficial to define at least a small common case study much earlier.
Approach to tool evaluation: The fact that the main work on the case studies and the tool evaluation has been done by	

the users – in frequent interaction with the tool providers – was extremely positive and could serve as an example for future projects	

9 Plans for the future

This section describes both direct exploitation plans as well as some future research directions, taking their root in work done in OMEGA. Some of these plans have already started to be realized, whereas others are still in a more preliminary phase[8].
The future plans are divided into three categories, plans with respect to the developed profile and semantics, plans with respect to the developed methods and tools and "other plans" including those of more general nature as well as plans aiming to cover a larger part of the development process.

9.1 Profile and semantics

The parts of the profile that turned out to be most useful will be maintained in the tools, further developed. Collaborations with tool providers are planned trying to push some of the ideas into existing case tools, and some activities will take place to influence the standard[9]. Notice that the principal exploitation of the profile is by the usage in tools. A profile is alive and useful mainly in combination with the tools that are exploiting it. All parts of the profile (the operational part, time extensions, observers, LSC and OCL) are already and will continue to be disseminated in publications, on websites and also in future collaborations of the partners. In the first part of this section, we mention precise relatively short term exploitation plans of the partners.

- LSC have already influenced the standard UML 2.0
- Some of the proposals in the time extension have influenced the *Call for Request* for the next version of the real-time profile. We intend to push Omega time extensions (mainly the event matching mechanisms) and the notion of observers in the next version by joining a consortium in the context of the ARTIST NoE.
- In the context of the French PERSIFORM project involving the Omega partners France Telecom and Verimag, we will extend the profile for including activity diagrams both as operational specifications and observers.
- In the framework of Artist and forthcoming national projects, Verimag will participate in the responses to some of the current RfPs of OMG, one concerning a profile for real-time and embedded systems and one concerning a semantic profile.

We anticipate that there will probably never be an unambiguous semantic prescription for UML defined by the standard, simply because the different involved parties will not be able to agree on such a unique interpretation. UML 2.0 has more semantic prescriptions than earlier versions, but not for all concepts, and moreover, the variety of

[8] Notice that, depending on their policies, some partners did want to reveal all of their exploitation plans in this report which is a public document.
[9] Notice that an academic partner alone has almost no chance to obtain anything at OMG and moreover this extremely time consuming activity is barely compatible with the obligations of academics. So the reasonable solution is to join a consortium, which we have done in the past and will do in the future.

profiles adopted by different tools is not likely to disappear. There is even some tendency towards domain and application specific languages.

- A first consequence is that tools must be open to several semantic choices for a given set of concepts. In the project, we have identified a number of interesting semantic variation points and their variations. It would be interesting to deepen this issue in order get the right flexibility for the kind of analysis provided by each tool.
- A second consequence for tools is that either, they must be open to new concepts – which may be easy in some cases but lead to important changes in others – or one has to better take into account the fact that tools may not handle a profile in its totality. We have already done this in Omega, but mainly by big categories. In the future, more effort should be made in handling properly the concepts of a profile not covered by a tool, and also in identifying the constraints that have to be imposed on the usage of the profile so that the analysis on the abstracted model carries over to the concrete model.

The scientific results obtained concerning coordination and scheduling frameworks, as well as the early prototype tools developed in the project, will be further developed and applied to practical case studies. One open issue is here to provide a more flexible integration between the synchronous and the asynchronous parts of the system, which is an explicit request by the users. One of the challenges is to get the abstraction level right, depending on the properties under verification.
We have concrete plans for exploiting these results for defining a small *semantic kernel*, for UML and other modelling formalisms. This will be done in a future project, in which we aim at an even larger integration of design formalisms and tools, on a larger spectrum of non functional features and on more lightweight, scalable analysis. Three Omega partners, IAI, OFFIS and Verimag will be partners of this project.

The experience obtained in this project demonstrated that a language like OCL will only be accepted if the language is properly supported in case tools. While OCL constraints are not as appealing as notations like LSC its declarative nature has many advantages. The group from Kiel will research trace based specification languages for object-oriented systems, based on OCL like notations, and its combination with graphical notations, similar to UML 2.0 protocol state machines. The remaining issues are how one has to deal with the dynamic evolution of object structures in such notations, especially with object creation, in a compositional setting.

9.2 Methods and Tools

All the tools developed in OMEGA are available, and can be used. For some of the tools plans for commercialization or integration in a commercial tool chains are being put up. Several projects have started or are in a planning phase where Omega tools will play an important role.

- The Omega webpage http://www-omega.imag.fr/tools.php/ provides an overview on all tools and case study results as well as links to the webpages of the individual tools which are maintained by the developer of each tool.
- There are advanced plans for providing in collaboration with I-logix and other Omega partners, the three main tools developed in the project, that is the UVE tool, the Play-engine and the IF verification tool as Rhapsody plug-ins.

The following plans concern uses and further developments of individual tools.

Play Engine

- The Play-Engine and research done in OMEGA will be applied in the future to research on biological modelling, a joint research effort of the Weizmann Institute with scientific collaborators at Yale and NYU.
- The Play-Engine is currently been used for educational purposes in university courses in the Weizmann Institute, New York University and other universities. It will also be a basis for research efforts at the Weizmann Institute and applications in cooperation with industrial partners.

UVE tool

- The UVE tool will be used and improved in other national and international research projects. It will be used for the verification of case studies and for the evaluation of model-checking methods over UML models. Among others, the following already running projects will use UVE: ARTIST2, OPRAIL, AVACS, (possibly EASIS).
- In the national project OPRAIL the UML profile is reconsidered to identify a UML subset which can be used in safety critical rail systems applications and which is in compliance with the CENELEC requirements regarding SIL3 and SIL4 applications. Based on that profile an adaptation of the UVE tool will be provided.
- OFFIS' aim is to put its tools to the market. As a non-profit organisation this cannot be done by OFFIS itself. Due to that, a spin-off company has been funded some years ago which has already commercialized some prototype tools developed by OFFIS within EU-funded projects.
  Regarding the RUVE tool, we think that some additional improvements are necessary before such a step will be possible. Nevertheless, a productization depends also on the market expectations and the willing of a possible tool vendor to put the tool on the market. OFFIS is permanently in contact with possible vendors and hence OFFIS is confident that a commercialization will be achieved if an acceptable maturity level of the tool is reached.

IF/IFx tool

- A common exploitation of the IFx and IF tools between Verimag and France Telecom is planned in the recently started French RNRT project PERSIFORM, where connections with commercial performance evaluation tools, mainly SES workbench, extensions of the profile to cover activity diagrams and another view on the passage between the service point of view and the component point of view are planned.
- A common exploitation of the IFx tool between Verimag and EADS is planned in the context of the recently started ASSERT IP. The Omega profile could be used for an initial modelling of the EADS case study – probably from the context of ATV – a AADL/UML profile is planned where we plan to have the UML part as close ads possible to the Omega profile. In this context, we will use IFx also in order to verify product lines obtained from UML component depositories using inheritance.

- At a relatively short term, we plan an integration of the timing extensions and the IF validation facilities as a plug-in into Rhapsody. This integration is planned as a common effort between IAI, I-Logix and Verimag.
- An adaptation of IFx to a UML profile for SoC will be used in a project initiated by the French Rhone-Alpes region, for the verification of the asynchronous parts (protocol layer) of a Network on Chip architecture.

OCL and PVS Based Tools

- We plan to extend the OCL and PVS based tools in different directions. This includes moving the input language accepted to UML 2.0, supporting larger subsets of these languages, like hierarchical state machines, and improve the assertion language to cover common idioms in a more convenient manner.
- The modular design of the OCL and PVS based tool allows to include more analysis phases and translations to other tools. CAU is interested in supporting provers which allow a higher degree of automation, aiming for a subset of constraints which can be checked automatically. This involves a combination of techniques, including static analysis techniques and model checkers.
- The OCL tools have been made available under the terms of the General Public License (GPL). Kiel plans to move the code base to a public CVS repository and invite interested parties to participate in the development of the tools.

General

Since the termination of the project, UML tools have been integrated in Eclipse[10], which makes now this environment very attractive for the OMEGA tools. Unfortunately, this was not yet the case during the project.

Notice however that all these tools are based on UML 2.0, which means porting Omega tools into this requirement requires some adaptation effort. Nevertheless, the existence of UML 2.0 environments opens also several interesting perspectives; in particular, it will allow us to better take into account architecture and components.

We plan to port several Omega tools into this environment and are currently preparing several project proposals for this. We have already started to provide a meta-model for the IF language compatible with Eclipse; this should allow us to ease future mappings from UML to IF by using rule-based transformation languages as they have been developed in or outside OMEGA.

9.3 Other plans`

- The OMEGA webpage will be maintained and improved during a period of at least 12 months and it will then remain available for at least another 5 years or until obsolete.
- The Omega initiated workshop SVERTS and the Omega initiated symposium series FMCO will be continued on an annual basis
- Several partners, in particular the groups from OFFIS and University of Nijmegen will incorporate the knowledge obtained and the tools developed into their courses on embedded systems

[10] In particular the commercial tool RSA from Rhapsody as well as some opensource tools, like Omondo and others currently under development in different projects

- The users, in particular EADS and IAI have already started to promote the Omega results internally, where the case studies play an important role.
- IAI will perform a follow-up of the progress in the maturity of some Omega tools (IF, UVE and Play-Engine).
- EADS foresees further presentations of the tools (UML and IFx), but for take-up of results commercial tools are a condition
- The partners from University of Nijmegen intend to reuse the work and experience from Omega in a project at the Embedded Systems Institute which aims at extending and applying the work on the coupling between UML-based CASE tools and Mathlab/Simulink
- Two partners consider the combination of existing automatic test case generation methods with the work done in Omega:
    - University of Nijmegen will investigate with Jan Tretmans at Nijmegen and ASML, the possibilities to start a project on test case generation from UML models, based on the Omega semantics.
    - Verimag envisages a more user friendly integration with UML and IFx of the TGV tool for test case generation, which can already today be used as a backend of the IF tool.
- CWI will continue working on compositional theories for UML models and on high-level object-oriented scripting languages for the simulation of UML models.
- University of Kiel has started to implement an SOS based semantics of UML2.0 statemachines in the rewrite engine Maude allowing already handling a large subset. This effort will be continued. Furthermore, in a continuation of the Dutch-German bilateral research project Mobi-J, it will extend its results on the semantics, specification, and verification of concurrent Java programs to cover more features of Java, and extend these results to a wider range of object-oriented languages.

10 Annex A: List of project publications

We provide here all papers based on Omega results published so far. We expect some more publications concerning the work done in the last period of the project, concerning mainly tools and case studies.

2005

Omega Publications

1. **Werner Damm, Bernhard Josko, Amir Pnueli, Angelika Votintseva** A discrete-time UML semantics for concurrency and communication in safety-critical applications In *Science of Computer Programming* 2005
2. **Susanne Graf, Ileana Ober, Iulian Ober** Timed annotations in UML *accepted to STTT, Int. Journal on Software Tools for Technology Transfer* Springer Verl. 2005
3. **Iulian Ober, Susanne Graf, Ileana Ober** Validating timed UML models by simulation and verification In *Accepted for publication in STTT, Int. Journal on Software Tools for Technology Transfer, 2004* Springer Verlag 2005
4. **Zanconi, Marcelo, Yovine, Sergio** Modeling and Analysis of Real Time Systems with Preemption, Uncertainty, and Dependency Verimag (TR-2005-1) January 2005
5. **D. Harel, H. Kugler, A. Pnueli** Synthesis Revisited: Generating Statechart Models from Scenarios-Based Requirements In *Formal Methods in Software and System Modeling,* Lect. Notes in Comp. Sci. vol. 3393 Springer-Verlag 2005
6. **D. Harel, H. Kugler, G. Weiss** Some Methodological Observations Resulting from Experience Using LSCs and the Play-In/Play-Out Approach In *Proc. Scenarios: Models, Algorithms and Tools* Lect. Notes in Comp. Sci. vol. Springer-Verlag 2005
7. **Jozef Hooman and Mark van der Zwaag**. A Semantics of Communicating Reactive Objects with Timing, Accepted for STTT, journal on Software Tools for Technology Transfer, 2005
8. **H. Kugler, D. Harel, A. Pnueli, Y. Lu, Y. Bontemps** Temporal Logic for Scenario-Based Specifications In (Eds.) *Proc. 11th Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05),* Lect. Notes in Comp. Sci. vol. Springer-Verlag 2005
9. **J.V.Guillen Scholten, F. Arbab, F.S. de Boer, M. M. Bonsangue** Mocha-pi: an Exogenous Coordination Calculus based on Mobile Channels In *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC 2005)* ACM Press. Accepted for publication 2005
10. **Marcel Kyas** An extended type system for OCL supporting templates and transformations. In Proceedings of Formal Methods for Open Object-based Distributed Systems. Accepted for publication. 2005

Related work, continuation of work in Omega

11. **I. Crnkovic, J. Axelsson, S. Graf, M. Larsson, R. van Ommering, K. Wallnau** COTS component based embedded systems - a dream or reality ? In *A Panel organised at the Conference on COTS-Based Software Systems, ICCBSS 2005 in Bilbao, Spain* LNCS 3412 2005
12. **H. Kugler, D. Harel, A. Pnueli, Y. Lu, Y. Bontemps** Temporal Logic for Scenario-Based Specifications In (Eds.) *Proc. 11th Intl. Conference on Tools and*

Algorithms for the Construction and Analysis of Systems (TACAS'05), Lect. Notes in Comp. Sci. vol. Springer-Verlag 2005

2004

Omega Publications

1. **Werner Damm, Bernd Westphal** Live and let die: LSC based verification of UML models In *Science of Computer Programming* 2004
2. **F.S. de Boer, M. Kyas, W.-P. de Roever** Compositional Verification of Object Creation with Interface Invariants being submitted
3. **Marcel Kyas** A Compositional Proof of the Sieve of Eratosthenes in PVS ificau 2004
4. **Marcel Kyas, Harald Fecher** An Extended Type System for OCL supporting Templates and Transformations In *Submitted for publication.* 2004
5. **Kyas, Marcel, Fecher, Harald, de Boer, Frank S., van der Zwaag, Mark, Hooman, Jozef, Arons, Tamarah, Kugler, Hillel** Formalizing UML Models and OCL Constraints in PVS In *Workshop on Semantic Foundations of Engineering Design Languages* Electronic Notes in Computer Science Elsevier 2004
6. **Kyas, Marcel, de Boer, Frank S.** On Message Specification in OCL In de Boer, Frank S., Bonsangue, Marcello (Eds.) *Compositional Verification in UML* ENTCS vol. 101 Elsevier 2004
7. **Iulian Ober, Susanne Graf, Ileana Ober** Validation of UML Models via a Mapping to Communicating Extended Timed Automata In *11th International SPIN Workshop on Model Checking of Software, 2004* vol. LNCS 2989, 2004
8. **Gregor Gössler, Joseph Sifakis** Priority systems In *proceedings of FMCO'03* LNCS 3188 2004
9. **Y. Abdeddaïm, E. Asarin, O. Maler** Scheduling with timed automata. *accepted to TCS* 2004
10. **M. Bozga, A. Kerbaa, O. Maler** Optimal Scheduling of Acyclic Branching Programs on Parallel Machines In *RTSS 2004*
11. **H. Kugler, G. Weiss** Planning a production line with LSCs Weizmann Institute (MCS04-05) 2004
12. **Susanne Graf, Jozef Hooman** Correct Development of Embedded Systems In *European Workshop on Software Architecture: Languages, Styles, Models, Tools, and Applications (EWSA 2004), co-located with ICSE 2004, St Andrews, Scotland* LNCS 3047 Springer-Verlag May 2004
13. **Susanne Graf, Ileana Ober** How useful is the UML real-time profile SPT without Semantics? April 2004
14. **Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, Joseph Sifakis** The IF toolset In *SFM-04:RT 4th Int. School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time* LNCS June 2004
15. **Ingo Schinz, Tobe Toben, Christian Mrugalla, Bernd Westphal** The Rhapsody UML Verification Environment In *Proceedings of the 2nd International Conference on Software Engineering and Formal Methods (SEFM 2004)* IEEE September 2004
16. **Erika Ábrahám, Marcello M. Bonsangue, Frank S. de Boer, Martin Steffen** Object Connectivity and Full Abstraction for a Concurrent Calculus of Classes In *To appear in the LNCS Proceedings of the First International Colloquium on Theoretical Aspects of Computing, ICTAC 2004* 2004

17. **J.V.Guillen Scholten, F. Arbab, F.S. de Boer, M. M. Bonsangue** A component coordination model based on mobile channels In *Journal of Fundamenta Informaticae, Special issue of Foclasa'02* vol. Accepted for publication 2004
18. **F. de Boer, M.M. Bonsangue, J. Guillen-Scholten** Component coordination: From objects to mobile channels In *Mathematical Frameworks for Component Software - Models for Analysis and Synthesis, He Jifeng and Zhiming Liu (eds.)* World Scientific vol. To appear
19. **Joost Jacob** A Rule Markup Language and its application to UML In *Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods, Paphos, Cyprus* LNCS 2004
20. **Marcelo Zanconi** Modélisation et Analyse de Systèmes Temps Réel avec Préemption, Incertitude et Dépendence Institut National Polytechnique de Grenoble June 2004
21. **T. Arons, J. Hooman, H. Kugler, A. Pnueli, M. van der Zwaag** Deductive Verification of UML Models in TLPVS In *Proceedings UML 2004* Springer-Verlag 2004
22. **J. Hooman, N. Mulyar, L. Posta** Validating UML models of Embedded Systems by Coupling Tools In *Proceedings Workshop on Specification and Validation of UML models for Real-Time and Embedded Systems (SVERTS 2004)* 2004
23. **D. Harel, H. Kugler, A. Pnueli** Smart Play-Out Extended: Time and Forbidden Elements In *International Conference on Quality Software (QSIC04)* IEEE Press 2004
24. **D. Harel, H. Kugler** The RHAPSODY Semantics of Statecharts (or, On the Executable Core of the UML) In *Integration of Software Specification Techniques for Application in Engineering* Lect. Notes in Comp. Sci. vol. 3147 Springer-Verlag 2004

Proceedings of events organised by the project

25. **Frank de Boer, Marcello Bonsangue, Susanne Graf, Willem-Paul de Roever** (Eds.) 2nd Symposium on Formal Methods for Components and Objects, revised lectures LNCS vol. 3188 2004
26. **Susanne Graf, Oystein Haugen, Ileana Ober, Bran Selic** (Eds.) 2nd workshop on Specification and Validation of UML models for Real Time and Embedded Systems, SVERTS 2004 Verimag technical report 2004
27. **Susanne Graf, Oystein Haugen, Ileana Ober, Bran Selic** SVERTS - Specification and Validation of Real-time and Embedded Systems, workshop overview LNCS 3297 2004
28. **F.S. de Boer, M. Bonsangue** (Eds.) Compositional verification of UML models In *Post proceedings of the UML 2003 workshop on compositional verification of UML models* Electronic Notes in Computer Science vol. 101 Elsevier Science 2004
29. **F.S. de Boer, M. Bonsangue** (Eds.) Formal Methods for Components and Objects - A theoretical perspective In *Special issue of Theoretical Computer Science* Journal of Theoretical Computer Science vol. In press Elsevier Science 2004
30. **F.S. de Boer, M. Bonsangue** (Eds.) Formal Methods for Components and Objects - Pragmatic aspects and applications In *Special issue of Science of Computer Programming* Journal of Science of Computer Programming vol. In press. Elsevier Science 2004

Related work, input to or continuation of work in Omega

31. **Erika Ábrahám, Frank S. de Boer, Willem-Paul de Roever, Martin Steffen** A Tool-supported Assertional Proof System for Multithreaded Java In *Journal of Object Technology* 2004
32. **J. Hooman, N. Mulyar, L. Posta** Coupling Simulink and UML Models In B. Schneider, G. Tarnai (Eds.) *Proceedings of Symposium FORMS/FORMATS 2004* 2004
33. **Susanne Graf, Andreas Prinz** Time in ASMs - Some problems and solutions In *Forte 2004, work in progress session* October 2004

2003

Omega Publications

1. **Werner Damm, Bernd Westphal** Live and let die: LSC based verification of UML models In Frank de Boer, Marcello Bonsangue, Susanne Graf, Willem-Paul de Roever (Eds.) *Proceedings of the 1st Symposium on Formal Methods for Components and Objects (FMCO 2002)* LNCS Tutorials vol. 2852 2003
2. **Susanne Graf, Ileana Ober** A Real-time profile for UML and how to adapt it to SDL In *SDL Forum 2003, July 1-4, Stuttgart* LNCS (2708) July 2003
3. **Ileana Ober** An ASM semantics for UML Derived from the meta-model and incorporating actions In *Abstract State Machines - Advances in Theory and Applications.* LNCS vol. 2589 Proceedings 10th International Workshop, ASM 2003 2003
4. **Gregor Gössler, Joseph Sifakis** Component-based construction of deadlock-free systems In *proceedings of FSTTCS 2003, Mumbai, India* LNCS 2914 2003
5. **Werner Damm, Bernhard Josko, Amir Pnueli, Angelika Votintseva** Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML In Frank de Boer, Marcello Bonsangue, Susanne Graf, Willem-Paul de Roever (Eds.) *Proceedings of the 1st Symposium on Formal Methods for Components and Objects (FMCO 2002)* LNCS Tutorials vol. 2852 2003
6. **M. van der Zwaag, J. Hooman** A Semantics of Communicating Reactive Objects with Timing In *Proceedings of Workshop on Specification and Validation of UML models for Real-Time Embedded Systems (SVERTS 2003)* 2003
7. **Gregor Gössler, Joseph Sifakis** Composition for Component-Based Modeling In *1st Symposium on Formal Methods for Components and Objects, revised lectures* LNCS Tutorials vol. 2852 2003
8. **Susanne Graf, Ileana Ober, Iulian Ober** Timed annotations in UML In *Workshop on Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS 2003), a satellite event of UML 2003, San Francisco, October 2003* October 2003
9. **Iulian Ober, Susanne Graf, Ileana Ober** Validating timed UML models by simulation and verification In *Workshop on Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS 2003), a satellite event of UML 2003, San Francisco, October 2003* October 2003
10. **Christos Kloukinas, Chaker Nakhli, Sergio Yovine** A Methodology and Tool Support for Generating Scheduled Native Code for Real-Time Java Applications In *EMSOFT 2003* LNCS vol. 2855 2003
11. **Christos Kloukinas, Sergio Yovine** Synthesis of Safe, QoS Extendible, Application Specific Schedulers for Heterogeneous Real-Time Systems In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)* ISBN 0-7695-1936-9 2003

12. **D. Garbervetsky, Sergio Yovine, Marcello Zanconi** Towards symbolic Reachability Analysis for preemptive Schedulers using difference constraints VERIMAG 2003
13. **David Harel, Hillel Kugler, Rami Marelly, Amir Pnueli** Smart play-out In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* ACM Press 2003
14. **Y. Abdeddaïm, E. Asarin, O. Maler** On optimal scheduling under uncertainty In *Proceedings of TACAS 2003, Warsaw* LNCS 2003
15. **M. Bozga, O. Maler** Timed Automata Approach for the AXXOM Case Study Verimag 2003
16. **Susanne Graf** States and events in the context of timed systems Verimag September 2003
17. **Y. Bontemps, P. Heymans, H. Kugler** Applying LSC to an Air Traffic Control Case Study In *Proc. 2nd Int. Workshop on Scenarios and State Machines (SCESM'03)* 2003

Proceedings of events organised by the project

18. **Frank de Boer, Marcello Bonsangue, Susanne Graf, Willem-Paul de Roever** (Eds.) 1st Symposium on Formal Methods for Components and Objects, revised lectures LNCS Tutorials vol. 2852 2003
19. **Susanne Graf, Oystein Haugen, Ileana Ober, Bran Selic** (Eds.) 1st workshop on Specification and Validation of UML models for Real Time and Embedded Systems (SVERTS 2003) In *Verimag technical report 2003/10/22 and http://www-verimag.imag.fr/EVENTS/2003/SVERTS/* 2003

Related work, input to or continuation of work in Omega

20. **Erika Ábrahám-Mumm, Frank S. de Boer, Willem-Paul de Roever, Martin Steffen** A Tool-Supported Proof System for Monitors in Java In *Proceedings of the FMCO 2002* 2003
21. **Erika Ábrahám, Frank S. de Boer, Willem-Paul de Roever, Martin Steffen** A Tool-supported Assertional Proof System for Multithreaded Java In Susan Eisenbach, Gary T. Leavens, Peter Müller, Arnd Poetzsch-Heffter, Erik Poll (Eds.) *Proc. of the Workshop on Formal Techniques for Java-like Programs - FTfJP'2003* 2003
22. **A. Pnueli, T. Arons** TLPVS: A PVS-based LTL verification system In *Verification--Theory and Practice: Proceedings of an International Symposium in Honor of Zohar Manna's 64th Birthday* Lect. Notes in Comp. Sci. vol. Springer-Verlag 2003
23. **T. Arons** Verification of an Advanced MIPS-type Out-of-Order Execution Algorithm In *Proc. 16th International Conference on Computer Aided Verification (CAV'04)* Lect. Notes in Comp. Sci. vol. 3144 Springer-Verlag 2003

2002

Omega Publications

1. **F. Arbab, F.S. de Boer, M. Bonsangue, J. Scholten** MoCha, a middleware based on mobile channels In *Proceedings of COMPSAC 2002* IEEE Computer Scociety Press 2002

2. **J.V. Guillen-Scholten, F. Arbab, F.S. de Boer, M.M. Bonsangue** Mobile Channels, Implementation Within and Outside Components In *Proceedings of Formal Methods and Component Interaction* Electronic Notes in Computer Science vol. 66.4 Elsevier Science 2002
3. **Marius Bozga, Susanne Graf, L. Mounier** IF-2.0: A Validation Environment for Component-Based Real-Time Systems In *Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen* LNCS (2404) Springer Verlag June 2002
4. **J.V. Guillen-Scholten, F. Arbab, F.S. de Boer,, M.M. Bonsangue** A Channel-based Coordination Model for Components In *Proceedings of 1st International Workshop on Foundations of Coordination Languages and Software Architectures* Electronic Notes in Computer Science vol. 68.3 Elsevier Science 2002
5. **D. Harel, R. Marelly** Playing with Time: On the Specification and Execution of Time-Enriched LSC In *Proc. 10th IEEE/ACM Int. Symp. on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002), Fort Worth, Texas* 2002
6. **Jozef Hooman** Towards Formal Support for UML-based Development of Embedded Systems In *Proceedings PROGRESS 2002 Workshop, STW* 2002
7. **Joseph Sifakis** Scheduler Modelling Based on the Controller Synthesis Paradigm In *Journal of Real-Time Systems, special issue on Control Approaches to Real-Time Computing* vol. 23 2002
8. **R. Marelly, D. Harel, H. Kugler** Multiple Instances and Symbolic Variables in Executable Sequence Charts In *Proc. 17th Ann. ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'02)* 2002

Proceedings

9. **Damm, W., Olderog, E.-R.(Eds)** FTRTFT 2002 In Formal Techniques in Real-Time and Fault-Tolerant Systems, 7th International Symposium, vol. 2469 LNCS 2002

Related work, input to or continuation of work in Omega

10. **Susanne Graf** Expression of time and duration constraints in SDL In *3rd SAM Workshop on SDL and MSC, University of Wales Aberystwyth* LNCS (2599) June 2002
11. **D. Harel, H. Kugler, R. Marelly, A. Pnueli** Smart Play-Out of Behavioural Requirements In *FMCAD conference 2002* Lect. Notes in Comp. Sci. vol. 2517, 2002
12. **K. Altisen, G. Gössler, J. Sifakis** Scheduler modelling based on the controller synthesis paradigm In *Journal of Real-Time Systems, special issue on Control Approaches to Real-Time Computing* 2002