Atoms based control of mobile robots with Hardware-In-the-Loop validation

Adrien Lasbouygues, Benoît Ropars, Robin Passama, David Andreu, Lionel Lapierre

Atoms Based Control of Mobile Robots with Hardware-In-the-Loop validation

Adrien Lasbouygues[1], Benoit Ropars[1,2], Robin Passama[1], David Andreu[1] and Lionel Lapierre[1]

***Abstract*— Mobile robots are nowadays used in a wide variety of missions especially for exploring environments and allowing scientists who study these environments to gather data about them. However the monolithic design of control laws is often a hindrance to adapting control solutions from a robotic application to another. We thus propose a new control description paradigm strongly focused on modularity as well as integrating constraints coming both from the system capabilities and from the concerns of control engineering (such as stability). In this paper, we present the main ideas behind our approach and illustrate it through Hardware-In-the-Loop simulation.**

I. Introduction

The first mobile robots were developed in the late 60's. Their development has never stopped since leading to more and more autonomous robots capable of performing missions in more and more complex environments. Thus they began interesting other sectors of the scientific community as a tool to explore environments too dangerous or too costly for human exploration. This is especially true in our application context which is underwater robotics (for karstic exploration, archeology or biology) in which we exploit application-dependent models of the environment.

However a broader use of robots in scientific applications is largely impeded both by the time and cost of developing a control for a given robotic application (i.e. the resulting interaction between a robot, its environment and the task performed) and by the difficulty to integrate knowledge from a specialist of the environment to this application.

The main underlying reason behind these issues is the way control laws are designed by control engineers. Indeed as explained in [1], control engineers focus on solving the control problem at hand often neglecting other considerations. Thus the resulting control laws are very monolithic, merging knowledge together while strongly hiding relationships between them as well as design assumptions.

Moreover control engineers rarely take into account the fact that the control will then be implemented on a Software and Hardware architecture which brings its own constraints and limitations (for instance sensors update rates or computation power). The monolithic nature of control equations also strongly limits software engineers leeway when implementing the control thus not allowing them to use the full scope of their proposals in term of software modularity and architectural solutions.

We thus believe that a strong focus should be put on modularity concerns from the very beginning of the control design and up to the implementation of this control. This is the guideline of the approach presented in this paper.

We are currently working on the modified version of the Jack ROV which was presented in [2].

This paper is organized as follows. In section II, we discuss related works as well as our positioning towards them. In section III, we describe the basic entity of our approach. In section IV, we describe how to compose these entities together to describe a control law. Section V presents an insight on how Constraints are studied and show their importance through a rather simple heading angle control example which is validated using Hardware-In-the-Loop (HIL) simulation. Finally this paper ends with a conclusion and an opening towards future work.

II. Related works and Positioning

A. Modularity in control engineering

If modularity is a recent concern for control engineers, a few approaches already tried to take it into account.

Interesting approaches are the ones based on hybrid control such as [3]. Indeed modularity is achieved by partitioning a complex control problem into simpler subproblems and to develop an individual solution for each of these subproblems. The complete solution is obtained by switching between the subproblem solutions according to the situation faced by the robot. However implementation on a software architecture is not taken into account in this approach.

Another interesting approach is Motion Description Language (MDL) introduced in [4] which was then extended in works over MDLe [5]. Focused on hybrid control, these approaches describe a control as a sequence of kinematic state machines with a lifetime T that can be aborted by an interrupt. Such an entity is called Atom or Modal Segment and if we use the same name for our basic entities (since in both cases they are "minimal" entities composed to form more complex ones), our definition of an Atom is very different as exposed in section III.

Another noteworthy work is the Modelica modelling language. Though Modelica primary concern is not control law design, it presents relevant concepts that could be extended to control design. Modelica is a language aimed at simplifiying the exchange of models and model libraries used for

The authors would like to thank the French Labex NUMEV for supporting this research.

[1]Adrien Lasbouygues, Benoit Ropars, Robin Passama, David Andreu and Lionel Lapierre are with LIRMM, University of Montpellier, Montpellier, France. E-mails: {adrien.lasbouygues, ropars, robin.passama, andreu, lapierre}@lirmm.fr

[2]Benoit Ropars is with Ciscrea, Toulon, France. E-mail: bro@ciscrea.fr

modelling complex systems [6]. It builds on the benefits of object-oriented methodology and applies its principles for general-purpose physical modelling thus allowing to model complex systems by assembling components [6].

Modelica was then given the possibility of describing controllers through the addition of several libraries such as the ones presented in [7] and [8]. But Modelica approach is not focused on control description and so does not encompass issues related to its real-time implementation.

B. Modularity and software engineering

On the other hand, modularity has long been a major concern for software engineers especially when designing robot control architectures. It has led to the development of several modular architectures such as CLARAty [9]. One can also cite GenoM [10], an approach based on the generation of modules in order to integrate control algorithms to allow their composition and implementation with no precise information about their content. One can also refer to robotic middlewares such as ROS[11]. All these works aim at developing modular control softwares and control modularity and requirements are not explicitely adressed.

In [1], a layered architecture was also proposed to improve modularity on sensing and actuating steps. Thus a Sensor Fusion layer and an Actuator Fusion layer are added to separate concerns between the control design and the physical system drivers. Moreover requirements and capabilities of the system are expressed through the interface of the components which encompasses data's unit, value, valid range as well as measurement uncertainties. However modularity of the control layer is not taken into account.

In [12], design contracts based on temporal constraints and delays are proposed to organize the interaction between control and software engineers. These contracts reify the necessary tradeoff between control and software engineers when implementing a control. However the design process in itself is not detailed.

C. Positioning

As we saw in previous subsections, no work tried to propose a comprehensive approach encompassing the whole control design process from its description to its implementation while putting a strong and effective emphasis on modularity. Moreover, while most approaches such as [1] propose a separation of concerns between control engineering and software engineering, we are considering the integration of both concerns throughout the design process aiming at a better integration of a control on the software architecture used to implement it.

Figure 1 presents an overview of the proposed methodology. On the one hand of the process, the control engineers describe the control from a composition of basic entities called Atoms which encapsulate knowledge coming from various sources. We will give a more precise description of this part in sections III and IV.

On the other hand of the process, the software engineers will implement the control on the Software and Hardware architecture of the implementation target using the projection of Atoms as software entities using a dedicated API. This part will not be specifically adressed in this paper.

However the frontiers between these different domains are rather porous since expertise of one domain specialist might also be required in the other one.

Thus we introduce between them a "Bridge" which, taking advantage of the precise control description allowed by Atoms, will allow us to study our control properties to check if the control can be implemented on our target system. And if it can, this study allows us to produce information that will help the software engineer in his task. This part will be presented in more details in section V.

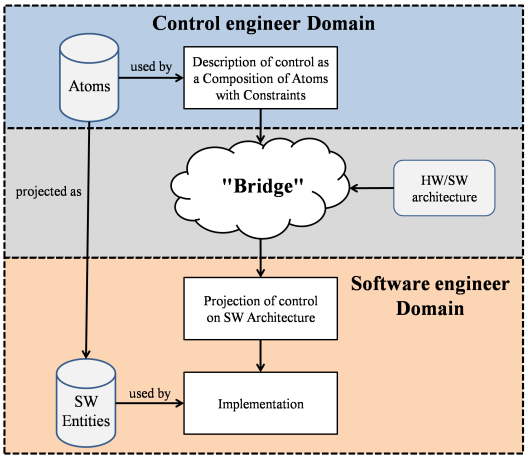

Fig. 1. The "Bridge" between control design and software implementation

III. Encapsulating knowledge: towards the Atoms

As we want to describe the control of a robot in a modular way, we have to determine precisely which criteria our modular approach should satisfy in order to design the entities which will be the base of our control design paradigm in the most suitable way.

A. Knowledge in control engineering

First of all we should define what we mean by the term knowledge.

We define as knowledge any relevant information allowing the robot to perform its mission online as well as any data that could be gathered to allow offline post mission processing.

This knowledge can take heterogenous forms (parameters, models, these models being maps, equations, ...). Moreover knowledge comes from various sources. For instance control engineers bring control equations, environment specialists provide their models of the environment, robot manufacturer knows the robot's hardware specifications. Thus we would like that control engineers who design the control equations and software engineers who then implement them to be able to handle this knowledge without being familiar with it.

Finally knowledge is only the expression of the current set of information we know about a specific point (this is particularly true for environment knowledge).

Thus knowledge is evolving as we gather data through experiments or when theoretical breakthroughs are achieved. We thus would like to be capable of making models evolve without having to redesign our whole control laws.

We should then design the control no longer as a monolithic block but as a composition of entities. These entities shall encapsulate knowledge to allow it to be manipulated easily without requiring information about their precise content and allow to modify their content without affecting the surrounding entities. We are thus required to design a self-sufficient interface to prevent the user from making any assumptions that could lead to composition errors.

Finally these entities should also convey the information that would be relevant for a software engineer to implement this control within his software architecture on a given hardware architecture while respecting the requirements from the control engineer.

B. Atoms: the basic entity

Based on previous statements, we created an entity named Atom to support our new control description paradigm. These Atoms are mainly inspired from the object paradigm and are in their structure similar to Modelica objects. Moreover we paid a particular attention on their interface as it will be discussed in the next subsection.

They are defined as:

Definition 1: An Atom is a minimal and indivisible composable entity encapsulating a piece of knowledge. An Atom, A, is represented as a 5-tuple: $A = (Name, Phy, Int, IntPar, Constraints)$

The Inerface parameters, $IntPar$, will be defined in the next subsection and Constraints will be presented in section III-D.

The interface, Int, is what allows an Atom to be connected to other Atoms. It is a 3-tuple (Ne, Pr, IS):

- The Needs (Ne): the set of knowledge required by the Atom.
- The Products (Pr): the set of knowledge produced by the Atom.
- The Internal Storages (IS): the set of knowledge stored by Atoms (for data historization or integration of local states for instance).

Any knowledge item belonging to one of these sets is called an interface element. It is a pair $(Name, Datatype)$. The Datatype of an interface element i will be noted as $D(i)$ and will be defined more precisely in the next subsection.

The Physics, Phy, is the heart of the Atom. It is an application defined as:

$$Phy : D(Ne_1) \times ... \times D(Ne_{Dim(Ne)}) \times D(IS_1) \times ... \times D(IS_{Dim(IS)}) \rightarrow D(Pr_1) \times ... \times D(Pr_{Dim(Pr)}) \times D(IS_1) \times ... \times D(IS_{Dim(IS)})$$

However there exists a specific kind of Atoms called External Knowledge Atoms characterized by an undefined Physics. They are used to explicit the elements that are not directly part of the control composition but either produce knowledge used by the control or use knowledge produced by the control (for instance sensors, actuators, network links, files to store data). Being implementation dependent, their Physics is not defined and they will be projected as software entities in a different way Atoms are.

We must note that the Atom paradigm, on the opposite of other object-oriented approaches such as Modelica, doesn't integrate inheritance mechanisms.

C. Interface characterization : the Datatypes

As mentioned previously, the focus on the interface definition is crucial for allowing an efficient use of Atoms. Thus the Datatype of the interface elements should convey all the required information to use the Atom.

The Datatype is a 3-tuple $(Type, Frame, Axis)$. It contains the Type of the exchanged data which can be either a physical quantity (i.e. Force, Speed, Acceleration and so on) or an arithmetic type (i.e. Boolean, Unsigned Integer and so on). While in other approaches such as [1] or [6], interfaces express the unit of the data, in our approach, the Type is an object that encapsulates the data and offers different getters and setters (in the way they are defined in object methodology) for the different units associated with the quantity. Thus the element is specified in terms of quantity and no longer in terms of unit.

But can we consider that knowing the Type is sufficient ?

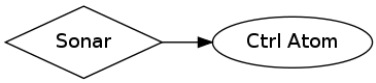

Fig. 2. The simple example of a control Atom using sonar measurements

Let us consider the Figure 2 example where measurements performed by a sonar are used by a control Atom. How can we interpret the control designer's intentions ? With no other information the following assumptions are possible:

- The control is performed in sonar frame.
- The sonar measurements are converted to robot frame in the sensor's driver.
- The sonar measurements are converted to robot frame in the control Atom.
- The sonar is mounted in such a way that sonar and robot frames coincidate and thus no conversion is required.

Without more information we can't sort out which one is the right assumption. And when we have to perform this kind of assumptions at numerous locations when reusing a control description, there is little chance our implementation will work and finding which error(s) were made on the interpretation of the control would be a very difficult task.

This is why in addition to the type, Atom Datatypes also encompass a Frame field used to indicate in which frame is expressed the data. In the previous example, we indicate that the Product of the sonar is expressed in the *PROXIMETER* frame while the Need of the control is expressed in the *ROBOT* frame. Thus direct connexion between them is impossible and we are forced to add an Atom in charge of performing a frame shift from the *PROXIMETER* frame to the *ROBOT* frame as shown in Figure 3, explicitely indicating the presence of a frame shift.

Similarly we add another field called Axis that is in charge of indicating to which axis or plane of the Frame is related

Fig. 3. With the Atom Datatype, we are forced to express explicitely the shift from PROXIMETER to ROBOT frame.

the data. With this approach, we are capable of detecting and preventing misconnections between Atoms during the design phase. To understand their importance let us consider the Atom presented in Figure 4 which is in charge of controlling the heading angle (Yaw) of a robot (either with a static reference or for path following).

Its Physics contains the following equation:

$$acc = acc\_des + K1 * de + K2 * e$$

where e is the Yaw error, de its derivative, acc_des the desired acceleration profile and acc the resulting acceleration to be applied to the robot's dynamic model.

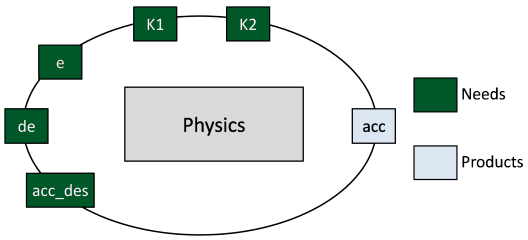

Fig. 4. Interface of the Atom containing the Yaw control equation

It is clear that e, de, acc_des and acc should relate to the same Frame and Axis, if not the control's behavior will not be the expected one. The Atom's Interface can then be described as :

$$\begin{aligned} Ne = \{&(e, [Angle, WORLD, z]),\\ &(de, [AngularSpeed, WORLD, z]),\\ &(K1, [Frequency, NOFRAME, NOAXIS]),\\ &(K2, [FrequencySquared, NOFRAME, NOAXIS]),\\ &(acc\_des, [AngularAcceleration, WORLD, z])\} \end{aligned}$$

$$Pr = \{(acc, [AngularAcceleration, WORLD, z])\}$$

where the Axis is set to z to indicate that the angle revolves around the z axis of the $WORLD$ frame, $NOFRAME$ and $NOAXIS$ indicate that this Need is not related to a particular Frame and Axis and FrequencySquared is the quantity name we have associated with the unit Hz^2.

To end the description of interface, we must introduce a final notion which is Interface Parameters. Indeed the previous equation used to control the Yaw angle of our robot could as well be used to control its Roll or its Pitch, the only difference being the Axis related to the interface. We could make three Atoms for those three situations but that would obviously be a waste of time (and also a quite error-prone process). Thus we introduce a parameter that will allow us to indicate that the Axis may vary depending on how the Atom is used. Hence in the previous interface, z can be replaced by an Interface Parameter.

We can of course also define Interface Parameters on the Frames. However defining Interface Parameters on Types is not allowed in our approach to avoid the design of models which would be described in a too abstract way.

D. Temporal constraints

We need to specify a given number of additional information to be capable of efficiently conveying knowledge from control description to its implementation.

These information, that we call Constraints, should allow us to encompass both the requirements of the control engineer and what the system can do, as described in [12].

We have chosen to focus on temporal constraints since they are particularly relevant in our context. Indeed control engineers study stability of their control in continuous time through tools such as Lyapunov functions. However since control is executed in a discrete-time environment, periods of execution become detrimental to maintain the stability of the controlled system (though current tools to determine these periods are very limited). On the other hand, temporality is instrumental for the software engineers to know with which mechanisms of their architecture and middleware they should wrap knowledge (for instance consider the difference between synchronous and asynchrounous execution). Moreover it is crucial for them to know how to set up the architecture and especially at which period the synchronous components should run.

Thus temporal constraints are applied both on the Needs of the Atom and on its Physics. A Constraint is a pair $(value, properties)$ with the set of available properties depending on the current Constraint $value$.

On the Physics, Constraints represent how the Physics should be run temporally and thus how Products values are updated. There exists three possible Constraint values:

- Constant: the Physics is run only once (initialization).
- Sporadic: we don't know when the Physics will be recomputed. It is associated with a property called $C_{s_{phy}}$. This property refers to an Atom describing the conditions under which the physics should run (an "important enough" change in a input value for instance). This woud allow us, for instance, to describe controllers based on Event-triggered control [13].
- Periodic: the Physics should run on a periodic basis. It is associated with two properties called $t_{comp_{max}}$ representing the worst-case computation time of the Physics (this value of course depends on the implementation target) and T_{phy} which is the allowed or possible set of intervals of periods for the Atom. For instance, in the case of a control Atom, T_{phy} will correspond to the set of periods for which we can guarantee that the controlled system remains stable. On the other hand, for a sensor, T_{phy} represents its set of update rates.

Constraints related to Atom Needs describe how the value of the Needs should be updated through time to ensure that computations performed in Physics always produce the expected result and/or preserve system stability. There are 4 kinds of constraints:

- Constant: the Need's value is set once and for all during initialization.
- Sporadic: we don't know when the Need's value will be updated.
- Coupled: the Need has to be updated every time the Physics is runned. This induces a temporal coupling between the two Atoms linked together.
- Periodic: the Need is updated periodically (independently of Physics Constraints). It is associated with a property T_{Need} specifying the allowed update periods.

The Constraint on a Need n, noted $Ctr(n)$ and the Constraint on a Product (and hence on the associated Physics) p, noted $Ctr(p)$, are said compatible if they are both Constant, both Sporadic, both Periodic or Coupled with Periodic. This is noted $Ctr(n) \equiv Ctr(p)$.

IV. A CONTROL LAW AS A COMPOSITION OF ATOMS

In the previous section, we introduced a minimal entity called Atom. To describe a control, we now need to connect them together.

A. The Molecules

Since Atoms are basic entities, it would be useful to allow to have an entity allowing us to use groups of Atoms without having to describe by hand their association every time we need them. This is the role of Molecules defined as:

Definition 2: A Molecule is a composable entity whose Physics is made of a composition of Atoms or Molecules. A Molecule M is a 5-tuple:

$M = (Name, Phy, Int, IntPar, Constraints)$

with $Phy = (E, l, L)$

where E is the set of entities (Atoms or Molecules) used in the Physics, l the set connections of these entities with the interface of the Molecule and L the set of connections between the entities.

B. Composition of atoms

The process of connecting Atoms and/or Molecules together through their interface is called Composition. It results in creating a Composition of Atoms which is the atomic description of a control law. However in order to use an Atom or a Molecule in a Composition, we have to set up how it will be used. This means we have to valuate its Interface Parameters and its Constraints (though the properties associated with the Constraints don't have to be since they may depend on the implementation target). This process will be referred to as Instantiation and the resulting elements as Instances.

A connection between a Product of an Instance and a Need of another one is called a Link. Let n be a Need and p a Product. The Link between them is noted $L(n, p)$ and we have : $fan\text{-}in(n) = 1$ and $fan\text{-}out(p) = \infty$.

Finally a valid Composition is defined as:

Definition 3: A Composition made of a set I of Instances and a set L of Links, noted $C = (I, L)$, is valid if and only if L_i is valid for all i.

Definition 4: A Link $L(n, p)$ is valid if and only if $D(n) = D(p)$ and $Ctr(n) \equiv Ctr(p)$.

V. THE ATOMS THROUGH EXAMPLE

In order to allow explanations and figures to remain simple to explain and to display, we will focus on a quite simple application: a heading angle control algorithm.

Figure 5 presents the legend that will be used in next Figures. Moreover, the Links between Atoms have been merged for the sake of readability of the figures.

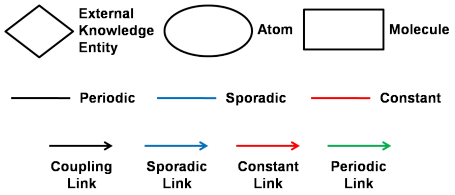

Fig. 5. Legend used on Figures 7 to 9

A. Control Description

Thanks to the fine grain description allowed by atoms we can represent a control description as a directed multigraph known as Knowledge Association Graph.

Definition 1: A Knowledge Association Graph $G_C = (V, E)$ is the directed multigraph associated with the Composition $C = (I, L)$ so that:

$V = I$ and $E = L$

Such a graph is presented in Figure 7 to Figure 9. The "Bridge" presented in Figure 1 then consists in manipulating this graph to structure the control and extract relevant information for its implementation as shown in Figure 6.

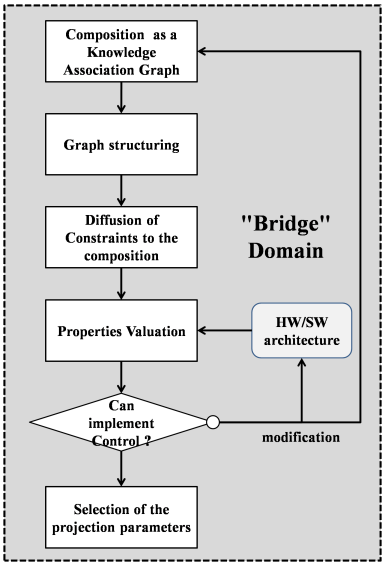

Fig. 6. The Different steps linking control design and implementation

Our robot, the Jack, is equipped with an Inertial Measurement Unit (IMU), a Loch Doppler and a Depth sensor. From these sensors, we can reconstruct our robot's state as shown in Figure 7. Moreover, since the way the IMU and Loch Doppler are mounted can influence the measurements, we must perform Frame Shifts in order to express the measurements in Robot's body frame. In Figure 7, ExpertiseIMU and ExpertiseLoch are External Knowledge Atoms used to indicate how they are mounted (information coming from the robot's manufacturer). You must also note three Molecules

RobotSpeedsBody, RobotSpeedsWorld (an Instance of the same Molecule as RobotSpeedsBody but with a different Interface Parameter) and RobotPose. They are used as buffers allowing to insulate the computation of the robot's state from the rest of the Composition. This is a design technique ensuring a better modularity in control design since we can change the way they are computed (by introducing sensor fusion for instance or by using a SLAM algorithm to compute robot's pose) without having to reroute the Links referring to Robot's state thus simplifying the changes.

Before the Actuators, the Dynamic Model (Figure 8) is used to turn the desired accelerations into forces and torques expressed in the robot's frame. Desired accelerations coming from the control don't have to be updated at every iteration hence allowing us to introduce Periodic Links (Figure 9) while robot's speeds are required at every computation to determine the damping and coupling terms to compensate them thus inducing a temporal coupling with Robot's state reconstruction. The control (Figure 9) is based on the DynamicAngleControl Atom presented in section III which is used to control the heading angle. The related ExpertiseYawControl allows to set the values of K1 and K2 as well as set to 0 the desired acceleration and speed (since reference is static). The Yaw error is computed by a simple substractor and the desired angle is set by the operator through a Network link whose value is updated on a Sporadic basis. The error derivative is computed by an AngularSpeed substractor. Moreover since the produced acceleration is expressed in the *WORLD* frame, we have to frame shift it to *ROBOT* frame in order for it to be used by the Dynamic Model which expects its inputs in *ROBOT* frame (Figure 9).

Finally we have introduced Separator Atoms between Robot's states and the control which uses them (Figure 9). The Separator Atoms are used to introduce periodic Links thus enforcing time-decoupling between the control Atoms and the ones in charge of reconstructing robot's state. Hence both can run at optimum periods without being constrained by the other part of the Composition.

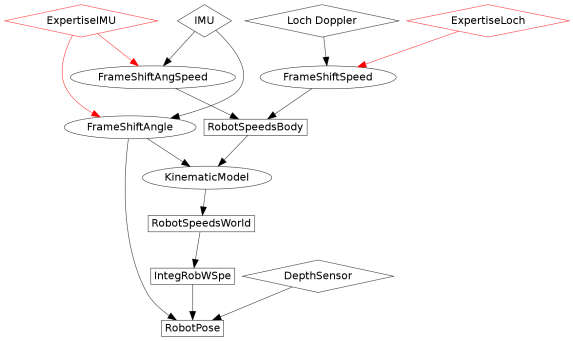

Fig. 7. Our example described as a Knowledge Association Graph part 1

B. Constraints evaluation

We should now see how a Composition can be structured so that we can merge the Constraints exerted on each Instance into Constraints exerted on their Composition as shown in Figure 6.

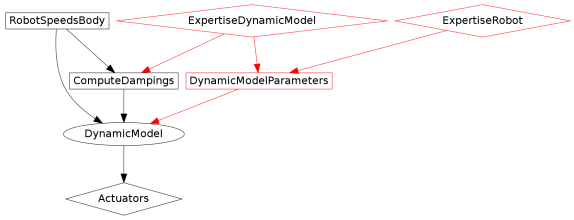

Fig. 8. Our example described as a Knowledge Association Graph part 2

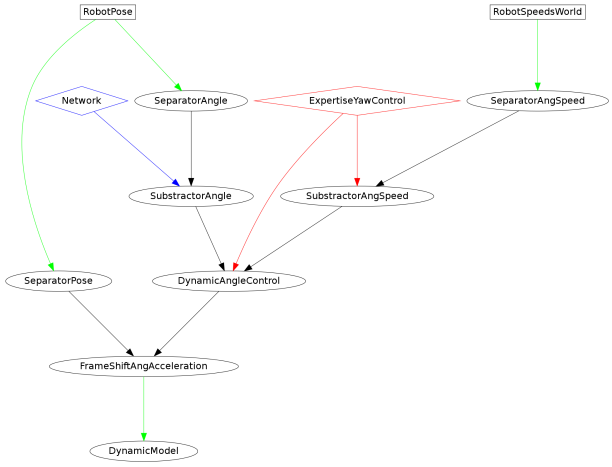

Fig. 9. Our example described as a Knowledge Association Graph part 3

We begin by checking the control validity. Then in order to simplify the Graph, we merge edges that connect the same nodes with identical Constraints. Finally since our main goal is to determine the period of executions of all periodic Instances, we remove the nodes and edges corresponding to non periodic Instances. The resulting Graph for our example is shown in Figure 10.

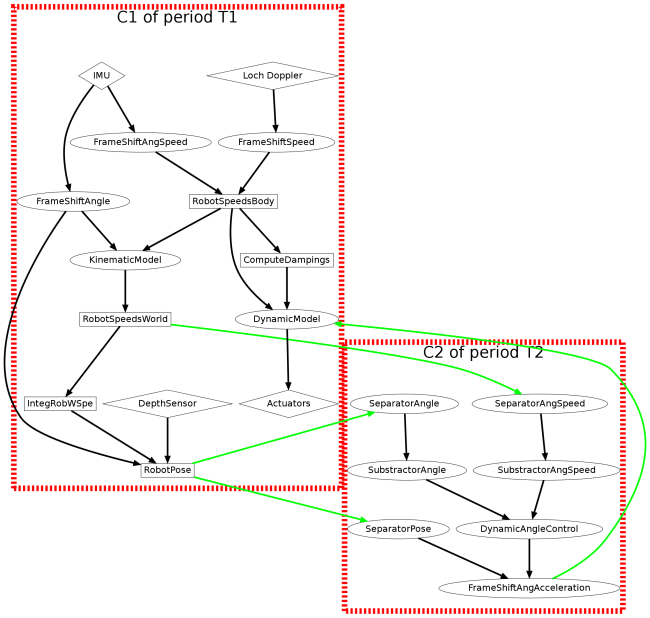

Fig. 10. The Graph with only periodic Atoms and separated Compositions

We then use the periodic Links to separate the Composition that are temporally independent and thus their Contraints will be studied independently.

From this we can determine the Constraints related to each Composition through the Diffusion process (Figure 6).

Let us consider the second one, $C2$. We have:

$$t_{comp_{max}}(C2) = \sum_{i=1}^{7} t_{comp_{max}}(Inst_i) + \Delta_{com} \quad (1)$$

where Δ_{com} represents the communication time between the Atoms of the Composition and $Inst_i$ the set of Instances included in the Composition.

We can then compute the set of possible execution periods of the composition, T_{exe}, through intersection of allowed period sets of the different Instances:

$$T_{exe}(C2) = \bigcap_{i=1}^{7} T_{phy}(Inst_i) \cap [t_{comp_{max}}(C2)\ \infty] \quad (2)$$

We then valuate the different properties. If T_{exe} of any Compositions is an empty set then the control cannot be implemented on the desired target. Otherwise one can choose a period $T2 \subset T_{exe}(C2)$ as the period of execution of all Atoms in the Composition. Similarly we can choose $T1 \subset T_{exe}(C1)$. This approach is for now an offline process but we are considering its embedding in the application to cope with potential changes in the temporality of some Atoms.

C. HIL setup presentation

Our HIL setup is constituted of the BeagleBone Black [14] equipping our robot and which serves as main controller. A Linux kernel 3.8.13 with a Debian distribution and patched with Xenomai 2.6.3 is installed on the BeagleBone. It embeds the ContrACT real-time middleware developed at LIRMM [15] that allows to precisely configure software components' timing properties, which reveals very complementary to the Atom approach. The controller is connected to the simulation PC through a network link and both communicate through an UDP protocol ensuring non blocking communications. The simulator contains a model of the robot, environment representation and sensors simulation with noise.

D. Simulation Results

The following results have been obtained with $T1 = 25ms$, $T2 = 50ms$, $K1 = 10.0Hz$, $K2 = 10.0Hz^2$ and $\psi_{des} = 75°$. Figure 11 presents the heading angle evolution through time. Maximal noise levels on the measurements of ψ and r are $noise_\psi = 0.1rad$ and $noise_r = 0.05rad/s$.

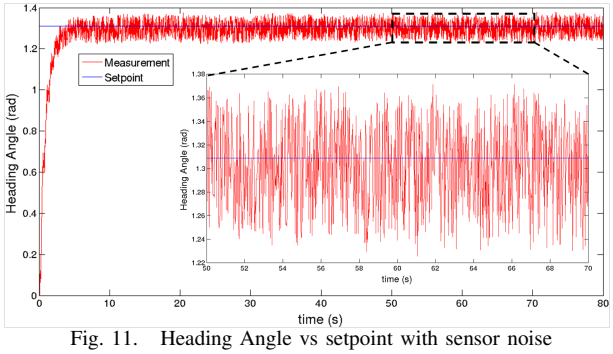

Fig. 11. Heading Angle vs setpoint with sensor noise

Figures 12 to 15 contain the execution time measurements performed on the real-time loops in which Compositions $C1$ and $C2$ have been implemented. We focused on the 20 first seconds of the simulation. There is no direct mapping between Atoms and real-time loops but the precise process is beyond the scope of this paper. $C1$ is implemented using 4 loops: Sensors and Actuators to replace the drivers of the real components, MDynamic where the Dynamic model is implemented and Navigation to reconstruct robot state. C2 is implemented by 2 loops: Control and Frameshift.

Figure 13 and Figure 15 present the overall execution times associated with the Compositions. They integrate computation and communication times between the real-time loops. We neglected context switching times and these measurements also do not integrate the applicative scheduler's computation times. These times remain inferior to execution periods and the variability observed can be explained by natural computation times variations, calls to non real-time functions with important duration variations (especially the ones used to log data) as well as possible preemptions by the scheduler especially since we use other loops to communicate with the simulator and to store data. They also illustrate how hard it is to have a precise estimation of $t_{comp_{max}}$ values for Atoms and Compositions hence underlining the importance of HIL simulation in the design process.

Figure 12 and Figure 14 represent the evolution of delays of the real-time loops start times compared to the theoretical ones after the initial scheduling. For the execution i of a control loop, the delay is computed as:

$$delay_i = t_{start\_the_i} - t_{start\_real_i} \quad (3)$$

These delays are influenced by computation durations which impact start times of loops. Moreover since the BeagleBone possesses only a single-core processor, potential delays on a loop may also impact other loops start times. However the delays remain very limited and, if some appear, the scheduler manages deadlines to compensate for them. The overall delays remain within admissible bounds due to system imprecision as shown in Table I. The loop MDynamic is the most affected one by delays because it is the most computation time demanding loop and is very susceptible to be impacted by other loops (since it needs data from most of them). Thus it is the hardest one to manage for the scheduler explaining delays superiors to the other loops.

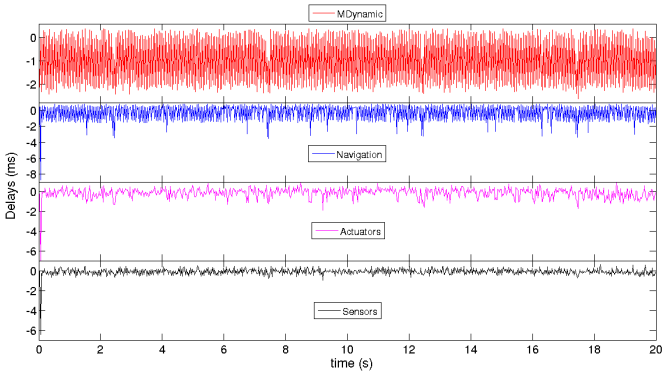

Fig. 12. C1 delays to theoretical start times

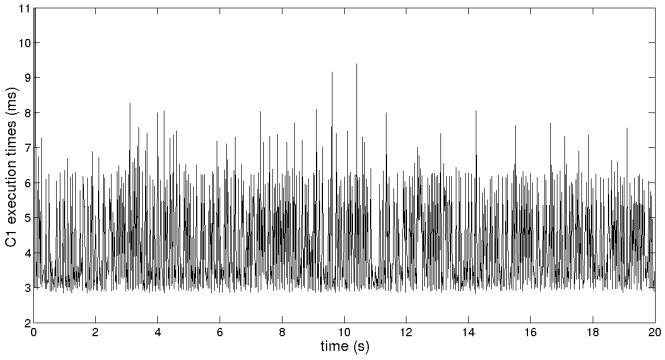

Fig. 13. C1 execution durations

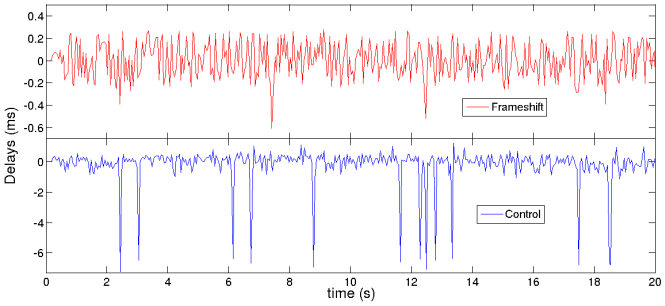

Fig. 14. C2 delays to theoretical start times

VI. Conclusion and Future Work

We have presented a new control design paradigm based on composable entities named Atoms and which benefits from object-oriented principles in order to design modular control laws.

We have illustrated it through an example described with this paradigm and shown how a control can be represented as a Knowledge Association Graph which is then manipulated to determine the Constraints applied to our Compositions, Constraints which are then used by software engineer to parameterize the real-time software architecture.

Further work is currently being done on defining a new entity called Alternative used to integrate concepts from switching and hybrid control to our control description methodology. Assessing the impact of the decomposition of monolithic control in terms of performance is also an important issue. Finally we would like to push forward the notion of Knowledge Association Graph to try an see if any further control properties could be studied from it as well as which kinds of Constraints it would be relevant to add to the already existing temporal constraints.

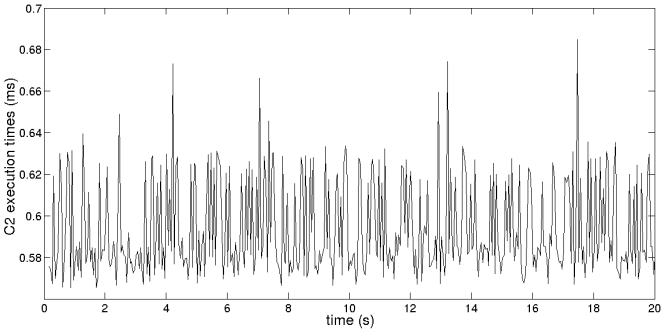

Fig. 15. C2 execution durations

TABLE I

Average delays of loops and desried periods.

RT Loop	Comp.	Av. delay(ms)	Period(ms)	Error(%)
Sensors	C1	-0.3583	25	1.433
Actuators	C1	-0.1535	25	0.614
Navigation	C1	-0.2808	25	1.123
MDynamic	C1	-1.0358	25	4.143
Control	C2	-0.4878	50	0.976
Frameshift	C2	-0.0173	50	0.035

References

[1] P. Ulbrich, F. Franzmann, C. Harkort, M. Hoffmann, T. Klaus, A. Rebhan, and W. Schröder-Preikschat, "Taking control: Modular and adaptive robotics process control systems," in *Robotic and Sensors Environments (ROSE), 2012 IEEE International Symposium on*, 2012.

[2] B. Ropars, A. Lasbouygues, L. Lapierre, and D. Andreu, "Thrusters dead-zones compensation for the actuation system of an underwater vehicle," in *European Control Conference (ECC'15)*, 2015.

[3] J. M. Toibero, F. Roberti, F. A. Cheein, C. Soria, and R. Carelli, "Stable switching control of wheeled mobile robots," in *Mobile Robots Navigation*, A. Barrera, Ed. InTech, 2010, ch. 19, pp. 379–400.

[4] R. W. Brockett, "On the computer control of movement," in *Proceedings of IEEE 1988 International Conference on Robotics and Automation*, 1988.

[5] D. Hristu-Varsakelis, P. Krishnaprasad, S. Andersson, F. Zhang, P. Sodre, and L. D'Anna, "The MDLe Engine: a software tool for hybrid motion control," Center for Dynamics and Control of Smart Structures, Tech. Rep. CDCSS TR 2000-8, 2000.

[6] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica - the new object-oriented modeling language," in *The 12th European Simulation Multiconference (ESM)*, 1998.

[7] M. Baur, M. Otter, and B. Thiele, "Modelica libraries for linear control systems," in *Proceedings of the 7th Modelica Conference*, 2009.

[8] M. Bonvini and A. Leva, "A modelica library for industrial control systems," in *Proceedings of the 9th Modelica Conference*, 2012.

[9] R. Volpe, I. A. D. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das, "CLARAty: Coupled layer architecture for robotic autonomy," NASA - JET PROPULSION LABORATORY, Tech. Rep., 2000.

[10] A. Ceballos, L. D. Silva, M. Herrb, F. Ingrand, A. Mallet, A. Medina, and M. Prieto, "GenoM as a robotics framework for planetary rover surface operations," in *ASTRA*, 2011.

[11] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on Open-source Software*, 2009.

[12] P. Derler, E. A. Lee, M. Torngren, and S. Tripakis, "Cyber-physical system design contracts," in *ACM/IEEE 4th International Conference on Cyber-Physical Systems (ICCPS '13)*, 2013.

[13] W. Heemels, K. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, 2012.

[14] G. Coley, *BeagleBone Black System Reference Manual - Revision C.1*, May 2014.

[15] R. Passama and D. Andreu, "ContrACT: a software environment for developing control architecture." in *6th National Conference on Control Architectures of Robots, Grenoble, France*, 2011.