Towards a Language for Understanding Architectural Choices in Robotics

Robin Passama, David Andreu, OBASCO (EMN-INRIA) & LIRMM (CNRS-University Montpellier 2)

Abstract— The paper presents a reflection on the state of the art in robot control architectures and evokes the main challenge for their design: the management of domain expertise. It shows the perspective of the production and the use of a Domain Specific Language dedicated to robots architects, to overcome the limitation of current software engineering techniques. The first step in this aim is to perform a domain analysis to define both terminology and concepts that can be understood by most of robot architects. To this end, we propose, as a first basis, a conceptual model, illustrated by two examples, for explaining robot control architecture design.

I. INTRODUCTION

Service robotics now becomes an industrial challenge, involving a great variety of expertise in automatics, informatics, telecommunications, electronics, mechanics and even “business” specific expertise (medicine, security, military, etc.). The first questions are: how can people involved in such projects understand themselves? How can they integrate their expertise in one global project in a coherent manner? Obviously, this cannot be done if nobody has the same common concepts and the same vocabulary to denote these concepts. It is also clear that such concepts would never been accepted if they limit the use of different paradigms. We have to admit that it is actually just a dream, if not an error, to try to convince all people to adopt a common conceptual frame. So, the spectrum of this frame has to be reduced and divided into more realistic ones. More precisely, it would be useful if at least each specific domain of expertise had its own conceptual frame.

We will show that, in software engineering, this question is already addressed in the field of Domain Specific Languages (DSLs). A DSL provides to an expert a language adapted to the vocabulary and concepts of its domain. With this aim, we reasonably have to wonder about the way expertise would be integrated in the architecture of a complex robotic system. We estimate that this would be very difficult if robotic architectures are not themselves designed with a common language that provides abstractions to describe the way others domains of expertise merge and interact. The definition of a preliminary conceptual model for robot control architectures is a first contribution to the domain analysis necessary for designing such a language. This model is sufficiently abstract and general to allow various robot control design approaches to be expressed with shared concepts and terms (as the terminology is as well of importance). Our intended goal is to propose a common basis, which could be improved and specified along researcher’s discussions.

Section 2 details the challenges posed by complex robotic projects and problems that emerge. The use of DSLs is given has a global direction to follow in order to overcome those problems. Section 3 briefly evokes the limitations of actual robot control architecture proposals. The conclusion of this section leads us to propose in section 4, a conceptual model for robot control architectures and examples in section 5. Finally section 6 draws up some concluding remarks and perspectives.

II. CHALLENGES, PROBLEMS AND DIRECTIONS

Currently, the global challenge in robotics development is to express very different human expertise into software pieces and to integrate all these pieces in a cohesive manner into common software architecture. This integration is not limited to expertise in robotic fields as control, navigation, vision, world modeling or artificial intelligence, but also concerns various “non-functional” domains as security, transactions, persistence and so on. Moreover, the emergence of service robotics will involve the necessity to express a large set of “business” domains, like medicine, human security, defense, etc. Identifying and managing each of these preoccupations independently already requires a high degree of expertise, but the emerging robotics/information industry will require managing them simultaneously for a given system.

Historically, software engineering aims at providing solutions for software analysis, design, development, deployment and testing. But generally these solutions are centered on programmer’s, software architect’s or software
project manager’s points of view, like for example in object [1], aspect [15], or component [27] paradigms. Even if software frameworks [14] encapsulate specific domain expertise, they only provide a solution usable by programmers, but not by most of domain experts. Consequently, we see two main problems in the use of well-known software engineering techniques: they don’t provide to domain experts an adequate “environment” to apply their expertise to a given problem nor to reuse their solutions; they don’t provide techniques at an adequate abstraction level to merge solutions from different domains of expertise into a global software system.

This is one of the intended goals of the Domain Specific Languages (DSLs) [20]. The main idea of this pragmatic approach is to provide high-level languages to domain experts to allow the description of solutions of domain problems. One advantage is that, thanks to the degree of abstraction of DSL, empirical or formal rules can be checked on models in order to validate or to verify various domain properties. Another property of DSLs is to allow for automatic code generation from solution models. This is related to language compilation, but with a greater abstraction gap between the source and the target models than for general purpose programming languages. To tackle the problem, the idea is, for a given DSL, to generate the domain model code on top of a specific software framework or API that contains the implementation of the domain model code on top of a specific software framework or API that contains the implementation of the DSL domain expertise and may also provide others functionalities like simulation, test, observation and control at run time. In fact, these ideas are not really new and some examples already exist in real application domains. The merit of the DSL researchers is to try to formalize and generalize the underlying methodologies, concepts and tools, a good example being ECLIPSE GMF [12]. One major methodological issue is a way to integrate solutions expressed in different DSLs. This is the aim of research projects at the confluence of active software engineering research fields like model driven engineering [7] (for model level integration) and aspect oriented programming [15] (for code level integration). The tendency is to define a “pivot” language, more or less close to an architecture description language [19], around which preoccupations (defined via DSLs) will merge. Unfortunately, the process itself is far from being formalized and stabilized. But it is certainly the more pragmatic and powerful approach that has been proposed to manage large system complexity. It matches the requirement of the development of complex robotic architectures, i.e. the management of the diversity of domain expertise.

In a DSL-based methodology, robotic projects should provide DSLs for domains of expertise involved in robotic architecture development like vision, control, world modeling, navigation, etc. Works done on specific frameworks, for instance in vision [29] or in navigation [28], could certainly be helpful. In order to obtain a dedicated methodology, robot architects should also propose a way for integrating all the involved domains, in other words a "pivot" language for robotic software integration. In a DSL design, the first step, which is mandatory before being able to define language syntax and semantics that matches domain terminology, is the domain analysis. It consists to define common concepts to understand and communicate domain expertise. It corresponds, in our context, to the definition of common concepts used by robot architects to explain architectural solutions in robotic systems.

III. CONTROL ARCHITECTURE LIMITS

Giving a common set of concepts and terms to explain and exchange architectural solutions is a really hard task, in regard to existing works in the domain of robot control architectures. The huge diversity of architectures proposed by the community has several origins. The first one is the coexistence of specific architectural schemas closed to a control design methodology [18]: the “reactive” approach like Subsumption architectures [8] and DAMN [23], the “deliberative” approach like NASREM [2] and 4D/RCS [3], the “hybrid” approach like Aura [5], ORCCCAD [25], CLARATy [30], LAAS architecture [4] and IDEA [21] or also the “behavior-based” approach [17].

This classification, presented in [18] has emerged along robotic history, but the distinction between these categories is sometime fuzzy. For example, the difference between reactive and behavioral-based approaches is really thin since it mainly relies on a criterion (“the behavior-based can store representations while reactive cannot”) that can be considered as subjective (in fact, only the life time of the representation differs). Another example is the difference between hybrid architectures like Aura on the one hand and LAAS or CLARATy on the other hand. In fact, Aura is hybrid in the sense that its reactive layer is organized around behaviors (like reactive or behavior-based approaches) and its decisional layer is organized around navigation and action planner entities. LAAS or CLARATy are more or less organized as deliberative architectures but with a greater uncoupling between decisional layer and functional layer activities (this is also subjective). So, the difference between Aura and LAAS/CLARATy approaches is important when considering the control design within the reactive layer: the former being based on a behavioral organization, the latter being rather based on a functional one. LIRMM [10] and ORCCCAD [25] architectures are viewed as hybrids because they have a layered style that allows for direct interactions between non adjacent layers (under given conditions), in order to improve reactivity. Our intuition is that a language for communicating architectural choices should provide general concepts that abstract from these subtle differences.
of designs.

The second origin of the diversity of those mentioned proposals is certainly the technology. This latter plays a great role in the diversity of terminologies and concepts. Many proposals are close to specific frameworks. For example CLARATy [22] is closed to a locomotion and navigation framework, LAAS [4] and ORCCAD [25] are close to a specific execution framework, and Chimera [26], OROCOS [24] and MirpaX [16] are close to communication frameworks. With so deep differences among proposals in their technological foundations, it is obviously difficult to denote recurrent (shared) concepts in all of them. Nearly all of these proposals limit the architect’s possibilities to a restricted set of concepts. One exception is the MARIE [11] framework, but unfortunately it is a "pure" technological proposal that does not provide a specific conceptual base. Our intuition is that the robot architects' language should be sufficiently general to allow the description of key design concepts without binding these concepts with underlying technology, implementation or even specific algorithms.

Consequently, we propose an informal conceptual model for understanding and communicating architectural choices, as the first step of domain analysis. The model must be both minimalist and generic; this is a big challenge in itself. As a first direction, we focus on the description, at a high degree of abstraction, of the distribution of decisional processes.

IV. A CONCEPTUAL MODEL FOR ROBOT ARCHITECTURES

The distribution of decisional processes within control architecture is often specific for each proposed control architecture design; all mentioned control architecture propositions have a different layout of these processes, a fortiori because they address different issues like control law selection, action planning, behavior arbitration, etc.. So, it is important to favor the allocation of any decisional process to any part of the control architecture. This allocation is, in most of cases, guided by the decomposition of architecture into different levels of abstraction of robot decision, namely the "layers". Concepts presented below, allow different distributions of decisional processes depending on architecture decomposition into layers, as the association decision process-layer is left open (not a priori imposed by the model). Depending on this distribution, control architectures nearly always impose specific coordination between decisional processes. As coordination plays a great role in the differentiation of control methodologies, the model impose to describe it, but not with a limited set of interaction types, to allow every solution to be expressed. One important thing to precisely qualify decision processes and their coordination is the knowledge that processes use and exchange. In this way, the model has to impose the description of knowledge and its relation with decisional processes. Moreover, we note that the decomposition of control architecture into coarse-grain decisional systems (or "agent" [21]) ease the design and the understanding of architectures. Each decisional system contains layers and the assignment of coordinated decisional processes to systems depends on “physical” properties like morphological or infrastructural properties of the robot. So, systems and layers are complementary concepts that should be both present in the model.

A. Main Concepts

First, we define four main concepts: Knowledge, Activity, Coordination, System. To this end, the term task is used, and it has to be understood in the general meaning, i.e. the fact of doing something.

Knowledge: a knowledge entity identifies a structured piece of information about the world within which the robot controller evolves. It can directly refer to the physical world (environment, robot body) or to a concept bound up to this physical world like a "phenomenon" or an "event". It can also refer to a know-how of the robot relative to this world, i.e. a way of detecting/solving problems relevant to this world (e.g. criteria defining singular configurations of the robot). Finally, the controller being itself part of the robot world, a knowledge entity can also explicitly refer to it, to get for instance a form of introspection.

Activity: an activity entity is responsible for the achievement of a task that plays a role in the robot decisional process, using a set of knowledge. For example, an activity entity can refer to the observation of the environment state, of the robot (body) state, or even of the controller state. It can also refer to: low-level control of the robot, like the application of a control law, medium-level control as control context commutation, and high-level control like planning. Finally, it can also refer to a learning activity (creating or refining knowledge) whatever the level of control is concerned.

Coordination: a coordination entity is responsible of the way a set of Activities interact by exchanging or sharing knowledge. For example, it can express collaboration, i.e. an interaction type defining how different tasks are distributed among Activities to achieve a more complex task. It can also express competition, i.e. an interaction type defining how several Activities achieve the same task.

Systems: a system entity is an abstraction of the control of a physical entity (morphologically distributed or not). It is responsible of the way a set of Activity and Coordination entities, concerning this physical entity, are organized in order to achieve a set of tasks. It also defines their relation with the controller's Infrastructure (the one being able to execute them). For example, a System entity can refer to the control of a robot’s part (e.g. the arm or the vehicle of a mobile robot), to the control of the (entire) robot (e.g. the mobile robot) or even to the control of a robot team.
The diagram of fig.1 shows how these concepts are related to each other. We can see that an activity uses internal knowledge entities. Coordination implies a set of activities with respect to the knowledge entities that are exchanged or shared among these activities. Readers can notice that an activity can be implied in more than one coordination. The diagram introduces two other entities: Layer and Infrastructure.

A layer is an abstraction that symbolizes a "level of decision" whatever the complexity of the decision is (from "simple" reactive decisions to "complex" deliberative decisions). A system organizes its internal activities by associating each of them to a layer according to its decision level in the System.

An infrastructure is an abstraction that represents whole or a part of the controller's physical part (processing nodes, communication links, etc.). A system is deployed on an unique infrastructure but an infrastructure can host many systems. We can see that infrastructures can be made of others infrastructures, in the same way as systems can be composed by a set of others systems. The constraint not expressed in the diagram is that a sub-system is necessarily deployed in a sub-infrastructure belonging to the infrastructure of its containing system.

Next sections will now enter in the details of each of the concepts that have been mentioned.

B. Knowledge

A knowledge entity is quite complex to detail without more specialization. At the highest level of abstraction, the only thing we can say is that it contains a model synthesizing the type of knowledge and data representing the parameterization of this model for a given context. By model, we mean any form of representation of the knowledge, being it mathematical formulas, empirical models, declarative models like CSP, biologically inspired models like neural networks, geometrical models, etc.

We think that knowledge entities should be specialized by means of more precise categories. In this way, the work done by Brugali and Salvaneschi in stable aspects of robot development [9] provides a good basis and we reuse their terminology here. So, knowledge entities can be arranged following three categories: Embodiment, Situatedness and Intelligence.

The Embodiment refers to the consciousness of having a body that allows the robot to experience the world directly. In this category, we find knowledge entities like that representing the Body or the Infrastructure of a robot, or some of their parts. These entities can be themselves associated with other knowledge entities of the Embodiment category, for instance a Body is associated to Morphologies and Kinodynamics [9].

The Situatedness refers to evolving in a complex, dynamic and unstructured environment that strongly affects the robot behavior. In this category, we find knowledge entities like that representing the Environment (or parts of it, in any dimension), the Interactions between the environment and the robot, the Phenomena that can occur in the environment. To describe these entities we need others ones like those representing Place (in the environment) and Time. Almost all of these entities are presented in [9], except Phenomenon. A Phenomenon refers to something that can occur in the environment, which is perceptible or estimable by the robot.

The Intelligence refers to the ability of the robot to adopt adequate and useful behaviors while interacting with the dynamic environment. In this category, we can find Behaviors and Actions. Briefly, an Action denotes a capability of the robot (or part of it) to act in order to obtain a given result (e.g. a given effect on the environment). The expected result can also be represented by an Objective (e.g. reaching a given place at a given time). Behavior denotes an abstract representation of the way the robot behaves when executing actions according to the possible robot-environment Interactions. Knowledge entities of this category can also represent Systems and Strategies associated to each System. A knowledge entity representing a Strategy contains an action planner and for each action defines the behavior(s) to be selected to reach its objective. Behavior is the result of the activation of a set of coordinated Activities (or a single one). So, to allow the precise description of Behaviors, knowledge entities can also represent the Activities and Coordination used to concretize the Strategy, to allow the robot to reason on. In this conceptual decomposition of Intelligence we are partially detached from [9] proposal, to put in adequacy our organization of knowledge entities with other main concepts of our model.

All these types of knowledge entities are denoting concepts sufficiently abstract for the description of a great amount of scientific areas. To this point, the model allows only to describe the "passive" characteristics of robot controller architecture, not the "active" ones.
C. Activity and Coordination

They are expressed thanks the Activity and the Coordination entities. An informal representation of those two types of entities is proposed in fig.2.

An activity is an entity of any level of decision that puts in place Perception-Decision-Reaction cycles. It receives Perceptions (required pieces of information, or significant phenomena notification) from other activities or from the infrastructure (including sensors). Each Perception of an activity is associated with one or more knowledge entities of any level of abstraction, from simple sensor data to complex computed robot or environment states. It contains a Decision mechanism that computes Reactions. This mechanism is of any level of abstraction, from simple control law computation to a high-level planning or supervisory control mechanism. The Decision mechanism handles activity internal knowledge and knowledge coming from Perceptions to determine the Reactions to adopt, which represent the way it wants its decision to be realized by (eventually) others Activities or by the infrastructure (including actuators). Each Reaction is associated with one or more knowledge entities of any level of abstraction, from simple actuator data to a high-level order (e.g. an Objective).

The Decision mechanism can be influenced by Intentions it receives. An Intention represents a goal that an activity intends to accomplish -i.e. a goal influencing its Decision. An Intention is associated to knowledge entities representing, for instance, the desired state of the robot Body, related or not to the Environment, or a desired Behavior. The Reaction emitted by an Activity can be viewed as an Intention by the activity that receives it.

The Decision mechanism can also emit Observations. An Observation represents an interesting state of: the Body, the Intelligence or the Environment. For instance, an activity that detects an obstacle in the environment can transmit the corresponding Observation associated with corresponding knowledge entities. An emitted Observation can be viewed as a Perception by activities that receive it.

Like for knowledge entities, we can categorize activities into more specific entities like for instance, Environment Observers, Body Observers, Body Controllers, Body Motion Planners, Body Action Supervisors, and so on. Intentions, Observations, Reactions and Perceptions are exchanged by activities thanks to coordination entities. A Coordination entity is an entity of any level of abstraction, that imposes a protocol for knowledge exchange between a set of activities. It can represent various interactions like simple notification channels, resource request on time interval as the one used in CLARATy, specific subsumption or inhibition links, as well as a complex vote protocol like in DAMN. In fact, it depends on the protocol and on the nature of Intentions, Observations, Perceptions and Reactions taken into account by the coordination.

We have to notice that Intentions, Observations, Perceptions and Reactions are optional features for both activity and coordination entities (even if using none of them makes no sense). Furthermore, we don't have any a priori on the way decision mechanisms or protocols are specified or implemented.

D. System

Systems being used to describe the control of a morphological part of the robot, they are in relation with Knowledge Body entities. These latter are useful to "reason" about the robot body while systems are useful to exploit it. First, a system describes the way its internal activities are deployed on the Infrastructure of the Robot Body. Systems contains a deploy property that define the loci of the Infrastructure (described by an Infrastructure Knowledge entity) where activity and coordination entities are executed. The loci have also to provide the access to sensors and actuators that are useful to activities. Second, each system has a control property that refers to a
of each layer (decisional, reactive, executive, functional, etc.) is left undetermined to allow a maximal flexibility.

Inside and across systems, activities are organized according to a layered approach, following the criterion presented in Fig.3. The number and the precise semantics of each layer (decisional, reactive, executive, functional, etc.) is left undetermined to allow a maximal flexibility.

V. EXAMPLES

This section presents the application of the conceptual model for two representative examples: Aura and CLARATy. These examples are defined according to our understanding of these two architectures, based on their related bibliography. This is an important precision: since we don't deeply know the software architectures, we could unintentionally not respect the author's initial viewpoints. So these examples should be seen as illustrations of our conceptual model rather than "definitive" opinions on the way these architectures are designed.

A. Aura

Aura is a generic and abstract control architecture which merges a reactive approach for low-level control design with a hierarchical "deliberative" approach for high-level control design. The example of the use of the conceptual model for describing the Aura "solution" mainly refers to our understanding of [5] and [6].

In Aura, the types of knowledge entities that are important to understand architectural choices are: Environment (1), Phenomenon (2), Interaction (3), Robot Body (4), Behavior (5), Action (6), Objective (7) and Strategy (8). Numbers in parenthesis are used to denote the knowledge entities handled by activity and coordination entities of the Aura architecture (cf. Fig. 4).

The decomposition of robot architecture into subsystems does not exist in Aura: each robot is associated to a single system. System decomposition arises as soon as a group of collaborative robots is considered, each system (i.e. robot) being deployed on its own infrastructure. Robot system architecture is decomposed into two layers. Each of these layers can, in turn, be conceptually decomposed into two (reactive) or three (deliberative) layers (not shown in Fig. 4 to reduce it).

At the top layer there is the Mission Planner activity, in charge of collecting user intentions (i.e. mission goals and constraints) and of defining long-term robot objectives. At the layer below, the Spatial Reasonner activity receives requests from the Mission Planner to define the path (sequence of spatial Objectives) the robot must or can follow to reach each Objective (7). Once a path is defined the Plan Sequencer activity is invoked to define the sequence of actions required to follow the path and to reach the global Objective. For example, if the global objective is to clean a building, the first action sequence would be: "go to room 1", "collect waste" and "put waste in the bin" if the path is "room1, room2, room3, etc.". The action sequence corresponds to a state diagram where states are actions to perform and transitions are action changes. Transitions are associated to specific stimuli (i.e. Phenomena (2)) or Interaction (3) that enable the state change. The activity entities of the deliberative layer interact around a Long Term Environment Memory Sharing coordination entity to exchange and update the Environment knowledge (1).

Once a sequence of actions has been defined, the Plan Sequencer invokes the Schema Controller activity of the reactive layer to realize each Action (6). This latter interacts
with Perception and Motor Schemas activities. Perception Schemas activities are responsible of the production of stimuli from sensors data.  

Stimuli are partial instantaneous representation of: the Environment (1), the Interactions of the robot (3) or the Phenomena (2) that occur. Perception Schemas can also produce long term Environment (1) representations (e.g. map of a room). Motor Schema activities put in place specific Behaviors (5). Each Motor Schema activity is viewed as a control low, computed using Environment (1) stimuli and Robot Body knowledge (4), to obtain a given Behavior (5), as for example “obstacle avoidance". Schema Controller activity coordinates Perception and Motor Schemas in different ways. First, it translates the Action execution request into a composition of Schemas (cf. Schemas). Motor Schemas are activated according to the Behavior they represent and can be configured (if needed) with the given action Objective; Environment stimuli generated by Perception Schemas are redirected to Motors Schemas following a predefined Strategy (8). Second, some Perception Schemas are activated to generate Interaction or Phenomena stimuli (cf. Stimulation Notification). Schema Controller uses these stimuli to know if the Action has been (or cannot be) realized. It can then reply to the Plan Sequence to indicate if the Action succeeded or failed; if the Action failed it tries to re-plan a sequence of Actions or it indicates the Spatial Reasoner that the path cannot be followed. Perception Schemas can also interact with the higher-level activity by updating the Long Term Environment Memory. Finally, the Schema Controller sums and balances the commands to motors generated by activated Motor Schemas (cf. Command Arbitration) according to the respective importance of Behaviors. Once done, the command vector is applied to robot’s motors.

Aura architecture description has proved the efficiency of the conceptual model on a complex hybrid architecture. However, Aura being not dependent on a given technology neither on particular algorithmic approach, the appropriateness of the conceptual model is favored.

B. CLARATy

CLARATy is a generic software architecture developed at NASA JPL, which relies on a more technically-centered approach. The example of use of the conceptual model for describing the CLARATy "solution" mainly refers to our understanding of [30] and [13]. We use the same notation as that previously used for knowledge entities (cf. Fig. 5).

In CLARATy, the decomposition of architecture into subsystems is explicit. Robot Body parts (e.g. the Arm) are associated to specific subsystems (e.g. the Manipulator system). A subsystem controlling the whole robot can also exist. Subsystem are contained in a system representing the entire robot control architecture that, in turn, can be contained in a system representing a robot team, allowing so to manage different levels of control according to systems of different granularities. They are situated in the functional layer. Each subsystem can be decomposed into a hierarchy of activities (cf. Fig. 5), ranging from subsystem’s Local Planning to Motor and Sensor control. These latter activities coordinate according to a classical request/reply protocol (not represented in Fig. 5). Subsystems are under the control of the Decision activity that belongs to the Decisional layer. The Decision activity, implemented with the CLEAR framework [13], merges mission and robot planning part with the executive part, unlike most (three) layered architectures.

Decision activity coordinates with each subsystem in two different ways: it queries its state during a given time interval (potentially only one time sample) to get information on perceived Environment (1) or on Robot Body (4) state; it orders the execution of an Action (6) with a given Objective (7). The originality of CLARATy is to allow the Decision activity to coordinate with any level of decision within subsystems (e.g. Local Planning, Action Execution, etc.), according to the level of precision of control (Action (6)) or perception (Environment (1) or Robot Body (4)) it requires.

Finally, the most important thing is that subsystems are based on hierarchies of knowledge entities which organize knowledge upon a specialization relationship. The specialization relationship provides Robot Body knowledge entities more or less precise and adapted to a specific hardware context, in other words, adapted to the physical properties of a robot.

The specialization comes from the use of the Object
paradigm, which also allows, in a limited context, the unification of Knowledge and Activity entities. This latter point has raised the importance of the design paradigm for the description of an architectural solution. So, the appropriateness of the conceptual model (to represent the CLARATy architecture) is less evident. That brings out the need of a specialization relationship in the model semantics.

Unlike Aura, CLARATy does not explicitly refer (cf. Fig. 4 and Fig.5) to Strategy, Behaviors and Interactions (as far as we know). From this report, the difference of control methodologies used in lower layers of these hybrid architectures (resp. reactive and functional) is highlighted.

VI. CONCLUSION

This paper has argued on the necessity to define a dedicated language for communicating and understanding architectural choices in robotics. It has also shown what could be the concrete advantage of such a language in a future development methodology based on DSLs, for an easier and faster integration of domain expertise.

We propose a conceptual model as a first basis to define such a language. This model abstracts from technological considerations. It also supports general definitions of recurrent concepts used in robotic architectures, like layers or behaviors. Finally it allows expressing architectural specificities by providing adequate abstractions and by promoting the importance of control organization thanks to different abstract types of entities: System, Activity, Coordination and Knowledge. Examples of use of this model have been developed on two different and well-known architectures. The use of a unique conceptual model helps in understanding differences among architectures.

Future work aims for the design of decision mechanisms of Activity entities and that of protocols of Coordination entities. Simple concepts and terminologies have to be defined to easily express their respective properties. Furthermore, control architectures being more or less generic and adaptable, the conceptual model should support the explicit description of instantiations, of variability (optionality), and of specialization (refinement) of concepts.

REFERENCES


[10] J.D. Carbone, D. Andreu, and P. Fraisse, "Events as a key of an autonomous robot controller". In 15th IFAC World Congress (IFAC b'02), 2002.


