From UML to AADL: a Need for an Explicit Execution Semantics Modeling with MARTE

Matthias Brun, Madeleine Faugère, Jérôme Delatour, Thomas Vergnaud

From UML to AADL:

a Need for an Explicit Execution Semantics Modeling with MARTE

Matthias Brun[1], Madeleine. Faugère[2], Jérôme Delatour[1], Thomas Vergnaud[3]

1: ESEO, 4 rue Merlet de la Boulaye, 49009 Angers Cedex - France
2: Thales Research and Technology Software Research Group, RD 128 - 91767 Palaiseau Cedex - France
3: CNES, 18 avenue Edouard Belin, 31055 Toulouse Cedex - France

Abstract:

A modeling process for real-time embedded systems may involve the coordinated use of several languages. Each of these languages are dedicated to a particular phase of development (specification, design, test, ...) and coupled with various tools (scheduling analysis, formal verification, model checker,...). The combined use of UML and AADL is an increasing practice. UML and its recent MARTE (Modeling and Analysis of Real-Time and Embedded systems) profile seem suitable for capturing requirements, analysis and preliminary design. AADL is tailored for the detailed design phase and offers linked validation and verification tools.
In order to combine UML/MARTE and AADL, translation mechanisms between these two formalisms have to be defined. Previous works have defined translations between the structural concepts of AADL and MARTE artifacts. However, the behavioral aspect have also to be treated.
The presented work focuses on the translation of the thread execution and communication semantics. It is a pragmatic and on-going approach, validated in an industrial context, on representative examples.

Keywords: UML, MARTE, AADL, thread execution semantic

1. Introduction

In order to overcome the increasing complexity of real-time embedded systems, the coordinated use of several modeling languages is more and more investigated. Indeed, different software specialists (in analysis, software architecture, design, real-time scheduling, test, safety...) must cooperate. For various reasons (cultural, educative, on the availability of validation and verification dedicated tools...), each of these specialists could use different modeling formalisms.

The combined use of UML and AADL is an example of that kind of practice. UML and its recent MARTE (Modeling and Analysis of Real-Time and Embedded systems) profile seem suitable for capturing requirements, analysis and preliminary design. AADL is more tailored for the detailed design phase and offers tools such as scheduling analyzers, code generators.

In order to allow the use of UML/MARTE and then, AADL, translation from UML to AADL has to be defined. Different approaches could be considered. We choose to work on the study of the AADL concepts and their representations in UML/MARTE. Different reasons explain this choice: the semantic gap between these two formalisms could be huge (and some UML/MARTE concepts are certainly not present in AADL), UML/MARTE allowed different modeling styles (Object-Oriented, Component-Oriented) whereas AADL is a component approach, an annex to the MARTE standard already deals with the translation between AADL constructs (components and features) and MARTE artefacts.

In this approach, we extend MARTE annex and previous work initiated by Thales [3] by the consideration of the behavioral semantics translation.

The presented work focuses on the translation of the thread execution and communication semantics. These parts are those which are precisely defined in the AADL standard. The other parts, belonging to the behavioral aspect, are still in discussion in the AADL standard.

After an overview of the UML/MARTE and AADL languages, a simple example will be presented. Based on this example, the translation of the thread execution semantics and the communication semantics will be detailed. Finally, we draw conclusions.

2. UML/MARTE and AADL overviews

2.1. UML/MARTE

MARTE (A UML Profile for Modeling and Analysis of Real-Time and Embedded systems) [1] is the new UML profile extension for real-time and embedded systems standardized mid 2007 at OMG (Object Management Group). MARTE defines concepts in terms of UML extensions needed to model and analyze real-time and embedded systems (RT/ES).

Today, UML is a well spread and used language in the Software Engineering Community, supported by many commercial tools. This unified language

provides a useful abstraction to object-oriented language and approaches, facilitating application design as well as common and simple means of communication between designers. With MARTE, these capabilities are extended to real-time and embedded domains, enabling both 1) RT application, software platform and hardware design, and 2) validation when coupled to verification tools through analysis characteristics specification.

MARTE proposes different artefacts to characterize a RTE system: 1) a non functional-property and value specification language 2) a rich time modeling extension allowing to explicitly design the relation of concepts to clocks (during modeling or analysis phases) 3) software and hardware modeling capabilities 4) allocation capabilities providing a link between applications and platforms, 5) support for quantitative analysis (i.e. scheduling and performance) [1] [2].

The benefits of using this profile for the designers are manifold. First, this language provides a common way to share RTE models between designers: multiple views (functional, non-functional, analysis, software and hardware resources, components…) may be shared between different actors to validate specific application aspects. Secondly, UML/MARTE allows the modeling and thus the capitalization of specific RT languages, and operating systems through run-time or run-time API modeling. Such run-time libraries greatly help the designer to align application domain semantics to run-time behaviours. Finally, such explicit run-time features facilitate model transformation specification.

2.1. AADL

AADL (Architecture Analysis & Design Language) [5] is an architecture description language (ADL) standardized by the Society of Automotive Engineers (SAE). It is particularly targeted at the description of distributed real-time and embedded systems.

The AADL standard defines both textual and graphical syntaxes to describe architectures based on components. Unlike UML, AADL provides a single view (with different possible syntaxes) to represent models. Thus, all information can be integrated on a single diagram (or text file).

The AADL standard defines several categories of components; each category corresponds to specific semantics. Processes, threads, thread groups, data, subprograms and subprogram groups are used to model software applications. They allow for the modeling of data structures and software topologies. Processors, virtual processors, memories, buses, virtual buses and devices are used to describe the hardware environment topology on which software applications are deployed. In addition, processors and devices can be used to describe the hardware elements of an application. AADL systems and abstract components are used as containers for other components or to allow for a certain level of inaccuracy in the early stages of the modeling process; they help in structuring architectures. Components can contain subcomponents; hence AADL architectures are hierarchical.

AADL components can have interfaces, called "features". Features can be communication ports, required or provided accesses to subcomponents, or required or provided accesses to subprograms. Thus, the AADL syntax can explicitly describe common communication paradigms (message passing, remote procedure calls, distributed memory).

Component composition describes the structure of architectures. AADL allows for the characterization of these components: properties can be associated with each architecture element (components, subcomponents, features, connections, etc.). AADL properties are used to specify constraints (e.g. execution time, memory size) or characteristics (e.g. period, thread dispatch policy). They can also be used to describe configuration parameters (e.g. the network address of a processor). AADL defines a set of standard properties and defines semantics for them, so that they can be interpreted by analysis and generation tools.

AADL constructions describe application structures that encapsulate algorithms. Algorithms can be specified using some syntactical constructions, through a behavior annex, or by associating source code with AADL components. Algorithms control the execution of the application, according to the different parameters provided by the architecture description (topology and execution parameters given by AADL properties). AADL descriptions gather all required information to define applications. Therefore, they can be used as backbones for various processings, including documentation generation, simulations, formal analysis and code generation.

Algorithms encapsulated in AADL components can interact with the outside environment through the component features. The AADL standard defines semantics associated with AADL constructions. Hence AADL specifies the actual behavior (regarding inter-component communications and execution) implied by the architectural constructions. AADL tools can rely on the same asumptions for their processings. The execution & communication semantics defined by the AADL standard restricts the possibilities for application design: for example, it is difficult to describe multimedia-oriented applications.

In counterpart, because of the implicit behavior specifications associated with AADL constructions, AADL models can be processed by application generators [13] that produce source code from the AADL components. Generated source code is linked

with algorithm source code through specified API. AADL applications are to be executed on the top of an AADL runtime that provides scheduling and communication services. Such a runtime is configured according to the architecture description.

3. A simple case study : a client/server architecture

In order to better communicate the need to make explicit the execution semantics of AADL components with UML/MARTE, the simple client/server architecture is used in this paper to illustrate the development process of a system with both UML/MARTE and AADL. This simple case study aims to illustrate the positioning of AADL relating to UML/MARTE during the development process. It also aims to underline the impact of the implicit concepts involved by the AADL standard.

This section introduces the case study using UML 2.1.1 [9].

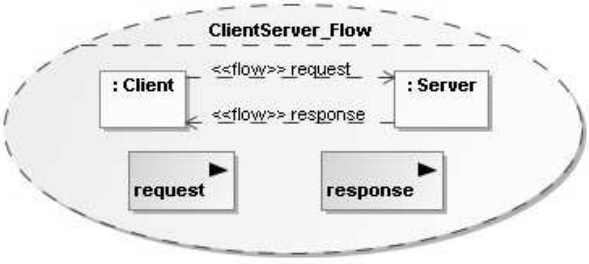
The client/server pattern involves major concepts met in real-time systems: concurrent entities (known as «*schedulable resources*» in UML/MARTE and «*threads*» in AADL) and communication between these kinds of entities.

As shown in figure 1, at the highest level of representation, the client and the server are two entities that communicate. This can be described in UML. The communicated protocol are not specified at this level.

Figure 1 : A client/server architecture.

Figure 2 details the communication describing the response of the server to a client's request.

In this more detailed view, information flows[1] could be refined using message communication (synchronous or asynchronous) between client and server, or using remote procedure call (RPC) provided by the server for the client.



1 Information flows was introduced in the supplement part «Auxiliary Constructs» of UML.2.1.1.

Figure 2 : Client/Server communication flows.

In order to translate the UML/MARTE client/server description to an AADL description, AADL mechanisms (or concepts), such as AADL concurrent entities or AADL communication, have to be directly used during UML/MARTE modeling phases. Making explicit the semantics of such AADL mechanisms, with the UML/MARTE formalism, allows to make the designer aware of these semantics early in the development process. The following part introduces how to take into account (during UML/MARTE modeling phase) the AADL components relevant to the case study and how to make explicit their semantics using UML/MARTE (particularly thread execution semantics). The next part introduces the communications intended by the AADL standard and their corresponding UML/MARTE models.

4. AADL thread execution semantics model with UML/MARTE

AADL concurrent entities are designed with *thread* components. These components are bound to processor components and executed within the virtual address space of process components.

In this case study, architecture designers have then to specify which threads execute the client and the server. Moreover, they may have to specify the thread allocations on the processor(s). For example, in the case study, designers may choose to execute the client and the server each in its own thread, on separate processors. Without considering thread communications, figure 3 illustrates the AADL specification of the client/server case study with that configuration.

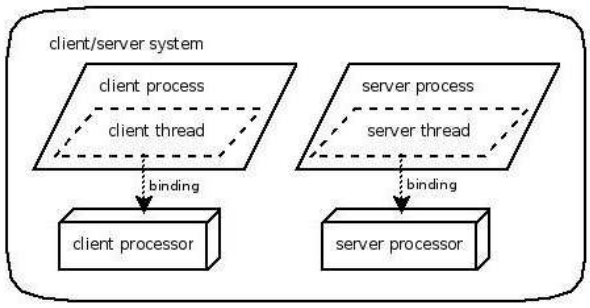


Figure 3 : AADL system to specify client/server case study (without communications)

Therefore, the UML/MARTE model of the client/server (in the case study) express the application according to the structure and the semantics of the AADL components involved. To do that, during UML/MARTE design phases, each UML/MARTE entity corresponding to an AADL component must follow the semantics of that component. For example, UML/MARTE concurrent entities semantics is regarded as AADL thread semantics.

Regarding the structure, UML/MARTE concurrent entities (corresponding to AADL threads) may be specified with the *swSchedulableResource* that is defined in the Software Resource Modeling package (SRM) [3]. In addition, to match the AADL components used in the case study with UML/MARTE entities, in a UML/MARTE description, the process is stereotyped with the SRM *memoryPartition* stereotype and the processor is stereotyped with both *hwProcessor* and *Scheduler* stereotypes. The *hwProcessor* stereotype, provided by the Hardware Resource Modeling package (HRM), allows to design the hardware processor. The *Scheduler* stereotype, provided by the Generic Resource Modeling package (GRM), allows to design the software that is responsible for scheduling and executing threads. At last, the AADL system may correspond to a SysML *block*. Figure 4 illustrates the AADL platform used for the case study and expressed with UML/MARTE.

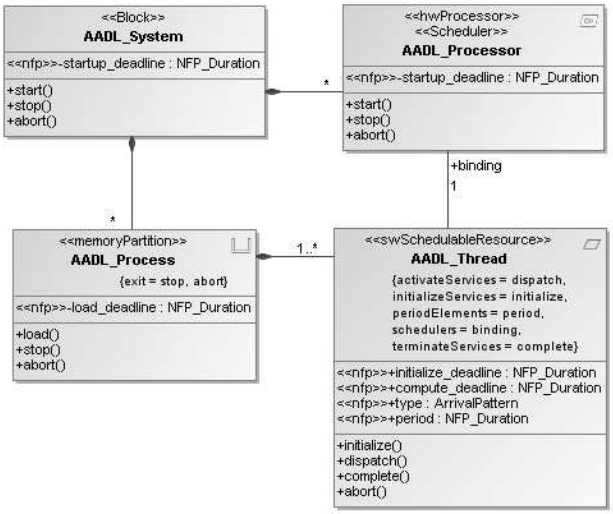


Figure 4: UML/MARTE description of the AADL components for the client/server case study.

To design the application, we can either use the AADL platform description (cf. figure 5) as a profile (creating and using stereotypes such as «AADL_Thread», etc.), or use directly the MARTE profile. Figures 5 and 12 illustrate examples of the latter approach with the client/server case study expressed using UML/MARTE according to the AADL concepts.

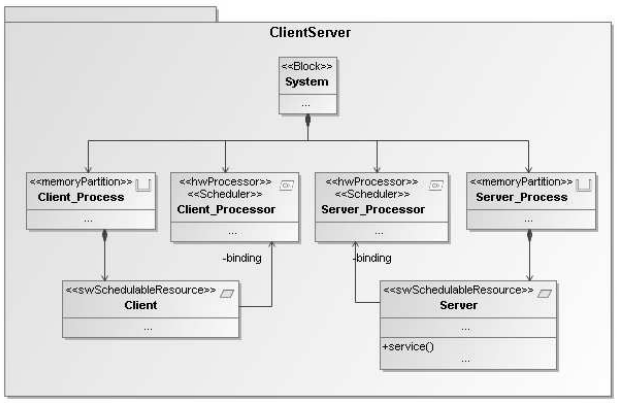


Figure 5 : UML/MARTE system to specify the client/server case study (without communication) according to the AADL approach.

Regarding the semantics, within the AADL standard, discrete and temporal semantics are described by hybrid automata notation. This notation consists of hierarchical finite state machine notation, augmented with real-valued variables to denote timing values. Using this notation, application startup that involve system, processors, processes and thread components is specified as well as thread execution semantics.

The application startup specified in the AADL standard (with system, processor, process and thread «states and actions» finite state machine) may be explicitly represented with a UML/MARTE sequence diagram (as described in figure 6).

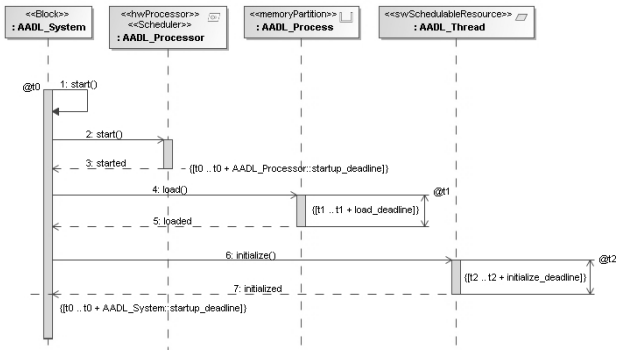


Figure 6 : Application startup.

This diagram specifies the system startup that results from processor startup, process loading and thread initialization sequence according to the AADL standard. Moreover, each of the system, processor and process «states and actions» finite state machines within the AADL standard can be translated in a corresponding UML/MARTE state machine.

However, AADL thread semantics describes more complex mechanisms than AADL system, processor or process components. It specifies the states, the dispatch (activation), the scheduling, the execution, the mode transition and the execution fault handling for thread components.

Without considering modes and fault handling, a UML/MARTE state machine (as described in figure 7) can make explicit part of the initialization and dispatch «states and actions» of threads conform to the AADL standard.

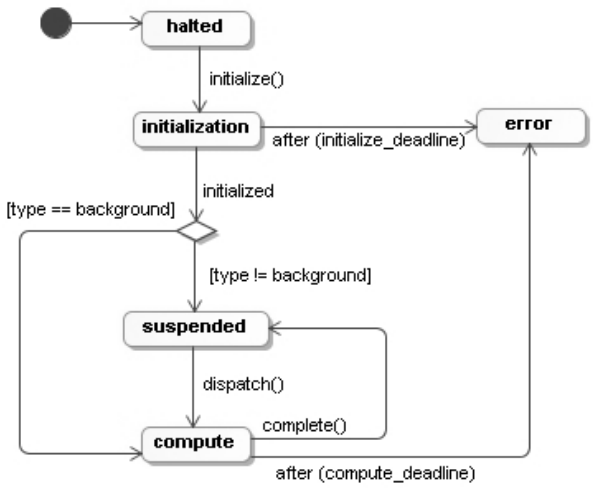


Figure 7 : Thread initialization and dispatch (without modes and fault handling).

Moreover, thread dispatch according to the dispatch protocol and thread scheduling (within the «compute» state) may be depicted using UML sequence diagrams, UML state machine and action languages. For this purpose, the runtime has to be specified either with an improved UML profile that is dedicated to operating system (OS) modeling, or by adding services to the relevant entities in the application model. We can notice that these services are out of the scope of the MARTE profile, because MARTE intends to model real-time embedded applications, not to model internal runtime mechanisms required for executing the application. For example, the MARTE *Scheduler* stereotype does not provide «resume» and «preempt» services (for the execution or the preemption of threads) because these services must not be available for the application. Thus, following the MARTE approach, specifying runtime within a separate model is a better solution to make explicit the AADL execution mechanisms, than specifying runtime services within the application model.

5. AADL communication semantics model with UML/MARTE

The AADL language allows different communication types through messages, notifications, shared data and services. Each communication type is represented either by a specific AADL concept (subprogram) or port type (event, event_data, data) linked to a specific run-time behavior. Domain application semantics must be in line with the run-time execution semantics, to make sense. Moreover, to make the validation/verification process at the model level more relevant, run-time specific behaviors need to be taken into account.

Different ways to use UML/MARTE to explicit AADL communication semantics will be illustrated on the client/server case study (presented section 3), during a refinement process.

Although proposing some abstraction mechanism - through abstract components and undefined ports - to promote high level design, the AADL language is very close to the execution platform: AADL application components are described in terms of AADL threads, processes, and critical regions, exchanging information through ports. This representation merges application design information and run-time resource management. This shortcut greatly simplifies the view for a well-established and finalized design, enabling the designer to focus on analysis and non-functional properties, but will be a handicap for high-level design. UML/MARTE allows separating the different concerns like application design, software platform modeling and hardware platform.

From a structural viewpoint, UML composite structures enriched with MARTE Message/Flow Ports typed by UML Signals, or interfaces (<<*bFeatureSpecification*>> stereotyped interfaces) are sufficient to distinguish AADL message/signal exchange from service request mechanisms (as illustrated Figure 8 and 9).

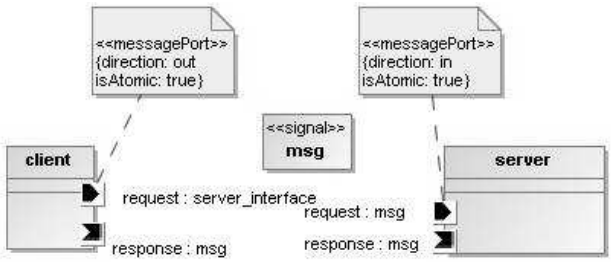


Figure 8: Event exchange through ports

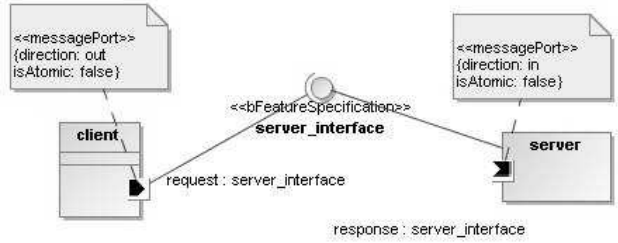


Figure 9: AADL subprogram call as interface requested services

Different implicit AADL behaviors and semantics are associated to these ports. So, AADL event data ports support a request/reply communication paradigm, while AADL event ports remain for notification. Different UML message types (synchronous messages, calls and replies will

capture these behavioral differences) as illustrated in the sequence diagram Figure 10 and 11.

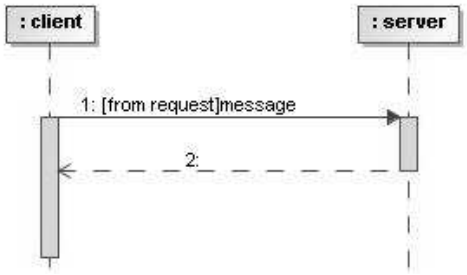


Figure 10: UML call and reply mechanism

Figure 11: UML asynchronous call

Many AADL components and properties rely on software platform modeling. UML/MARTE allows to explicit the resource platform models, and furthermore the relationships between this resource platform and the application using MARTE allocation and resource concepts.

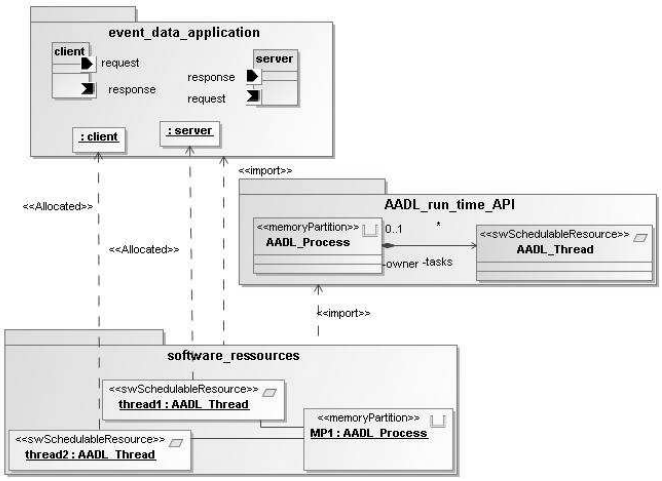
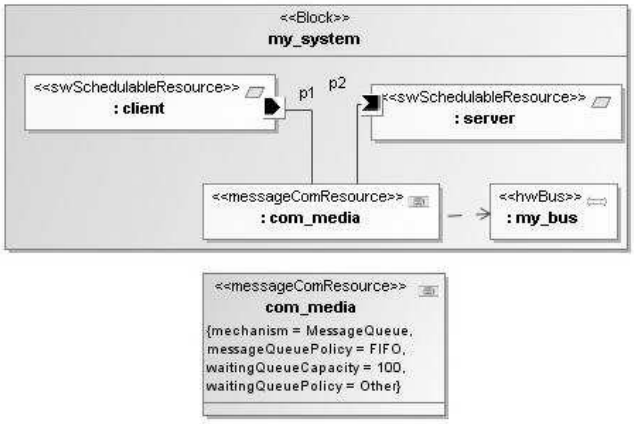


Figure 12: Application to software platform allocation.

Each component or component behavior specified at the application design level will be executed by one or more threads, represented in UML/MARTE as *swSchedulableResource* as presented in section 4. MARTE's flexibility allows a fine-grain application-to-platform allocation as pictured Figure 12.

As one of MARTE's aims is to provide a generic language for software and hardware resource description and run-time API characterization, MARTE's resource concepts integrate the most important resource and resource management features.

So, AADL messages sent by event data ports are represented by MARTE *MessageComResources*, as illustrated in figure 13, spooled in a queue. A part of AADL resource properties like *Queue_Size, Queue_Processing_Protocol, Overflow_Handling_Protocol, Dequeued_Protocol* properties will be provided intrinsically by MARTE: the *waiting_queued_policy* tag stays for AADL *Overflow_Handling_Protocol*, *message_queue_policy* for AADL *Queue_Processing_Protocol*, and *waiting_queue_capacity* for the AADL *Queue_Size*.



6. Conclusion

In this article a pragmatic approach for translating UML/MARTE detailed design into AADL design has been presented. We extend previous works in order to undertake part of the behavioral semantics translation. Rather than investigating the numerous possible UML/MARTE designs and their possible translations in AADL, we choose to map AADL concepts into UML/MARTE. It appears that some AADL concepts could not be defined in MARTE (for instance, port message queue policies). They will be proposed in the next MARTE version, enhancing MARTE – AADL concept alignment.

Therefore, it implies that methodological guides for designers have to be defined so that their UML/MARTE designs conform to elements which could be easily translated.

This guidance of the designer must include: UML/MARTE pattern modeling compliant with AADL, explicit representation of the AADL execution semantics and translation process divided in steps. This article presents part of this guide.

Therefore an automatic translation could be envisaged. A prototype is currently under development at Thales.

Nevertheless, this translation will be partial. We have furthermore to investigate the other parts of the behavior and the translations of non-functional properties. Indeed, UML/MARTE allows to describe non-functional properties such as temporal constraints for which AADL properties have to be translated.

7. Acknowledgement

This work is partially funded by the Topcased project [11] of the French "Aeronautic Valley" cluster.

8. References

[1] Object Management Group, "*UML Profile for Modeling and Analysis of Real-Time and Embedded systems*" (MARTE), Beta 1, OMG Adopted Specification, Document: ptc/07-08-04.

[2] S. Gérard, J. Médina, D. Petriu: "*MARTE: A New Standard for Modeling and Analysis of Real-Time and Embedded Systems*", ECRTS, Italy, 2007.

[3] M. Faugere, T. Bourbeau, R. Simone, S. Gerard, "*MARTE: Also an UML Profile for Modeling AADL Applications",* 12th IEEE International Conference on Engineering Complex Computer Systems, 2007.

[4] The OMG MARTE home page: http://www.omgmarte.org

[5] SAE Standard, "*Architecture Analysis and Design Language*" (AADL), document AS5506, June 2006.

[6] SAE AADL information site: http://www.aadl.info

[7] John Hudak, Peter Feiler, "*Developing AADL Models for Control Systems: A Practitioner's Guide*",Technical Report, CMU/SEI 2007-TR-014, 2007

[8] Peter H. Feiler, David P. Gluch, John J. Hudak, "*The Architecture Analysis & Design Language (AADL): An Introduction*", Technical Note CMU/SEI 2006-TN-011, 2006

[9] Object Management Group, "*Unified Modeling Language: Superstructure*", version 2.1.1, OMG Adopted Specification, formal/2007-02-03.

[10] F. Thomas, S. Gérard, J. Delatour , F. Terrier, "*Software Real-Time Resource Modeling*", FDL, Barcelona, 2007.

[11] Toolkit in OPen source for Critical Applications and SystEms Development (TOPCASED) : http://www.topcased.org

[12] M. Brun, J. Delatour, Y. Trinquet, "*Code generation from AADL to a real-time operating system: an experimentation feedback on the use of model transformation*", 3rd IEEE International UML&AADL workshop, Belfast, Northern Ireland, 2008.

[13] T. Vergnaud, I. Hamid, K. Barbaria, E. Najm, L. Pautet, S. Vignes. "*Modeling and generating tailored distribution middleware for embedded real-time systems*". 2nd European Congress Embedded Real-Time Software (ERTS'06), Toulouse, France, 2006.

9. Glossary

AADL: Architecture Analysis & Design Language

GRM: Generic Resource Modeling

HRM: Hardware Resource Modeling

MARTE: Modeling and Analysis of Real-Time and Embedded Systems

OMG: Object Management Group

RPC: Remote Procedure Call

SAE: Society of Automotive Engineers

SRM: Software Resource Modeling

SysML: Systems Modeling Language

UML: Unified Modeling Language