RsaML : A Domain Specific Modeling Language for describing Robotic software architectures with integration of real-time properties.

Valery M. Monthe
Faculty of Sciences
University of Yaounde 1
Yaounde, Cameroon
valery.monthe@gmail.com

Laurent Nana
Lab-STICC / UMR 6285
University of Brest
Brest, France
nana@univ-brest.fr

Georges E. Kouamou
National Advance School of Engineering
Yaounde, Cameroon
georges.kouamou@polytechnique.cm

Claude Tangha
PUCA, Yaounde-Cameroon
ctangha@gmail.com

ABSTRACT

This paper deals with the problem of expression and representation of robotics software architectures, at a level of abstraction high enough, and independent of the implementation platform, taking into account real-time properties. It also addresses the problem of standard representation, communication between domain experts, and therefore that of reusability of these architectures. It presents RsaML (Robotic Software Architecture Modeling Language), a Domain Specific Modeling Language (DSML) for robotics software architectures that we proposed in order to solve the problems mentioned above. The conceptual model defining the terminology, and the hierarchy of concepts used for the description and representation of robotic architectures in RsaML are presented in this paper. RsaML is defined through a meta-model which represents the abstract syntax of the language. Real-time properties of robotics software architectures are identified and included in the meta model. The use of RsaML is illustrated through the definition of a robotic system and the description of its architecture with the help of the language. The support tool used for this work is the Eclipse Modeling Framework (EMF).

CCS Concepts

•**Computer systems organization** → **Robotic control;** *Real-time system specification;* •**Software and its engineering** → **Domain specific languages; Embedded software;** *Real-time systems software;*

Keywords

robotics software architecture, domain specific modeling language, MDE, real-time and embedded system, Eclipse EMF

EWiLi'16, October 6th, 2016, Pittsburgh, USA.

1. INTRODUCTION

Robotic systems are real-time systems whose behavior, of type "perception, decision, action", is founded on information extracted from the environment with the help of sensors (for example images captured by a camera). The information provided by the sensors should be processed within a bounded time interval in order to provide a new command to the robot, and this, before the capture of a new data.

Robotic systems are generally composed of two parts: the physical part, and the software part that allows the robot to operate in a complex, dynamic and unstructured environment, and that affects its behavior [3]. The software part has an architecture that is considered as the software architecture of the robotic system.

The development of robotic systems has always been a big challenge because of heterogeneous technological aspects involved. The development of the software part of these systems is even more complex, for several reasons, including:

- The lack of a standard representation of robotic architectures;
- The inadequacy of generalist languages for the representation of concepts which are specific to the field of robotics;
- The lack of a language for the description of these architectures. Such a language is needed to facilitate communication and exchanges between members of the community, and therefore to increase the reusability;
- The lack of tools to rapidly build these architectures and to ensure early validation of their design.

One of the challenges of robotics today is to use well-known and proven software engineering techniques such as Model Driven Engineering (MDE), that are proven in the development of traditional software, to solve these problems. MDE provides the ability to define a domain specific modeling language (graphical or textual), and to use it to build models (architectures) and generate code from these models.

This paper presents RsaML, a Domain Specific Modeling Language (DSML) for robotics software architectures, which

is a solution to problems mentioned above. In the second section, we present an overview of robotic software architectures. The third section is dedicated to related work. The fourth section deals with the RsaML language that we proposed. The fifth section presents the experimentation of RsaML on a sample robotic system. A discussion is presented in the sixth section. The paper ends by conclusions and future works in the seventh section.

2. OVERVIEW OF ROBOTIC SOFTWARE ARCHITECTURES

Robotic software architectures can be classified into four categories based on how they structure the elements and operations: (i)traditional centralized architectures, (ii) hierarchical architectures, (iii) behavioral or reactive architectures and (iv) hybrid architectures. We describe briefly below these different categories.

2.1 Centralized architectures

The first work on robotic control architectures enters in this category, and were inspired from artificial intelligence, that is, organized around decision-making and a symbolic state of the world and the robot[5][7]. These architectures place planning at the system center and share the axiom that the central problem in robotics is cognition, that is to say, the manipulation of symbols to maintain and act on a model of the world. The world is the environment with which the robot interacts.

2.2 Hierarchical architectures

The organizational model of hierarchical architectures, also called deliberative architectures, center the design over the decisional system[5][7]. These architectures are organized in most of the proposals in several layers (also called Levels) hierarchy[1][6][11]. A layer directly communicates only with the immediatly lower layer and the next higher layer. It breaks down a task that was recommended to it by the higher level, into more simple tasks for the lower layer. The highest level manages the overall objectives of the application, while the lowest controls the robot's actuators level[7]. These architectures typically have three layers: functional (contains the perception of modules), Executive (in charge of the supervision of robot tasks), decisional(in charge of planning).

2.3 Behavioral or reactive architectures

In a reactive architecture several modules connect the sensor inputs to the actuators. Each module implements a behavior, i.e a basic functionality of the robot associating each input vector (the set of sensor values) to an output vector applied to the actuators. These behaviors are called "reactive" because they immediately provide an output value from a value in this entry. Inspired by observing behavior of animals, these architectures are built according to the idea that, a more mature behavior can emerge from a combination of a set of simple basic behaviors.

2.4 Hybrid architectures

Hybrid architectures combine the reactive capacity of behavioral architectures and reasoning skills (decisional) of hierarchical architectures [5][7]. These architectures include a hierarchy of layers, and reactive nested loops allowing each layer to provide tailored responses to its dynamics.

As mentioned earlier, one of the challenges of robotics today is to use well-known and proven software engineering techniques. In the next section, we present an overview of MDE, a software engineering technique, that we have chosen in this perspective. MDE has proven its efficiency in the development of traditional software systems.

3. RELATED WORK

More and more solutions for the use of software engineering techniques in robotics are proposed at different levels. But these works are very recent and are not numerous. In this section, we present a few.

In [8], Passama proposes a solution to the problem of representation and communication of Robotic software architectures between domain experts. He proposes a specific modeling language for the domain. This with the aim to express and easily compare different architectures.

Xavier Blanc and al. in [2] address the benefits brought by the MDE (Model Driven Engineering) approach to the development of Embedded systems and robotics. They apply this approach to develop a software system for Aibo, which is a type of robot.

Authors of [12] address the issue of model driven engineering applied to the design of a service robot controller. They develop a scalable modeling approach for the development of real-time control software of a prototype of seven-axis arm actuated by artificial muscles.

In [10], authors also address the issue of MDE process for robotic systems. They are particularly concerned with taking into account of non-functional properties in modeling. Properties such as quality of service, management of real-time resources (eg the schedulability analysis of real-time tasks).

The work proposed in [13] applies a MDE approach to design specific domain solutions for the development of robot control system, using subsumption architecture. It presents a case study of the entire process: identification of domain meta model, definition of graphic notation and code generation.

The authors of [3, 4] formulate the problem of the development of stable software systems in the robotics field, analyze the elements that make this problem difficult, and identify challenges that robotics community must face in order to build stable software systems.

In [9], authors present the contributions of model driven approach compared to code driven approach in the development of robotic software systems.

The works presented above deal with the application of the MDE approach in the field of robotics software architectures. The work [8] which is devoted to the definition of a language has some failure in relation to the problems that we have laid in section 1: it does not represent all categories of architectures; For example, it does not take into account some specific aspects of behavioral architectures, such as inhibition of outputs and deleting entries. Some properties relating to the functioning of the robot software architectures can not be defined solely by the structural relationships between concepts. Such constraints must be verified by other means, for example using the OCL language. Some of the works presented, such as [9, 10] deal with the MDE in robotics in general, without providing a solution to the specific problem of representation and communication of robotic software architectures. The work of [13] is specific

to the subsumption architecture. The study conducted by [2] concerns a specific case of robotic system, and therefore the meta model it proposes is limited to Aibo system. [12] is limited to the development of software for controlling a given prototype. Almost all these works do not address the real-time side of these systems.

4. DEFINITION OF THE DOMAIN SPECIFIC MODELING LANGUAGE PROPOSED

In this section we present our solution to the problem of representation and communication of robotic software architecture between experts. The section first presents some challenges and describes the concepts identified for the language. The graphical representation and constraints of the language (meta model) are then discussed. This section concludes with the real-time aspects to manage in the language.

4.1 Objectives and challenges

The development of a domain specific modeling language begins with the identification of modeling concepts. This requires to understand the functioning of the field, the notions which are used there, how to use them and the links between them. In our case we relied on reference documents (original publications) of each class of architectures, and publications in the field. Different concepts are used in each category. Thus the main challenges are: (i) define concepts which allow to represent any robotic software architecture; (ii) define the types of these concepts in the model, the relationship between them and especially cardinality, which allow to respect the structural and functional aspects of these architectures; (iii) be concise, but accurate and complete, to avoid proposing a model dense and therefore complex, which may be difficult to analyze and understand for end users of the language; (iv) the opportunity to define the real-time properties in the models that will be built.

4.2 Domain analysis : The main modeling concepts

A careful study of each robotics software architecture class was made. This initial work has resulted in a list of concepts per architecture. Thereafter, it was question of integration for a single list, where we would find a concept to represent each concept used in each class of architectures. Finally, we define a set of concepts, which are used in the different types and examples of architectures; and which may enable to represent any of these architectures. For a better understanding and ease of use of concepts in the language, we defined a hierarchy of these. The diagram in figure 1 shows the hierarchy of the different concepts identified.

In the diagram of figure 1, a robotic system consists of a robot, the environment in which it operates and the software that controls it. A robot has a central part which we call the "body of the robot" and a set of hardware components. The software allows it to evolve in this environment (set of physical and logical resources). The hardware components can be either sensors, actuators, or any other devices for communicating and evolving (observation, perception, reaction) in its environment. The software consists of a set of decision-making components (layer, activity, database, knowledge, etc.) and behavioral components (action, function, module, modifier, goal, etc.). There are two types of ports: input ports and output ports. In some architectures such as subsumption, the port data can be modified by those of another. Thus, an input port can carry multiple suppressors and an output port several inhibitors.

Once identified, each concept was described, based on the concept of robotics it models. Table 1 shows in alphabetical order some of these descriptions.

Table 1: Some concepts used

Concepts	Descriptions
Action	basic task of the robot;
Actuator	allows the robot to act on the environment
Activity	task or sequence of tasks of the robot
Database	database, knowledge base, etc.
Sensor	allows the robot to detect its environment
Order	order sent to the robot to perform a task
Knowledge	data handled and how it is used
Layer	level in the hierarchy of the architecture
Function	set of functions implementing a module
Inhibitor	allows to inhibit some outputs
Mission	established set of goals, tasks and paths
Module	any robotic software component
Plan	sequence of actions to perform a mission
Supervisor	controls plans execution
Suppressor	allows to remove or edit entries

4.3 Graphical representation: the meta model

Once the concepts identified, the next step was to formalize a model, which shows the way in which these concepts are put in relationship to define an architecture. Thus, from the list of concepts proposed in section 4.2, we identified those of them, that are used in robotics software architecture, to propose a meta model for the representation of these architectures. It is shown in figure 2.

The diagram in figure 2 can be summarized as follows: a robotic control software can be composed of: a database containing a set of knowledge, several layers, and must have one or more goals to achieve. The objective can be broken down into sub targets reached by performing a set of activities. These activities follow a given strategy. The activities are carried out by execution of processes that are implemented in modules. The modules that are on the layers may have several functions that implement processing. The modules communicate through ports. Each function is a set of actions that can be simple or contain real-time constraints (real-time action). An action can be followed sequentially by another or run in parallel to several others.

Once the meta model designed, it was necessary to define a set of constraints (building rules) to control the handling of the metal model concepts. We have defined a set of constraints using OCL. They are presented in section 4.4.

4.4 Definiton of constraints on the meta model

Several constraints cannot be defined by current meta-modeling languages using only graphical elements. Thus we defined a set of constraints for verifying and validating architectures (models) that will be built using the language(meta-model). OCL is employed to define these constraints as so-called well-formedness rules. In the following, we introduce some examples for well-formedness rules in natural language and subsequently show the corresponding OCL invariants.

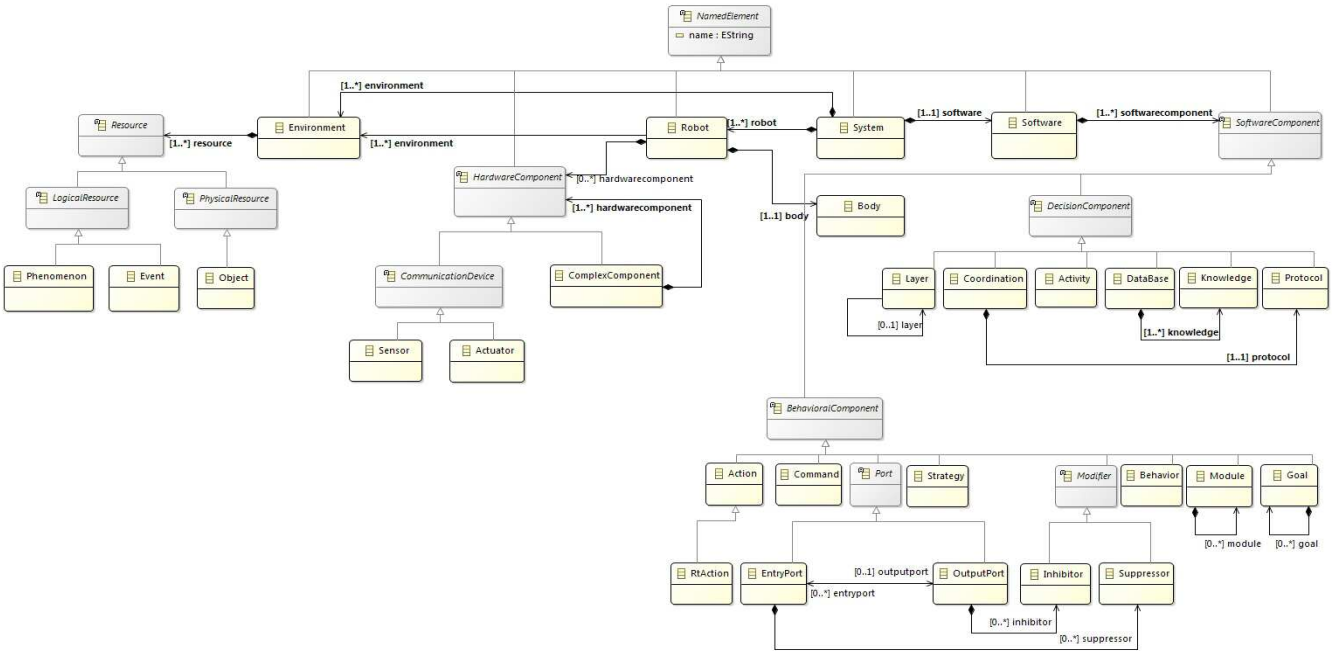

Figure 1: Hierarchy of concepts used in robotic systems

−−1. Two modules that communicate must share the same type of data on their communication interfaces.
context EntryPort
inv : self .dataType=outputport.dataType
context OutPutPort
inv : self .dataType=entryport.dataType
−−2. A module can inhibit (resp remove) the outputs (resp entries) module of immediately lower layers.
context OutPutPort
inv : self .control−>forAll(m|m.port.module.layer.number=self.module.layer.number−1)

Listing 1: some OCL constraints imposed on the domain' s meta-model

4.5 Real-time properties support

The integration of real-time properties was made at different levels in the definition of the meta-model. First of all in the structure of the meta-model, by setting attributes for specifying these real-time properties; next, through the definition and adding of constraints to the meta model. Table 2 describes some of these properties. These constraints are shown in listing 2. Note that these constraints are defined after laying a few assumptions. We assume that: (i) the system response time, switching time and the generation time of the mission plan are all zero; (ii) every action has a limited and known time; (iii) any time other than execution one is assumed to be zero.

−−1. If the type of module is real−time, its period and delay must be not null.
context Module
inv : self .type=ElementType::realTime **implies** (delay <>0 **and** period<>0)

Table 2: the real-time properties in meta model

Attributes	Descriptions	Concepts
MaxExecTime	Maximum execution time	layer
type	type of the element (simple or real-time)	module, action, function
period	period of execution	- - - - -
delay	response time	- - - - -
execDuration	execution duration	- - - - -
startDate	start time	action
timeUnit	time unit	- - - - -
execType	sporadic, periodic or not	- - - - -
synchroType	synchronous or not	function

−−2. A module is real−time if all its functions are real −time
context Module
inv : self .type=ElementType::realTime **implies** (function−>forAll(f|f.type=ElementType::realTime))
−−3. Runtimes of all modules of a layer are bounded by the maximum execution time of the layer.
context Layer
inv : module−>forAll(m|m.execDuration<=self.maxExecTime)

Listing 2: OCL constraints for management of real-time properties

4.6 Implementation : metamodeling in eclipse

4.6.1 The meta model

After designing the meta model, it was implemented in Eclipse, thanks to its Framework EMF. Once the meta model

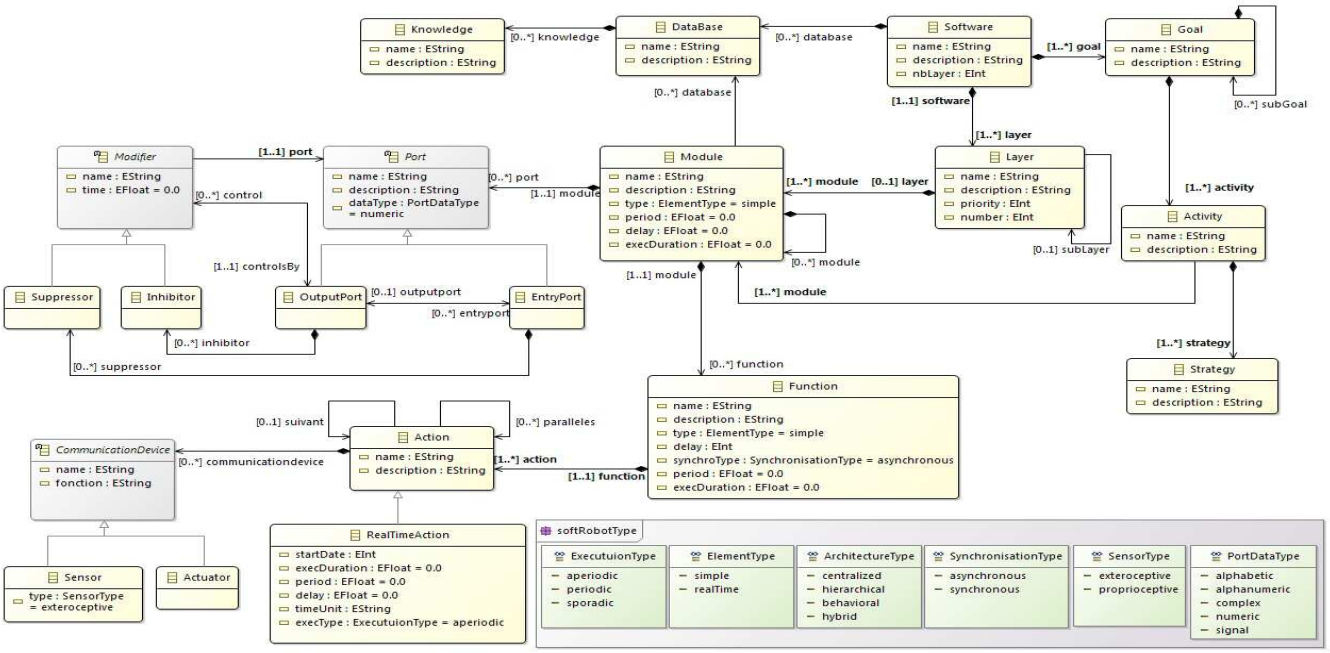

Figure 2: Proposed meta model for the definition of robotics software architectures

built, we validated it by verifying that all the syntactic rules of construction were satisfied (for example that we did not forget to specify the type of an attribute).

4.6.2 *Integration of OCL constraints in the meta-model*

For defining OCL constraints for our meta model, we use Eclipse OCL, a project focusing on the implementation of the OCL standard within Eclipse. Before integrating them to the meta model, we verified that each of our OCL constraints was syntactically correct, and provided the expected result. For this, we used the evaluation interactive console for OCL expressions, which is a tool of Eclipse OCL project. Figure 3 shows an example of that evaluation. The result is ***False*** in this case, because two modules (m12,m12) have the same name.

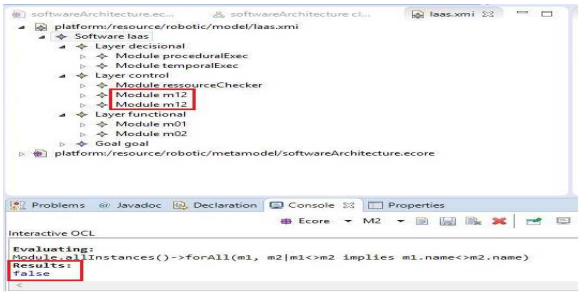

Figure 3: Evaluation of OCL constraints

5. EXPERIMENTATIONS

We consider a very simple robotic system. It is a system wherein the real time executive is represented by a single processor. The latter is connected to a sensor and an actuator, by means of two grabbers. The software system that will control the robot is shown above the executive as shown in the diagram in figure 4.

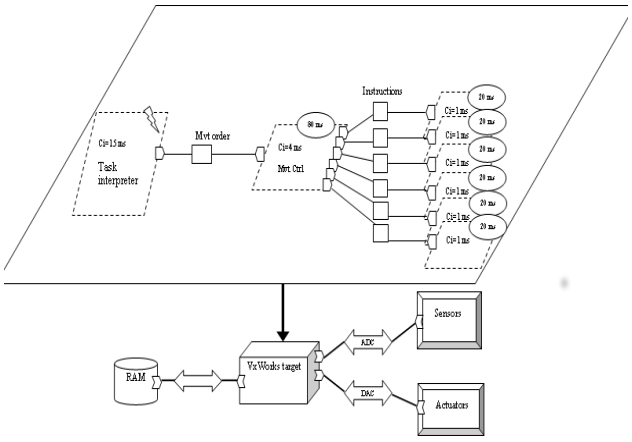

Figure 4: An example of robotic system architecture[12]

We can represent a software system using the language proposed by completing the following steps: Sensors and actuators can be represented by corresponding sensor and actuators concepts of our language, an action by an action, a task by a function, the runtime system on which the real-time controller will be executed by a Processor, other software components (acquisition of sensor data , processing of those data, etc.) by modules, the real-time controller with a module, the connections between components by links between their respective ports, the memory by database and the whole system by the Software concept. We built this software system using the tree editor of Ecore, as shows the figure 5.

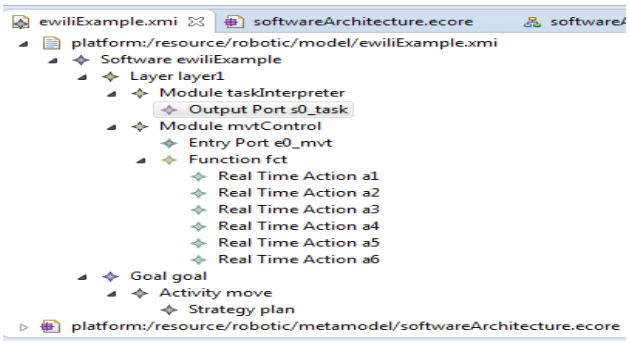

Figure 5: Example represented using the language in the tree editor of Ecore

6. DISCUSSION

The works presented in section 3, have each in its own way, opened a path to the application of software engineering techniques (especially MDE) in the field of robotics. Nevertheless, as mentioned in section 3, they have some limitations with respect to the issues discussed here, and the objectives of this paper. This work has provided answers to most of the shortcomings of the previous works:

- It has included the modifier concept (inhibitor and suppressor), to handle the case of some behavioral architectures (e.g. subsumption);

- It has integrated OCL constraints, thus enabling to verify properties specific to the area of robotics, that can not be managed solely by the structural relationships between concepts in the meta model;

- It does not address a particular category or a particular architecture, but offers a language that takes into account all categories;

- RsaML was designed, disregarding the robot (hardware) that will be handled by the software architecture modeled;

- It incorporates elements for specifying real-time properties of the modeled system.

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed a DSML for describing robotic software architectures. Robotic systems are special cases of embedded real-time systems. An analysis of the modeling domain was made by identifying the needs of these systems and studying the different categories of their software architectures. This analysis ended with the identification and description of useful concepts for the representation of these architectures. A hierarchy between these concepts was defined. Useful attributes to take into account real-time properties have been identified, defined and integrated into the relevant concepts. A meta model was proposed to describe robotic software architectures. OCL constraints have been defined and included in the meta model to check some semantic rules specific to the field of robotics and check out some real-time properties, not definable by the only model structure. The EMF Framework was used to implement the language. We finished by an experimentation of language on an example of robotic system.

The proposed language allows to describe robotic software architectures, that take into account some real-time properties. To better meet the needs that we expressed at the beginning of this paper, we plan in our future works, to: (i) provide a graphical editor to facilitate the representation of architecture and that would be easier to use than the tree-based editor currently used; (ii) design a transformation engine, to provide a complete chain from modeling to code generation; (iii) the opportunity to transform the models built, in other models (eg AADL), in which the verification of real-time properties can be realized.

8. REFERENCES

[1] J. S. Albus, H. G. McCain, and R. Lumia. Nasa/nbs standard reference model for telerobot control system architecture (nasrem). 1989.

[2] X. Blanc, J. Delatour, and T. Ziadi. Benefits of the mde approach for the development of embedded and robotic systems. In *Proceedings of the 2nd National Workshop on "Control Architectures of Robots: from Models to Execution on Distributed Control Architectures", CAR*, 2007.

[3] D. Brugali and M. Reggiani. Software stability in the robotics domain: issues and challenges. In *IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.*, pages 585–591. IEEE, 2005.

[4] D. Brugali and P. Salvaneschi. Stable aspects in robot software development. *International Journal of Advanced Robotic Systems*, 3(1):017–022, 2006.

[5] A. El Jalaoui. *Gestion Contextuelle de Tâches pour le contrôle d'un véhicule sous-marin autonome*. PhD thesis, Université Montpellier II-Sciences et Techniques du Languedoc, 2007.

[6] E. Gat et al. On three-layer architectures. *Artificial intelligence and mobile robots*, 195:210, 1998.

[7] L. Nana. Architectures logicielles pour la robotique. In *Journees Nationales de la Recherche en Robotique*, pages 239–248, 2005.

[8] R. Passama. A modeling language for communicating architectural solutions in the domain of robot control. In *Proceedings of the 2nd National Workshop on "Control Architectures of Robots: from Models to Execution on Distributed Control Architectures", CAR*, pages 109–121, 2007.

[9] C. Schlegel, T. Haßler, A. Lotz, and A. Steck. Robotic software systems: From code-driven to model-driven designs. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–8. IEEE, 2009.

[10] A. Steck and C. Schlegel. Towards quality of service and resource aware robotic systems through model-driven software development. *arXiv preprint arXiv:1009.4877*, 2010.

[11] A. Tate, B. Drabble, and R. Kirby. *O-Plan2: an open architecture for command, planning and control*. University of Edinburgh, Artificial Intelligence Applications Institute, 1992.

[12] D. Thomas, C. Baron, and B. Tondu. Ingénierie dirigée par les modèles appliquée à la conception d'un contrôleur de robot de service.

[13] P. Trojanek. Model-driven engineering approach to design and implementation of robot control system. *arXiv preprint arXiv:1302.5085*, 2013.