
Model driven engineering method for SAIA architecture design

Julien DeAntoni — Jean-Philippe Babau

*CITI laboratory – INSA-Lyon
21, avenue Jean Capelle, 69621 Villeurbanne cedex
julien.deantoni ; jean-philippe.babau@insa-lyon.fr*

ABSTRACT. SAIA is an architectural style for the development of systems dedicated to process control. Designing an architecture that conforms to a style implies the manipulation of a lot of entities and the respect of numerous constraints. The approaches based on models and models transformations are well adapted to manage the complexity and to enforce the separation of concerns. This paper presents a model driven engineering method for the development and the validation of systems that conform to SAIA. Moreover, a tool supporting the method allows a systematic use of models and transformations.

RÉSUMÉ. SAIA est un style architectural destiné au développement de systèmes dédiés au contrôle de procédés. Construire une architecture conforme à un style architectural donné nécessite la manipulation de nombreuses entités et le respect de nombreuses contraintes. Les approches basées sur les modèles et les transformations de modèles permettent de gérer cette complexité et d'imposer une séparation des préoccupations. Notre objectif est alors de fournir une méthode basée sur les concepts définis par l'ingénierie dirigée par les modèles pour le développement et la validation de systèmes conformes à SAIA. Enfin, un outil implémente la méthode proposée afin d'aider le concepteur à respecter les modèles et leurs transformations.

KEYWORDS: MDE, real time, development method, architectural style, model validation, modeling tool.

MOTS-CLÉS: IDM, temps réel, méthode de conception, style architectural, validation de modèles, outil de modélisation.

1. Introduction

Over the last few years, engineers have been faced with the problem of developing more and more complex real time systems in a world where time-to-market constraints are constantly increasing. Among real time systems, Systems Dedicated to Process Control (SDPC) are systems closely linked to the physical environment.

A SDPC builds, in real time, a view of the controlled process by using sensors (Cf. figure 1). Then it computes the necessary commands in order to control the process. Finally, commands are performed using actuators. Since the external environment has its own behaviors, all operations must be performed respecting real time constraints such as a minimum frequency or a maximum delay.

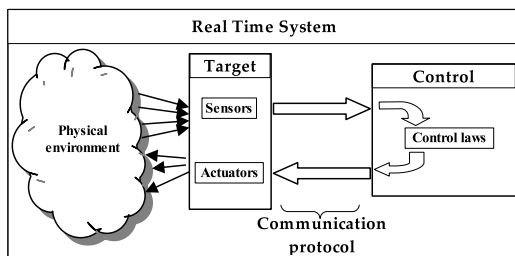


Figure 1. Representation of a SDPC

For these reasons, the SDPC communication part is developed in an ad-hoc and hand-made way. Of course, this type of development leads to concern merged code and specifications. In consequence, a software engineering approach has to be used to precisely specify the communication protocol; in a functional way as well as in an extra-functional way (Jumel, 2003). Among the various software engineering sub-domain, the software architecture gives propositions about the development and the early validation of software (Shaw *et al.*, 1996, Ghezzi *et al.*, 2000, Bass *et al.*, 1998).

The study of software architecture has highlighted that redundant configuration are used to solve classes of problems. These specific structures are identified as architectural style. SAIA¹ (DeAntoni *et al.*, 2005b) is an architectural style that has been defined for the development of SDPC. Its goal is to facilitate the development, the evolution and the timing analysis of the communication protocol part.

Building an architecture that conforms to a style implies the manipulation of a lot of entities and constraints. It severely increases the engineers' works and its complexity. So, to efficiently build and validate an architecture, methods and tools are necessary. As highlight by MDA (Model Driven Architecture : (OMG-MDA, 2003)), dealing with models and models transformations is a way to properly apply separation

1. SAIA : Sensor/Actuator Independent Architecture

of concerns, and so, to reduce complexity. The aim of this paper is to propose a model driven engineering method to support the conception of system with SAIA.

First, the section 2 presents an overview of SAIA. The next section describes the different models and models transformations implied during the conception of the system. Then, the section 4 presents a tool that implements the method. After, we present the related work and we give some conclusions and perspectives.

2. SAIA architectural style

2.1. SAIA concepts

Sensors Actuators Independent Architecture is an architectural style which defines entities and entity-associations for the modeling of SDPC. From a software engineering point of view, its goal is to separate the concerns between target, communication protocol and control (see figure 1). From a practical point of view, it allows a development of SDPC independently of specific sensors actuators; a realistic simulator-based simulation (DeAntoni *et al.*, 2005b) and an easy timing analysis (DeAntoni *et al.*, 2005a).

SAIA is a layered architecture. It distinguishes three layers through three models: one for the target, one for the communication protocol and one for the control. It allows the evolution of the control or the target without impacting each other. Each layer is specified thanks to components whose behaviors are accessible through interfaces.

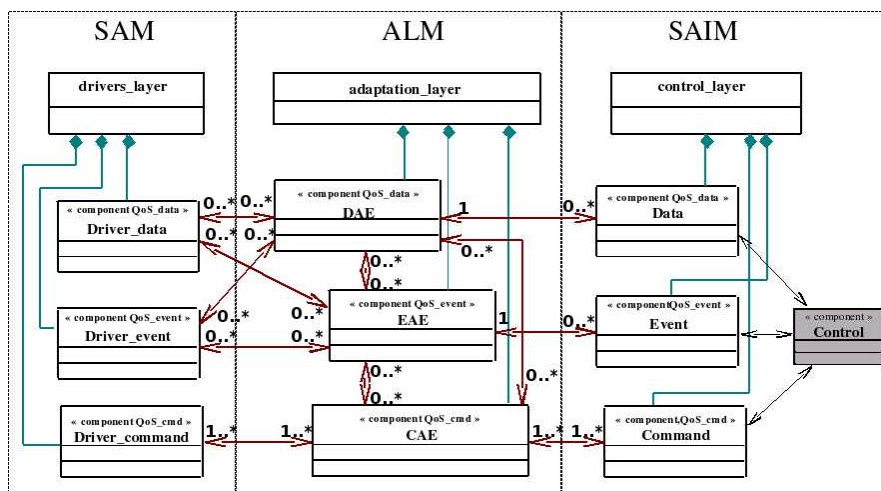


Figure 2. SAIA layered organization

2.2. Control layer

In the control layer, the control activity is based on a description of an ideal environment view called Input/Output (I/O). It models the state of the environment through data, events and the requirements on environment through commands. Each I/O describes the required QoS needed for the correctness of the control. There are three QoS characteristics : the arrival law, the delay and the precision (precision is not specified for an event). SAIA focuses on the interaction of a system with the external environment. Consequently, the control laws are seen in SAIA like black boxes. All these pieces of information are described in the SAIM (Sensors/Actuators Independent Model).

2.3. Driver layer

The driver layer is described by a model called SAM (Sensors/Actuators Model). It specifies the sensors and actuators services available on a specific platform. In the same way than the SAIM, there is a distinction between three entity types: driver_data, driver_event and driver_command. The two first entity types encapsulate a sensor driver whereas driver_command encapsulates an actuator driver. It provides a platform-specific view of the environment. Sensors and actuators drivers describe the QoS provided by the underlying hardware with the same QoS characteristics than the SAIM. To formally evaluate the provided QoS by sensors and actuators drivers is not a trivial activity but there are works on (Robles *et al.*, 1999, Ben-Hedia *et al.*, 2005). SAIA considers that the QoS provided is a priori known.

2.4. Adaptation layer

The adaptation layer, described by the ALM (Adaptation Layer Model) realizes the "smart" link (functional and extra-functional) between the ideal environment view of the SAIM and the platform specific environment view of the SAM. To do that, the adaptation layer is composed of three elements (Data Adaptation Element, Event Adaptation Element, Command Adaptation Element), each dedicated to a specific I/O type. Three² internal activities has been identified to specify an ALM element:

- `format`: it is used to cast driver_data or command value in order to manipulate them consistently. It consists in unity change and/or reference mark change.
- `interpret`: it encapsulates the human knowledge specifying how high abstraction level inputs are produced from low abstraction level driver information or how high abstraction level command are performed by low abstraction level driver actions. This activity owns a dynamic and functional³ behavior. The dynamic behavior repre-

2. expect for event where format() is not applicable

3. In SAIA, the functional behavior possess a temporal budget called WCET: Worst Case Execution Time

sents the reactive part of the interpretation (way to collect and produce information) whereas the functional behavior represents the transformation part of the interpretation (way to compute pieces of information together: average, max, ...).

- `qosAdapt`: it specifies a modification/adaptation of the QoS. This activity is mainly represented by the dynamic behaviors (creation of a constant delay, ...) but can also embedded a functional behavior for complex computation (i.e. interpolation).

The association of the three previous layers (SAM + ALM + SAIM) produces a representation of the whole system called SASM (Sensors Actuators Specific Model).

2.5. SAIA and the QoS

Since SAIA goal is the verification of the QoS conformance during conception, no online QoS information is needed. Thus, SAIA specifies the QoS in a lightweight way, like in (L.DiPippo *et al.*, 1999). The QoS characteristics are specified by interval [Min ; Max]. Once specified, SAIA mixes two different approaches for QoS consideration. In one hand, the SAIM specifies the required QoS and the SAM specifies the provided QoS. This results in a contract for the QoS in the sense of the fourth contract level described in (Beugnard *et al.*, 1999) for component contract specification. On the other hand, the QoS provided by the ALM is computed from the QoS provided by the SAM and the adaptation element behaviors. It is evaluated by the analysis of a specific configuration.

Now, the concepts and terms used by SAIA have been introduced. The next section describes the proposed method by beginning with the SAIM modeling.

3. SAIA MDE method

In SAIA the control activity is developed independently of the sensors actuators. In consequence, the development method starts with the modeling of the SAIM and its validation. Then, the SAM and the ALM modeling are presented. To finish, the models and the models transformations used for the validation are detailed. It is important to notice that all models and models transformations are done during the conception of the system. No transformations are realized on the deployed system.

3.1. SAIM development and validation

To begin, it is necessary to produce the SAIM from the specifications. The first stage is the determination of the I/O needed by the control. It is important to notice that the I/O represent data, event or command which are independent of any sensor or actuator technology. The I/O identification is extracted from the specification according to the functionality needed by the control activity (labeled 1 on figure 3). After the I/O identification, it is still impossible to describe the QoS required because it highly

depends on the control activity. Therefore the second stage is the control activity modeling (labeled 2 on figure 3). The control model must be based on the I/O description; the other information is extracted from the specifications. The modeling of the SDPC control activity is not treated here because SAIA sees it as a black box. The following stage (labeled 3 on the figure 3) extracted the non-functional constraints from the specifications. These constraints reflect the QoS objectives for the application in term of deadline, power consumption and user level quality of service (i.e. no deviation in the trajectory, etc). These constraints are a goal to reach and the constraints derivation explicitates that the QoS objectives are met as long as the I/O QoS required is respected (Torngren, 1998). This models transformation produces the QoS required for each I/O (labeled 4 on the figure 3). Now, the I/O functional model, the QoS required by the I/O and the control model can be mapped to produce a single model: the SAIM (labeled 5 on figure 3).

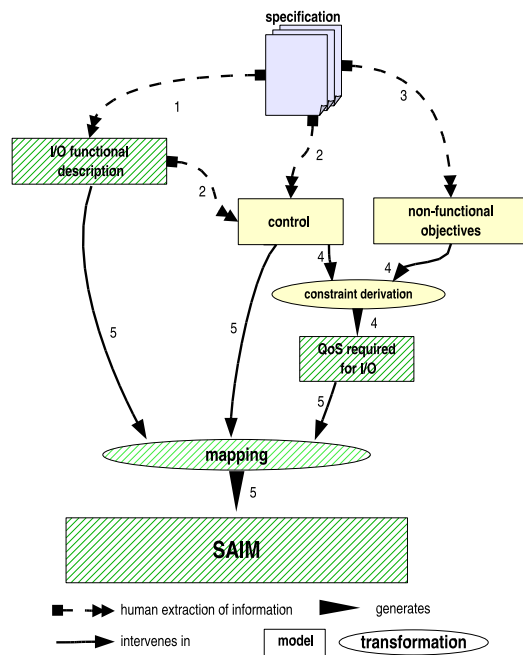


Figure 3. *SAIM development process*

3.2. SAM modeling

The second stage in the system development is to choose and model the sensors actuators of a specific platform. The output of this stage is the SAM. The SAM specifies the platform driver types and their provided QoS. Once the SAM modeled, we must

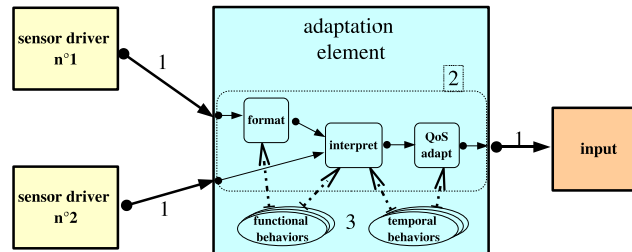


Figure 4. ALM development process

link the SAM with the I/O of the SAIM. It must be done by respecting the functional as well as extra-functional constraints.

3.3. ALM modeling

Since the ALM is in charge of the complex link between the SAM and the SAIM, its modeling specifies the mapping between these two models. The ALM modeling consists in three main stages. The first one (labeled 1 on the figure 4) introduces the structuring knowledge reflecting the drivers involved in the construction/accomplishment of the I/O. In other words, connectors are created between the drivers interfaces and the associated adaptation elements interfaces as well as between the adaptation elements interfaces and the associated I/O interfaces. This stage requires pieces of information from both the SAM and the SAIM (labeled 1 on the figure 5). The two other stages consist in the modeling of the internal part of each adaptation elements. In the second stage (labeled 2 on the figure 4), a structuring of the internal entities must be done (i.e. Is format useful? Is QoS adaptation useful and where, before or after the interpretation? ...). This structuring is mainly dictated by the domain knowledge and experiences. Then, in the last stage (labeled 3 on the figure 4), for each internal activity of the adaptation elements the dynamic and functional behaviors have to be specified. It gives information on the elements behaviors as described in the section 2.4 (interpretation type, functions WCET, and so on).

Once the ALM modeled, a mapping of the three models (SAM SAIM and ALM) generates the SASM (labeled 2 on the figure 5), i.e. a Sensors Actuators Specific Model of the whole system. At this stage, the SASM is functionally complete but the QoS provided by the ALM is not known. The remainder of the method is dedicated to the completion and the validation of this model.

3.4. System validation

The system validation is decomposed into two main stages. The first one consists in the evaluation of the QoS provided by the ALM. The second stage consists in a

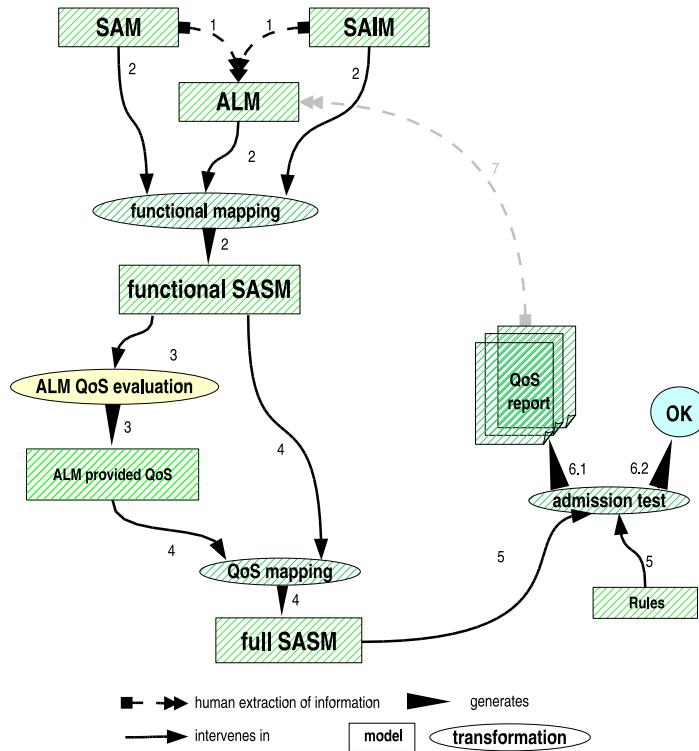


Figure 5. SASM models and transformation for validation

comparison between the QoS provided by the ALM and the QoS required by the SAIM. This second stage is called admission test.

3.4.1. ALM provided QoS evaluation

To produce a model of the ALM provided QoS, the following information must be available for all adaptation element in the ALM:

- the QoS provided by the sensor drivers (for Input production) or by the Output (for actuator driver),
- the adaptation element internal structure,
- the adaptation element activity temporal behaviors.

Now, all these pieces of information must be analyzed and computed to produce the adaptation element provided QoS. This step has to be done for each adaptation element in order to produce the ALM provided QoS (labeled 3 on the figure 5). When finished, the ALM provided QoS model is mapped with the SASM to generate a full

SASM; i.e a SASM including functional and extra-functional information (labeled 4 on the figure 5). The determination of its validity is in charge of the admission test.

3.4.2. Admission test

To be valid, the provided QoS characteristics of the ALM must satisfy the required QoS characteristics of the SAIM. In SAIA, the QoS characteristics are specified thanks to intervals [Min ; Max]. Consequently, the admission test must verify these rules:

$$\{\forall(QoScharacteristic) \in ALM\} \subseteq \{(associated\ QoScharacteristic) \in SAIM\}$$

Therefore, for each QoS characteristics:

$$[Min_{provided}; Max_{provided}] \subseteq [Min_{required}; Max_{required}]$$

The admission test transformation can generate two outputs depending on its success or its failure. If the admission test fails (labeled 6.1 on the figure 5), a QoS report is generated to identify where the rules violation have occurred. This report can be examined and the QoS adaptation of the responsible element(s) has to be done or changed (labeled 7 on the figure 5) and a new validation process must be done. If the admission test succeeds (labeled 6.2 on the figure 5), the validation process ends.

4. The SAIA method implementation

In the previous section, all the models and models transformations for a SAIA MDE development and validation have been presented. In order to implement the proposed method, a SAIA development environment has been developed. The generation of this specific modeling environment is based on the Model Integrated Computing (Sztipanovits *et al.*, 1997). MIC uses meta-models to explicit the rules governing the modeling of valid systems. MIC is implemented by the Generic Modeling Environment (GME (ISIS, 2005)), a meta-programmable toolkit for the creation of domain-specific modeling environments. The meta-modeling paradigm employed in GME is based on the UML.

Consequently, SAIA has been meta-modeled in GME meta-modeling paradigm. After the meta-modeling of the SAIA entities in GME, presentation information must be added to the meta-model for the generation of the SAIA modeling tool. This information describes various views where entities are viewable or not according to the model (see 1: in the figure 6) and the place in this model. Since each model is described in a different view, in a specific model, only the entities belonging to this model are usable (see 2 in the figure 6). In addition of the different views, it allows creating a depth hierarchy. For instance, an entity such as the `interpret` activity is only accessible when "entering" in an adaptation element (see 4 in the figure 6). This way, it separates the concerns between the different models but also between the structure of the system (see 3 in the figure 6), the internal structure of components (see 4

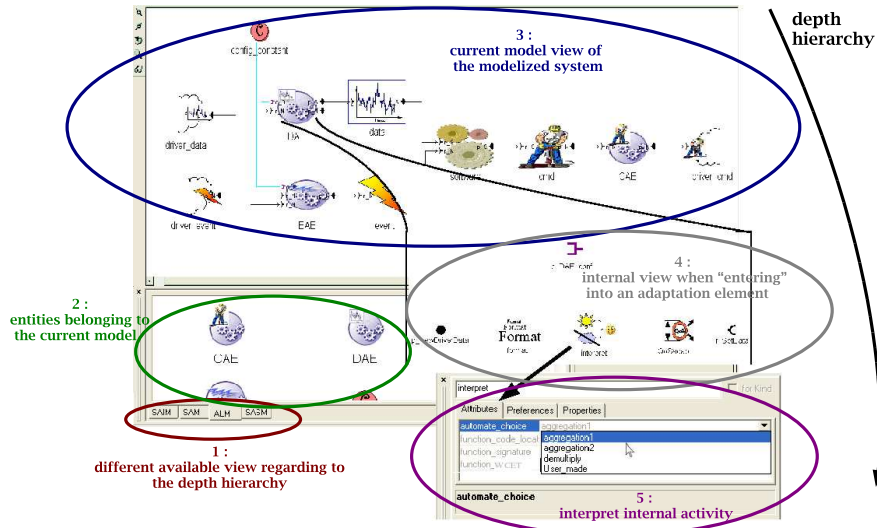


Figure 6. a view of the tool and its hierarchy

in the figure 6) and the behaviors of component internal activities (see 5 in the figure 6). A screenshot of the tool and its different views are given figure 6.

Now, it is possible to use GME with SAIA paradigm, i.e. it is possible to create most of models described by the method. On the figure 3 and 5, all the models and transformations represented by a hatched box are supported by the tool. The other models and models transformations includes the modeling of the control and the constraints derivation transformation. For SDPC, the control model and the constraint derivation can be implemented by the MATLAB tool suite (Mathworks Inc, 2005a, Mathworks Inc, 2005c, Mathworks Inc, 2005b).

The use of SAIA has highlighted the use of redundant interpretation and QoS adaptation reactive part. Consequently, a toolbox of predefined automata for `interpret` and `qosAdapt` activities is provided for SAIA. The interpretation, as well as the QoS adaptation, modifies the QoS provided by the SAM. To facilitate the evaluation of the new provided QoS, all automata in the toolbox are associated with equations. These equations characterize the QoS provided in output of the automaton according to the QoS provided in input and the automaton characteristics (DeAntoni *et al.*, 2005a).

This way, if the modeler uses only predefined automata, the transformation called: "ALM QoS evaluation" on the figure 5 can be done by applying the equations to the QoS characteristics provided in input of the automaton. On the contrary, if the modeler chooses user-made automata, analytical equations are not available and the validation is more complex. In this case, the tool acts as an input generator for analysis tools allowing the evaluation of the ALM provided QoS.

One solution has been implemented by the tool. It is based on timed automata modeling and analysis. The chosen language for the timed automata modeling is IF (Bozga *et al.*, 2002). IF is composed by a formal timed automata language and an associated tool. It allows the evaluation of some QoS characteristics (Ben-Hedia *et al.*, 2005). Consequently, the SASM described in the SAIA modeling tool is translated into an IF model. For the first stage of this translation, we use the faculty of GME to export models in an XML format whose the DTD is a representation of the modeling paradigm used. In the second stage of the translation, a parser retrieves the necessary information from the XML file. Then it generates the appropriate IF model from this information. Now, the analysis can be performed thanks to the analysis tools suite.

After these steps, the QoS provided by each adaptation element and accordingly by the adaptation layer is known and can be added to the SASM model in the tool. The admission test must now verify that the QoS provided by the adaptation layer satisfies the QoS required by the SAIM.

The rules governing the admission test are the ones described in the section 3.4. This admission test is implemented in the tool thanks to OCL constraints. The evaluation of these constraints can be done in the SAIA modeling tool by asking for constraints evaluation. If the test fails, for each constraint violations the tool gives the adaptation element and the QoS characteristic where the constraint is violated.

This tool has been used with success in the realization of an exploration robot (DeAntoni *et al.*, 2005c). This exploration robot must conform to the specification dictated by the maRTian Task challenge. After the modeling of the robot, the source code must be generated. This code generation needs information about the Real Time Operating System (available scheduling policy, etc) where the system is executed. It also needs mapping rules specifying how the components and their activities are mapped into RTOS tasks. Since the code generation goal was only to verify that all the mandatory pieces of information are included in the full SASM model, this knowledge has been brought manually.

The method advantages are highlight by the fact that the exploration robot development relies on a simulator. The SAM in this case represents the services provided by the simulator. Since the SAM is separated from the SAIM by the ALM, during the deployment on a real target, the changes are not propagated in the whole system. The changes impact the SAM and the ALM but the SAIM stays unchanged.

5. Related work

In SAIA and the proposed method, a clean separation has been done between the control activity and the sensors actuators provided by a specific target. This approach is to assimilate to the MDA (Model Driven Architecture: (OMG-MDA, 2003)) philosophy where a Platform Independent Model (PIM), a Platform Model (PM) and a Platform Specific Model (PSM) are defined. The proposed method is an implementation of the MDA philosophy where the platform is identified to the Sensors and Actuators.

MDA is successfully used for the development of general purpose systems (Frankel *et al.*, 2002, D.Wampler, 2003, P.Caceres *et al.*, 2003) but there are few MDA-based implementations for dedicated system where correctness strongly relies on hardware performance.

(P.Boulet *et al.*, 2003) proposes an implementation of MDA for systems on chip design. This MDA adaptation of the "Y-approach" introduces an association model in order to model the mapping between software and hardware models. This approach considers machines and buses as the PM but not sensors and actuators dependencies. Moreover, contrary to the proposed approach, no extra-functional descriptions are made. However, the two approaches use the same idea which is the use of an intermediate model to link the platform specific and platform independent models.

In the software architecture domain, lots of different works and strategies have already been proposed for the development and the validation of systems. One approach is specifically closed to the one described in this paper: the approach developed for the REACT project (Faucou *et al.*, 2004).

The REACT project is based on the architecture description language CLARA (Durand, 1998), dedicated to the description of reactive systems architecture. The first stage of the project concerns the validation of a candidate architecture regarding its functional and extra-functional requirements. The similar point with the proposed approach is the use of external formalisms and tools to validate the systems. However, conversely to the proposed approach, the CLARA description does not identify various models and models transformations to conduct the architecture building.

Sensors / actuators communication is seldom dealt with but, (Cristina *et al.*, 2005) proposes a component based package for modeling of sensors and actuators in an OSGI context: SensorBean. The approach identifies some services which are also identified in SAIA. Since it is a recent approach, there is no detailed information about extra-functional requirements, specification, and validation.

6. Conclusion

This paper presents a model driven methodology for the development and validation of SDPC communication protocol. Various models and model transformations are identified. It facilitates the management of the SAIA entities and constraints during development and validation of systems. The generation of a tool supporting the method allows a systematic used of the models and transformations. Moreover, it has allowed the validation of the approach through an international challenge: maRTian task (DeAntoni *et al.*, 2005c).

Several orientations for the future works can be interesting. First, for an use in an industrial context, a SPEM (OMG-SPEM, 2005) modelisation of the proposed process could facilitate evolution and reuse through systems families. Another interesting orientation is to allow the creation of a bridge between the SAIA modeling tool and

UML modeling/simulation tools; i.e. the creation of a SAIA UML profile. This way a transformation from the SAIA modeling environment to a UML editor could be implemented. Next orientation is the formalization of the transformation between SAIA and IF models at a meta-model level. Finally, SAIA could be extended in order to allow an automatic code generation for RTOS.

7. References

- Bass L., Clements P., Kazman R., « Software Architecture in Practice », *Addison Wesley, Reading, Mass*, 1998.
- Ben-Hedia B., Jumel F., Babau J.-P., « Formal evaluation of Quality of Service for data acquisition systems », *Forum on specification and Design Language*, 2005.
- Beugnard A., Jézéquel J.-M., Plouzeau N., Watkins D., « Making component contract aware », *IEEE computer*, 32(7)p. 38-45, 1999.
- Bozga M., Graf S., Mounier L., « IF-2.0: A validation environment for Component-Based Real Time systems », *In ed. Brinksma, K.G. Larsen (Eds) Proceedings of CAV'02*, 2002.
- Cristina M., Donsez D., Lalanda P., « Approche IDM pour le développement des services basés capteurs », *Ingénierie Dirigée par les modèles (IDM'05)*, 2005.
- DeAntoni J., Babau J.-P., « A MDA approach for systems dedicated to process control », *eleventh IEEE International Embedded and Real Time Computing Systems and Applications (RTCSA'05)*, 2005a.
- DeAntoni J., Babau J.-P., « A MDA-based approach for real time embedded systems simulation », *Nineth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05)*, 2005b.
- DeAntoni J., Babau J.-P., « SAIA: Sensors/actuators Independent Architecture - A showcase through maRTian task specification », *Proceedings of the ERTSI 2005 - Embedded Real Time Systems Implementation Workshop, held in conjunction with 26th IEEE International Real-Time Systems Symposium*. 43-50, 2005c. <http://www.cs.york.ac.uk/ftpdireports/YCS-2005-397.pdf>.
- Durand E., « Description et vérification d'architectures d'applications temps réel: CLARA et les réseaux de Petri temporels », *PhD thesis, école centrale de Nantes*, 1998.
- D.Wampler, « The Role of Aspect-Oriented Programming in OMG's Model-Driven Architecture », *Aspect Programming Inc*, 2003.
- Faucou S., Déplanche A.-M., Trinquet Y., « An ADL centric approach for the formal design of real time systems », *In Architecture description language, IFIP*. 67-82, 2004.
- Frankel D., J.Parodi, « Using Model-Driven Architecture(tm) to Develop Web Services », *IONA Technologies PLC, White paper*, 2002.
- Ghezzi C., Jazayeri M., Mandrioli D., « Software Architecture: a Roadmap », *In A. Finkelstein editeur, International Conference on Software Engineering, ACM press*, 2000.
- ISIS, « The Generic Modeling Environment (GME) », 2005. ISIS: Institute for Software Integrated Systems. Vanderbilt University.
- Jumel F., « Definition and management of a quality of service for real time applications (in french) », *thesis in LORIA laboratory, Nancy*, 2003.

- L.DiPippo, L.Ma, « A UML Package for Specifying Real-Time Objects », *Computer Standards and Interfaces 2000*, 1999.
- Mathworks Inc, « MATLAB », <http://www.mathworks.com/products/matlab/>, 2005a.
- Mathworks Inc, « real time workshop », <http://www.mathworks.com/products/rtw/>, 2005b.
- Mathworks Inc, « SIMULINK », <http://www.mathworks.com/products/simulink/>, 2005c.
- OMG-MDA, « Model Driven Architecture guide V1.0.1 », <http://www.omg.org/mda>, 2003.
- OMG-SPEM, « Software Process Engineering Metamodel Specification Version 1.1 », <http://www.omg.org/docs/formal/05-01-06.pdf>, 2005.
- P.Boulet, J.L.Dekeyser, C.Dumoulin, P.Marquet, « MDA for SoC Embedded Systems Design, Intensive Signal Processing Experiment », *FDL03*, 2003.
- P.Caceres, E.Marcos, B.Vela, « A MDA-Based Approach for Web Information System Development », *wisme*, 2003.
- Robles E., Held J., « A comparison of Windows driver model latency performance on Windows NT and Windows 98 », *Proc. OSDI third symposium*, 1999.
- Shaw M., Garlan D., « Software architecture: Perspectives on an emerging discipline », *Prentice Hall*, 1996.
- Sztipanovits J., Karsai G., « Model-Integrated Computing », *IEEE Computer*, vol. 30, no. 4, pp. 110-112, 1997.
- Torngren M., « Fundamentals of implementing Real-Time Control applications in distributed computer systems », *Real Time System*, 1998.