Real time system modeling with UML: current status and some prospects

François Terrier, Sébastien Gérard

LETI (CEA - Technologies Avancées) DEIN - CEA/Saclay
F-91191 Gif sur Yvette Cedex France
Phone: +33 1 69 08 62 59 ; Fax: +33 1 69 08 83 95
Francois.Terrier@cea.fr ; Sebastien.Gerard@cea.fr

Abstract

Engineers are more and more faced to the hard problem of developing more sophisticated real-time systems while time to market and cost constraints are constantly increasing. The necessity of adopting object oriented modeling in the real-time domain appears to be essential in order to face the rapidly changing market conditions. Main obstacles are the lack of standards and no good adequation with real-time needs. With the standardization of UML 1.3, the first main drawback is coming down. Current work performed at the OMG on UML standard evolutions to better integrate real-time issues shows both that there is a strong interest on the subject and that current proposals are neither completely satisfying, neither completely compatible. In particular , some available methods (namely ARTiSAN, RT-UML, UML/SDL and UML-RT) provide a good support for parallelism modeling of an application. Nevertheless, they but are often poor to express quantitative real-time features (such as deadlines, periods, priorities…) and are always based on proprietary solutions not obviously fully compliant with the UML standard. So It remains two important challenges: first to define how to support high level real time specification concepts in UML; and, second, to unify them within these proposed by existing tools.

1. Introduction

Several years ago the market of real time and embedded systems was considered as a verify specific and “confidential” sector (about 5% of the global market of software based systems). Now several studies of the evolution of software market consider that embedded systems can represent near years 2003 more than 50% of the global market (including personal computers, client-server and information system applications).

This explosion of the market of real time systems and the constantly enrichment of the embedded services urge engineers to face more and more to the hard problem of developing sophisticated real-time systems while competition through time to market and cost constraints increase day after day. Classical real-time development of software systems is reaching its limits in a world were in one hand hardware targets cannot be known in advance and where in the other hand version evolution becomes increasingly fast and time to market must be shorten drastically in order to meet economical requirements. Reusability and evolutivity become mandatory requirement for the modeling methods and development techniques. In such a context, real-time systems development cannot be achieved efficiently without a strong

methodological support and accompanying tools. In parallel, object oriented techniques have reached a sufficient level of maturity in order to provide successfully the flexibility required by new applications. Up to now however, real-time community has long been reluctant to cross the Rubicon for mainly two reasons:

- object orientation offer was not mature enough to provide stability in their solutions (methods, tools,...);
- and the real-time specificity was generally not well covered by existing methods.

During the last years *UML* has become the lingua franca among system modellers all over the world. With the arrival of ***UML*** standard [OMG 97], the signal that many editors were waiting for has appeared, and a first step is being achieved that will permit the spread of a new tool generation. Indeed, object oriented modeling with a standard formalism such as UML brings significant answers to the issues mentioned previously. At least because, UML becomes “de facto” a widespread standard for software analysis and design, but also because object oriented approaches fit now quite well with both following needs:

- to have a fine level of modularity for component based development;
- and to improve reusability properties of subsystems.

Although its presence in the software domain has been successful, *UML* still lacks significant semantics that will allow its dominance in specific domains, like the one of real-time systems; the description of the real-time behaviour of such systems is not completely satisfying yet. This leads to maintain the debate on the adequacy of object oriented methods with real-time specificities. In particular, this is the case for the real-time behavior specification of an application and its mapping on implementation level where generally concepts of objects have not been introduced until now. At least current proposals are not yet completely satisfying, from two points of views:

- first, its remains some lacks to express timing properties (deadlines, periods, priorities, execution time, …) and some points of semantic on the dynamic models have to be clarified;
- second, users needs are evolving due to the market explosion. In particular, by the fact that real-time system modeling concerns more and more non real-time specialists willing to specify, and even sometimes to develop, new embedded services or systems.

First point can be tackle by enriching and clarifying semantics of current UML concepts connected to real-time modeling, but this point leads to necessity of proposing new modeling concepts at higher level hiding completely underlying implementation techniques and solutions. This requires consequently to provide more elaborated procedures (and also tools) to automatically generate application implementations.

2. Current proposals for real time system modeling with UML

A certain number of solutions have resulted in some commercially available methods (and tools) such as ***ARTiSAN*** [31, 32, 33], ***UML/SDL*** with ***ObjectGeode*** [Arthaud 95], [Leblanc

96], [38, 39, 40], ***ROOM*** with ***ObjecTime*** [23] that becomes now ***UML-RT*** with ***Rose-RT***) [35, 36] and ***RT-UML*** with ***Rhapsody*** [28, 29, 30]. Though a great effort has been done in order to provide a good compromise between task oriented modeling and object oriented one, these proposals reflect a great diversity in the possible methodological choices offered to handle real-time and object orientation together. These proposals generally maintain two separate conceptual models : the object one and the task one. However, both models belong to the same global application model and require interaction among their elements. Their integration in a global and consistent unified UML model is very variable. These four principal approaches are briefly discussed in the following section after a rapid presentation of current status of UML 1.3 [18].

2.1 UML 1.3

Current status of UML [18] proposes only few concepts for real-time modeling and the most relevant for real-time aspects are: the concept of active object, the management of state automaton and the expression of timing specifications.

2.1.1 Active objects

An active object is the support of parallelism within an application. It is attached to an independent thread of control that allows it to perform processing in parallel with the rest of the active objects of an application. It is not specified if it has only one thread or if it can have several threads, even if it is generally understood as “one thread (and only one) by active objects”. No particular relation is made between the messages it receives and the use of the thread to process them (or not). Implementation can be specified through two stereotypes “process” (then it will run in its own address space) and “thread” (that allows active objects to share the same address space).

2.1.2 Sate machines

The behaviour of an active object can be described thanks to a state machine (as any UML object). The management of the state machine is defined by the “Run To Completion” (RTC) assumption. This hypothesis means that an event can only be dequeued and dispatched if the processing of the previous current event is fully completed. It stopped the processing when it reaches a state where no transition can be triggered. UML state machines are widely used to define real-time behavior of an application. However, UML does not set formally a link between the fact that an active object has a thread and the thread used to execute its state machine (so, it is possible to execute it in another thread than this attached to the active object…). Transitions in the state machine can be triggered by several kinds of events: receipt of an operation call, receipt of a signal, receipt of a time event and state change on the object. UML state machines have a message queue that allow to store incoming messages during the RTC step. The order of dequeuing the events is not defined in UML, leaving open the possibility of modelling several priority based schemes.

2.1.3 Timing specifications

UML proposes two ways to specify timing properties. The first one relies on the use of timers in the state machines. Timers setting allows to generate a event at a date defined in the model. The second one is through specification, in the sequence diagrams, of point of time (dates) attached to particular events, such as: receipt of a message, end of processing of a message… Until now, these specifications have only the status of informal information that are not formally connected with the other parts of the model. Taking them into account requires to the developer to translate them into low level instructions inside the implementation code that try to provide to the application a behaviour respecting them… On both point UML requires clarification of the time semantic and on the eventual impact of these specifications. In particular scheduling policy is kept completely under the responsibility of the developers.

2.2 ARTiSAN

ARTiSAN [31, 32, 33] maintains two completely separated models: the first one is for task architecture specification using concepts of task, signal, semaphore, channel… This model called concurrency diagram use specific notations and concepts that are an extension of DARTS [34]. For the moment, they are not related with UML standard elements. At this time, the tool does not provide any support for automatic implementation of the concurrency model. The second model is a classical object oriented model of the application structure. The use of both orthogonal models requires the developer to manage in parallel these two conceptual points of view and needs that he/her has a high competency both in real time development and in object oriented modeling and programming. After building both specific models, links between them are performed by putting the objects inside the tasks that will run them. Communication among objects being in different tasks must be completely manually programmed and implemented with level elements defined in the task model (functions and concepts usually provided by the real-time operating system). Real-time constraints are expressed in the model only through the implementation of the mechanisms that try to respect them. Typically period constraints of processing are translated into priorities assignment on the tasks when implementing them…

2.3 RT-UML

RT-UML (Rhapsody) [28, 29, 30], also maintains two orthogonal models: concurrency/task and objects. The concurrency model is there based on standard UML concepts. Within this approach the active object is considered as the representation of the real-time concept so-called a “task”. This model is limited to the definition of the task and of their connections. A task appears also as a composite object grouping objects of the structural model that participate to the task. In addition, this approach introduces the notion of “reactive object” having a state machine that describe its behavior. Communication among tasks (active objects) is performed through message passing among the reactive objects attached to the tasks. This ensures a better level of integration between objects and task models.

Communication through signal sending and operation call are distinguished and are associated to different mechanisms and behavior of the reactive objects:

- signals are received in a queue and will be extracted to trigger a transition of the object state machine, they are processed asynchronously and may convey input parameters. Moreover the action executed following the transition firing are controlled within the treatment resource of the target object.
- operation call may have two status. Either the call is directed forward an operation call "primitive" or an operation call "triggered". In the first case, the operation call is seen inside the target object as a classical object oriented operation call. Its receipt may also start the execution of the method implementing the called operation. The second case the operation call is seen as an event receipt. In this case, the operation call is stored in the event queue of the target object and will be dequeued latter. The call event is also dispatched to the state machine of the object, triggering one of the transition or been lost or even deferred (as for the signal events).

In both cases, operation calls are processed in a synchronous way using the execution resource of the task (active object) supporting the sending of the message. When return values are needed, the sender has to be sure that the target object is in a state where it is able to manage the call event. Mixture of both communication mechanisms leads to application having complex behaviors. The combined use of these different communication schemes may lead to concurrency problem within an object. For example, an object, executing a sequence of actions following the receipt of a signal may too accept at the same time a "primitive" operation call. Both communications running into different treatment resources may also impair the integrity of the object if the sequence of actions and the method implementing the primitive operation access to the same attributes of the object. Finally priority values can be assigned to the active objects, but not on the event they can receive. Consequently the relations between constraint of priority on processing and priority values on the active objects (tasks) can become very hard to understand and manage. In particular, due to the fact that several "reactive objects" can be assigned to a single "active object". Real time constraints are also expressed here only through the implementation of the low level mechanisms that try to respect them.

2.4 UML-RT

UML-RT (Rose-RT) [23, 35, 36], proposing a solution very close to the previous one. However the concurrency model remains a logical model of parallelism. By this way it provides an approach a little bit more integrated. Actual mapping between active objects and tasks is decided only at the final step of the application implementation (i.e.: a task will support the implementation of one or several active objects). Active object concept proposed here is quite specific and is called "capsule". As "reactive objects" of Rhapsody, capsules have a state automaton describing their behavior and the processing triggered by the messages they receives. The main difference concerns the communication model. UML-RT allows communication among "capsules" only through dedicated connection links called "ports".

These “ports” define the set of the messages that can be exchanged between tow capsules and their relative order. They define a point to point communication protocol. Consequently the interface of the “capsule” object is defined through is external “ports” and their associated “protocols”. Indeed, a “port” is always linked to a concept called a “protocol”. This one defines the input and output signals that an associated “port” can convey. The main drawback of this approach is that it provides specific concepts that are not currently known and manipulated by a large community of object oriented developers. Indeed, the paradigm used within UML-RT is closer to this one used in SDL formalism. At the implementation stage, the “capsules” have to be assigned to thread of the underlying operating system. The mapping is fully kept under the developer responsibility. He/she has to decide how the capsules will be supported by each thread of the system. At the initial modelling stage of an application priority value can assigned to message sending. The event receipt by the target object are then stored in different prioritised queues. Each capsule dequeues the incoming events in the order of the input queued according to their priority levels (from the highest to the lowest). Later in the development (at the implementation stage) when the developer assigns capsules to threads on the underlying operating system, he/she as to attach a system priority to each thread. So they is two different levels of priority specification within this approach. An the main issue of this feature is a lack of consistency between these two ways to specify priorities within an application. In addition, there is the same limitation than for the others for the expression of timing specifications and the approach requires also big ability to translate of high level real-time constraints into low level mechanisms implementing them.

2.5 UML/SDL

UML/SDL (ObjectGEODE/TAU) [37, 38, 39, 40] proposes to use the SDL formalism [41] in association to UML. The use of UML is used at the early stage of the development, typically for the analysis stage, while SDL is postponed at design and implementation stage. Concurrency specification appears in both models: at the analysis stage, the UML model specifies the structure of the application in terms of objects. Some objects can be identified at this level as being active objects that will introduce parallelism in the application. At the next step, it is proposed that passive objects will lead to the definition of the Abstract Data Types of SDL, while the active objects are interpreted as SDL processes. But this proposal does not rely on any formal rules and the designer has to be careful during this step of model translation from UML to SDL formalisms. This translation can break the seamless feature of object oriented approaches. The state machines of the active objects are used to defined the state machines of the SDL processes. However, semantic differences between UML state machines and process state machines must be managed carefully by the developer. Some other points must also be clarified, in particular how to manage in SDL passive objects that are used (shared) by several active objects. Communication is specified with SDL concepts and, thus, focuses on signal exchanges and not on operation calls. Links between SDL communications and the initial UML model can be difficult to maintain. In addition, we must mention that the final implementation model as a strong influence on how to organize the

SDL process model. And thus, if we want to have a good mapping on a given implementation model, we must take it into account very early in the model. This restrict the real independency between implementation choices and application modeling. Finally, we find here again the same limitations as previously for the expression of timing specifications that requires their translation into low level mechanisms implementing them (also only specified thanks to timers).

3. Some open questions on UML evolution for real-time modeling

Solutions described before often propose specialized models or paradigms to model the real-time point of view of an application. This makes difficult the communication and interaction between usual object oriented modeling and the real-time one. Moreover, they generally maintain two separate conceptual models: the object one and the task one. This requires, from the designers, high level skills for maintaining consistent these two different views, all along the development process.

At this time the OMG leads works to improve the integration of real-time issues into UML. This concerns mainly two points:

- UML profile for scheduling, performance and time [RFP 99]: the objective is to clarify the semantic of the concept of time provided by UML and to specify the low level services and concepts needed to implement real-time applications (task, semaphore, priority…). This will concretely leads to the definition of an object oriented real-time "virtual machine" (or simply a UML interface with real time operating systems).
- Action semantics for the UML [RFP 98]: the objective is to formalize the semantic of the different actions that can be specified in a model, in particular the actions specified on the transitions of the state machines. No specific notations will be provided as "standard", only their semantic and integration into UML meta-model.

Even with the results of the current works leaded by the OMG, it is clear that the solutions and concepts proposed address more specialist of real time system development than common people developing object oriented applications. In particular, three examples of divergence between approaches for real-time specialists and the common object oriented programming paradigms can be pointed out:

- The role of the interface of active objects (or "capsules" or "reactive objects") that becomes unclear: while in traditional object oriented development it defines the behavior of the object (i.e.: the only way to communicate with it and the processing triggered by the messages it receives), in current proposals either active objects have no such interface either they have one but the most important part of their behavior is defined outside: in the object state machine.
- The communication schemes are also considered with different points of view. Indeed, real time approaches have based the communication schemes essentially on signals that can not be mapped to usual operation calls (for example, they can not support easily the usual notion of output parameters of the operations). In addition, the execution models of signal

processing and of operation calls (when they are supported) are completely different: signals are associated to asynchronous communications that trigger the processing in the target object using an other processing resource than this of the sender of the signal (this creates concurrency); while operation calls generally are attached to synchronous communications that trigger immediately their processing in the target using the calling resource (no concurrency created).

- The state machines used to specify active object behavior describe both which messages the object can process, in which conditions it is possible and which actions are performed to implement the object reaction to the messages it receives. This breaks the usual structure of object oriented approaches where the program is first described in terms of objects, second, at a finest level, each object is described in terms of operations and attributes and, finally, at the lowest level each operation is described in terms of a set of elementary actions. In fact, due to the emphasis put on state machine specification and to the mixture of object level control specification with low level processing action specification, all the proposals described above loose the structural feature of the operation level.

In addition, timing specifications are always introduced either through informal information (i.e.: time points identified in sequence diagrams) or by timer definitions in state diagrams. This imply that the developers always need to translate the real-time constraint defined in requirement documents into implementation mechanisms. Moreover, setting of priorities on task and definition of scheduling policies (either for the application task management or for the messages management in the queues) is not really integrated in the models. This requires again to the developers to have a deep knowledge of these real-time mechanisms.

4. A prospective approach

Several years ago studies have been performed to extend object oriented programming paradigm to concurrent programming [13, 14, 27]. They have lead to the definition of active object concepts integrating better both set of paradigm in fully object oriented and unified models. They have been the basis for the definition of the real-time objects paradigm supporting real-time developments with automatic real-time implementations keeping the low level real-time implementation mechanisms transparent to the developers [8, 9, 10, 11, 12].

The *ACCORD* approach is one such attempt [5, 6], [Terrier 97], [Gerard 1998]. Its objective is to provide a framework for real-time development as close as possible to classic object oriented methods. Thanks to high level abstractions (namely the real-time active object concept [12]) real-time modeling can be achieved without mixing up implementation issues with domain specific one. The main purpose of this new concept is to offer to engineers a rich and efficient abstraction that will allow them to model very fast complex real-time applications. It aims to answer to their needs of describing real-time features of their applications. So within this approach, it is tried to keep the well-known advantages of object oriented techniques yet with the idea to integrate in this model real-time features in order to provide sufficient abstractions of the real time field.

To achieve this objective, ACCORD/UML proposes an extension of UML supporting both aspects of a real-time behavior specification: quantitative and qualitative. For that, the main modeling features introduced by ACCORD/UML method are:

- Concurrency level of the application is defined through identification of the real-time objects (extension of UML active objects). ACCORD method proposes to define them on the basis of the following heuristic: real-time objects are either interface objects of the application managing events coming from the application external environment either objects that need to perform automatically processing, in particular for the periodic activities. Implementation of these real-time objects is taken in charge by a dedicated implementation frameworks supporting task allocation, object queue management and inter-objects communications. In addition, the method allows to define passive object shared by several real-time objects, they are called “protected objects”. In this case concurrency management of their parallel use by several objects is automatically implemented by the ACCORD implementation framework.
- The specification of quantitative real time constraint on the model itself (deadlines, periods, priorities, etc.). These specifications can be set from the early stage of the development and are maintained during model enrichment until final implementation. Executing model of real-time objects and ACCORD implementation frameworks avoid to require the developers to translate the real time constraint into low level mechanisms implementing their management (timers setting is no more necessary and scheduling policies are automatically provided and implemented in the application through the implementation framework). Concurrency constraints are managed in the same way by the approach: there are declared at the operation level (because they relies on their implementation specification) and automatically managed in the final application. Scheduling policies integrate these both types of constraints (i.e.: the real-time and concurrency constraints).
- Communication scheme among objects remains very close to this used in sequential object oriented programming: objects can exchange messages both through operation calls (with output parameters if needed) and also with signal sending (in this case, it also allows to have broadcast and anonymous communications: the sender does not need to know the target objects and target objects do not need to know the senders of the signals). Execution model for operation call and signal sending are the same: they are processed in the processing resources of the target if it is a real-time object and they are processed in the sender processing resource when the target is a passive object.
- Usual object oriented structure of the model is preserved by specification of object behavior with two levels of state machines. First, at the object level state machines only specify the logical control of chaining the operations performed by the object (it corresponds to the “protocol state machine” defined in UML 1.3). In this state machine the transitions only trigger the processing of an operation defined in the object interface (actions are restricted to local invocation of an object method). Second, at the object operation level, state machines specify the detail of the actions performed to execute this

operation (actions used at this level are chosen to be consistent with the semantic resulting of the OMG works on “action semantic”).

Even if this approach is the result of laboratory studies, its operational feasibility has been demonstrated by its complete implementation on an industrial UML modeling case tool: Objecteering 4.3[1]. The available configuration involves a set of modules allowing the generation of the real-time behavior of an object from its state machine specification into C++ code. Moreover, a specialized C++ generator has also been developed in order to support the concepts of the extended active objects. Finally, with respect to the underlying real time operating system (RTOS), two layers have been defined between the application and the RTOS, namely *ACCORD/UML Kernel* and *ACCORD/UML Virtual Machine*. The first one implements mechanisms supporting active object semantics and above all the mechanisms allowing to schedule application tasks respecting the main scheduling policies such as the Earliest Deadline First (EDF) policy or the Highest Priority First (HPF)[2] policy . The second one gives the application independence as far as the underlying RTOS is concerned. The latter, exists for Solaris and VxWorks5.2.

Integrating the active object paradigm within an object oriented method permits a simple, complete and homogeneous modeling of requirements and needs for multitasking within an application. The use of the same object paradigm along the whole development cycle of an application reinforces the seamless of the model among the several modeling stages. The ACCORD/UML proposal for extending UML to real-time is based on this paradigm enriched for real-time concerns. On this basis we can obtain with very few specialization an enrichments of UML, the frame for methods supporting real-time development with intellectual approaches and concepts very close to those used in usual object oriented modeling. Moreover, this proposal allows to postpone all the parallelism, real-time and implementation choices very late in the design. Thus changing parallelism granularity will not cause the redesign or reanalysis of all the application, but will just require the tuning of implementation choices and of the last real-time design of the application. The approach maximizes reusability and designers may fully benefit from object oriented experience acquired in other domains.

In addition to an obvious benefit in terms of quality due to the fact that constraints related to multitasking and real-time behavior of the application can be expressed in the model itself, the use of the ACCORD/UML approach prevents users from having a deep knowledge of fine synchronization mechanisms used in multitasking programming. Its similarity with usual object oriented programming model allows its use by non real-time specialists and the ability to implement automatically the real-time features specified in the model ensures that they do not need to have a deep knowledge of the real-time development techniques.

---

[1] Objecteering is trademark of Softteam.

[2] in each cases constraint inheritance mechanisms have been introduced to ensure the consistency between the global scheduling policy, operation concurrency constraints and real-time constraints put on the processing/messages [Rioux 97]

This way they can concentrate on the modeling and on application specific code rather than wasting precious time in tuning low level mechanisms. Development time is saved, it becomes possible to develop quite rapidly multitasking applications whose behavior can be as reliable and efficient as code obtained with the use of classical techniques using ad hoc system solutions for each application.

Further works are performed in the framework of the AIT-WOODDES[3] project. They focussed on two points:

- To define a UML profile dedicated to embedded system modeling for the automotive industry. This profile will be presented at the OMG with eventual standard evolution proposals required to support it.
- To obtain formal UML models of a real-time application. On this point studies are currently under progress at the LSP Laboratory of the CEA-Leti to translate extended UML models into a model based on asynchronous communicating automata. It will then be possible to analyze these automata -that will be automatically build- with existing tools and methods for validation. In particular ACCORD/UML models will be interfaced with the *AGATHA* tool [Gallois 97], [Dupuy 98], [Gallois 99], [Fievet 2000]. This tool relies on proof and model checking techniques and help the engineer to validate the behavior of his or her application, in particular by automatically extracting from the UML model all the possible execution paths of the application and by generating test cases of each of them.

5. Bibliography

[OMG 97] UML Proposal to the Object Management Group, Version 1.1, September 1997.

[31] A. Moore and N. Cooling, “Real-time Perspective - Overview (Version 1.0),” : ARTISAN Software, 98.

[32] M. J. McLaughlin and A. Moore, “Real-Time Extensions to UML,” Dr. Dobb's, 98.

[33] A. Moore, “Why tasks aren't objects and how to integrate them in your design,” Embedded Systems, pp. p18-30, 00.

[Arthaud 95] R. Arthaud, OMT-RT : Extensions of OMT for better Describing Dynamic Behavior, in proc. TOOLS Europe’95, Versailles, France, February 1995.

[Leblanc 96] P. Leblanc, V. Encontre, ObjectGeode : Method Guidelines, VERILOG SA, 1996

[38] P. Leblanc, “Object-Oriented and Real-Time Techniques: Combined Use of OMT, SDL and MSC,” in Current Issues in Electronics Modeling Series, 96.

[39] ITU-T, “SDL + Methodology,” ITU-T, Geneva 96.

[40] V. Perrier, “Combining Object-Oriented and Real-Time Programming from an OMT and SDL Design,” Verilog France, Toulouse.

[23] B. Selic, G. Gullekson, and P. T. Ward, Real time Object-oriented Modeling: John Wiley & Sons, Inc., 94.

[35] B. Selic and J. Rumbaugh, “Using UML for Modeling Complex Real-Time Systems,” ObjecTime Limited 98.

[36] ObjecTime, C++ Language Add-in, C++ Language Guide, ObjecTime Limited ed, 00.

[3] AIT- WOODDES project, n° IST-1999-10069, is partly funded by the European Commission and involves the following partners: Peugeot Citroën Automobiles – France, Mecel – Sweden, Commissariat à l’Energie Atomique – France, Telelogic/Verilog France, I-Logix Ltd – Israel, Intracom S.A – Greece, Uppsala University – Sweden and Kuratorium OFFIS e.V. – Germany.

[28] B. P. Douglass, Real-Time UML : Developing Efficient Objects for Embedded Systems, 98.

[29] B. P. Douglass, Doing Hard Time : Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns, 99.

[30] D. Harel and E. Gery, “Executable Object Modeling with Statecharts,” IEEE Computer, vol. july, pp. p31-42, 97.

[18] OMG, “Unified Modeling Language Specification,” OMG, Request for proposal ad/99-06-99, June 99.

[34] H. Gooma, Software Design Methods for Concurrent and Real-time Systems, 92.

[41] R. Z. ITU-T, “Specification and Description Language (SDL),” ITU-T, Geneva 96.

[RFP 98] OMG, RFP ad/98-11-01 “Action semantics for the UML”, 1998 (www.omg.org/techprocess/meetings/schedule/Action_Semantics_for_UML_RFP)

[RFP 99] OMG, RFP ad/99-03-13, “UML profile for scheduling, performance and time”, 99 (www.omg.org/techprocess/meetings/schedule/UML_Profile_for_Scheduling_RFP)

[13] A. Yonezawa, ABCL: An object-oriented concurrent system: MIT Press, 90.

[14] D. G. Kafura and K. H. Lee, “Act++: Building a concurrent C++ with actors”, Journal of Object-Oriented Programing (JOOP), pp. 25-37, 90.

[27] C. Atkinson, Object-oriented reuse, concurrency and distribution : an Ada-based approach: Addison-Wesley, 91.

[8] Y. Ishikawa, H. Tokuda, and C. W. Mercer, “Object-oriented real-time language design: Constructs for timing constraints”, ECOOP'90 & OOPSLA'90), Ottawa.

[9] K. Takashio and M. Tokoro, “DROL: An object-oriented programming language for distributed real-time systems”, OOPSLA'92.

[10] L. Nigro and F. Tisato, “RTO++: A framework for building hard real-time systems”, Journal of object-oriented programming (JOOP), pp. 35-47, 93.

[11] G. Fouquier and F. Terrier, “Introducing priorities into a C++ based actor language for multithread machines”, TOOLS PACIFIC'94, Australia, 94.

[12] F. Terrier, G. Fouquier, D. Bras, L. Rioux, P. Vanuxeem, and A. Lanusse, “A Real Time Object Model”, TOOLS Europe'96, Paris, France, 96.

[5] F. Terrier, A. Lanusse, D. Bras, P. Roux, and P. Vanuxeem, “Concurrent objects for multitasking”, L'objet, vol. 3, pp. 179-196, 97.

[6] A. Lanusse, S. Gérard, and F. Terrier, “Real-Time Modeling with UML : The ACCORD Approach”, UML'98, Mulhouse, France.

[Terrier 97] F. Terrier et al., Des objets concurrents pour le multitâche, Rev. L’Objet, Ed. Hermès, V3-2, 1997.

[Gerard 1998] S. Gérard et al., Developing applications with the Real-Time Object paradigm : a way to implement real-time system on Windows-NT, in Real-Time Magazine, 3Q,1998

[Rioux 97] L. Rioux et al., Scheduling Mechanisms for Efficient Implementation of Real-Time Objects, in ECOOP'97 Workshop Reader, S. Mitchell & J. Bosch Ed., Springer Verlag, December 1997.

[Gallois 97] Jean-Pierre Gallois et Agnès Lanusse. Le test structurel pour la vérification de spécifications de systèmes industriels. Génie Logiciel, 46 :145-150, 1997.

[Dupuy 98] Stéphan Dupuy et Jean-Pierre Gallois. Génération automatique de jeux de tests à partir de spécifications Statemate, ICSEA’98, 1998. Mots clés : Statemate, PSA.

[Gallois 99] Gallois, J.-P., A. Lapitre, et al. (99). Analyse de spécifications industrielles et génération automatique de tests. ICSEA’99, Paris.

[Fiévet 2000] Emmanuel Fievet, Jean Pierre Gallois, Arnault Lapitre. Analyse de spécifications STATEMATE et génération automatique de tests. RTS’2000, 153-170, mars 2000.