Control of Interconnected Homogeneous Atoms: Language and Simulator[1]

Michel Dubois, Yann Le Guyadec , Dominique Duhaut
Valoria Laboratory of the University of Bretagne-Sud, Vannes - Lorient, France
{michel.dubois,yann.le-guyadec,dominique.duhaut}@univ-ubs.fr

SYNOPSIS

Our work takes place in the general problem of building and controlling homogeneous robotics components that can dynamically reconfigure themselves to adapt their behaviour to a task. We present the design of our basic component, called atom, and the underlying functionalities. We show how these interconnected components may be controlled using a distributed reactive model (DROM) based on reactive objects. We discuss the definition of a high-level programming language based on the data-parallel paradigm offering control of parallelism and topological abstractions. Finally, the architecture of a reactive simulation engine compatible with the DROM model is presented.

1 INTRODUCTION

Our work takes place in the general field of modular robotics research. In this field we can distinguish two kinds of work. The first one tries to define elementary modules that can be human-assembled to build rapidly robots that respond to specific problems. For instance, RMMS (1) applies this approach to build manipulators, as well as the Golem project (2) where "*robots have been robotically designed and robotically fabricated*", and Swarm-bots (3) for the design and implementation of self-organising and self-assembling artefacts. On the other hand, modular robotics looks for self-reconfigurable structures. In this case, the problem is to build (usually homogeneous) components that can dynamically reconfigure themselves to adapt their behaviour to a task. Molecule Robots (4, 5) are based on a single component with two elementary moves, Conro (6) builds serial chain with 2DOF in rotation, I-Cube (7) modules are quite similar but with 3 rotations. In the Crystalline robot (4, 8), the elementary

[1] This work is supported by the French CNRS Robea program.

component is based on a 2D translation movement, Telecube (9) implements a 3D version of this Crystalline component, PolyBot/Polypod (9) are very rich structures based on simple components, and finally the MEL (10) proposes a two-rotation element with a universal connecting plate that allows for dynamic coupling.
We are interested in the analysis and the design of simple homogeneous walking robots called atoms. An atom is build as a kernel sphere on which 6 legs are plugged (see fig. 2). A leg can move (see fig. 3) following 2 degrees of freedom (45° around their axes) and each leg extremity is equipped with an IR send/receive component to perform relative localization between distant atoms, and a specific hanger which enables to interconnect atoms passively and to disconnect them actively. Regarding the kernel side of the leg, we use 3 force sensors for collisions and contact detection, and a gyroscope that provides general information on the atom (see fig. 1). No absolute localization capabilities are available. Interconnected atoms form a molecule that offers enhanced mechanical and networking capabilities using IR devices. A wireless device (bluetooth is under study) enables to communicate with distant atoms by broadcast and to centralize the deliberative level into a dedicated computer.

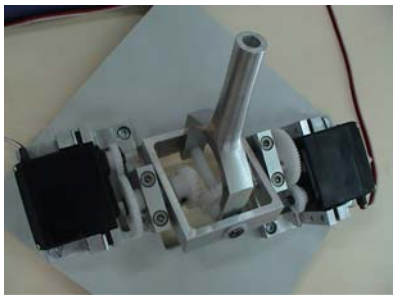

Fig 1: Prototype of a leg

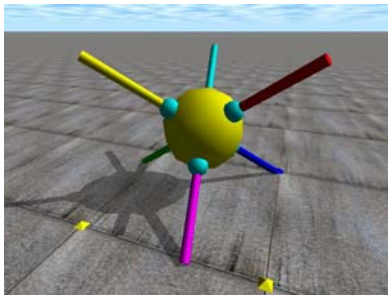

Fig. 2: A simulated atom

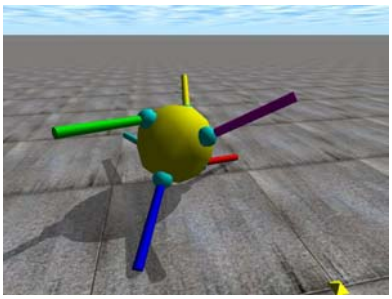

Fig. 3: ...moving legs

The ability to dynamically reconfigure the topology of a molecule gives us the capability to deal with many missions (walking, climbing,...). We use this platform to experiment different approaches of control: learning, multi-agent programming with emergent global behavior and direct programming language. A simulator of atoms and molecules is under development using ODE (Open Dynamics Engine) and Übersim to offer a common engine to perform tests until the real atoms are produced.
In this paper we present the execution model of atoms, and propose a programming interface to control atoms and molecules. Then we discuss on a distributed reactive implementation of the API. We present some ideas on the definition of a programming language that offers the programmer high-level abstractions to control atoms and molecules. Finally, we give details about the architecture of our simulation tool.

2 EXECUTION MODEL AND COMMUNICATION CAPABILITIES

From a microscopic point of view, the local embedded hardware on each atom is based on a FPGA to perform real-time control: perception, servo-control and local computations. From a macroscopic point of view, the architecture of interconnected atoms (a molecule) obeys the model of a network of synchronous modules interconnected by point to point wireless channels (IR devices are used both for relative localisation and point to point communication between interconnected atoms); each individual channel is lossless and preserves the ordering of messages, but the different channels are not mutually synchronised. Furthermore, (disconnected) atoms can communicate by broadcast using an embedded wireless device.

This gives us many information on the execution model we have to deal with to control atoms and molecules: finite memory assumption, local computation mechanism, point to point and broadcast networking capabilities.

3 FUNCTIONNALITIES

In this section, the services offered by the Application Programming Interface (API) are described. For the sake of simplicity, a light version of the class diagram is given in fig. 4.
We focus on specific points of the API. ***[classes ServoMotor and Leg]*** The *ServoMotor* class provides one degree of freedom to the *Leg* and offers PID control of velocity and position. Regarding the relative localization of a *Leg*, its *Spheric Position* gives two radius values which reflect the two freedom degrees of the leg, and its relative *Cartesian Position* which gives the position of the leg extremity relatively to the center of the *Atom*. The Map services (*getMappedLeg*, *setMap*) are a first step to offer the programmer a way to increase the symmetry in computations: one can logically rotate atoms and access legs via their *logical* ID instead of their *physical* ID. The methods *sendMessage* and *getMessage* offer asynchronous point-to-point communications between linked legs using the IR capabilities. ***[class Atom]*** *getNeighbors* allows an atom to refer its direct neighbors (the ones connected to its legs). ***[class Molecule]*** *atomize* disconnects every link in the molecule: each atom is promoted to a molecule. *stop* is implemented using the *stop* methods of *Atom*, *Leg* and *ServoMotor*: it turns off every servo-motor in a molecule. The reflexive association of the class *Leg* provides information about the topology of the corresponding molecule. The Link services (*setLink*, *unlink*) offer topological reconfiguration capabilities.

Note that there is no *link* method: this is a high-level knowledge which is deducted at the deliberative level because of our passive connection approach. This high-level information is simply stored in the object using the *setLink* method.

4 DISTRIBUTED REACTIVE IMPLEMENTATION

In this section, we show how the functionalities presented above can be implemented through the Distributed Reactive Object Model (DROM) (11, 12). This model supports either a centralized or a distributed deliberative control that takes in charge the achievement of the mission. For the sake of simplicity, we make the assumption that the deliberative layer is simply made of a single (centralized or replicated) process that communicates with local and distant reactive objects by (remote) method calls, but it can also support a reflexive reactive model that is suitable to dynamically adapt the task-level which continuously runs a reactive machine, like the Asynchronous Reflexive Model (ARM) (13).

4.1 Distributed Reactive Objects

A reactive system (11) is composed of a set of synchronously executed parallel components communicating by instantaneous broadcast events: communication is similar to the radio transmission where emitters send information that is instantaneously received by all listeners. The model proceeds in successive reactions, one at each logical instant. At each instant, the reaction is computed using the current input events and some internal information. The global behaviour of such a system results from the cooperation of its components.

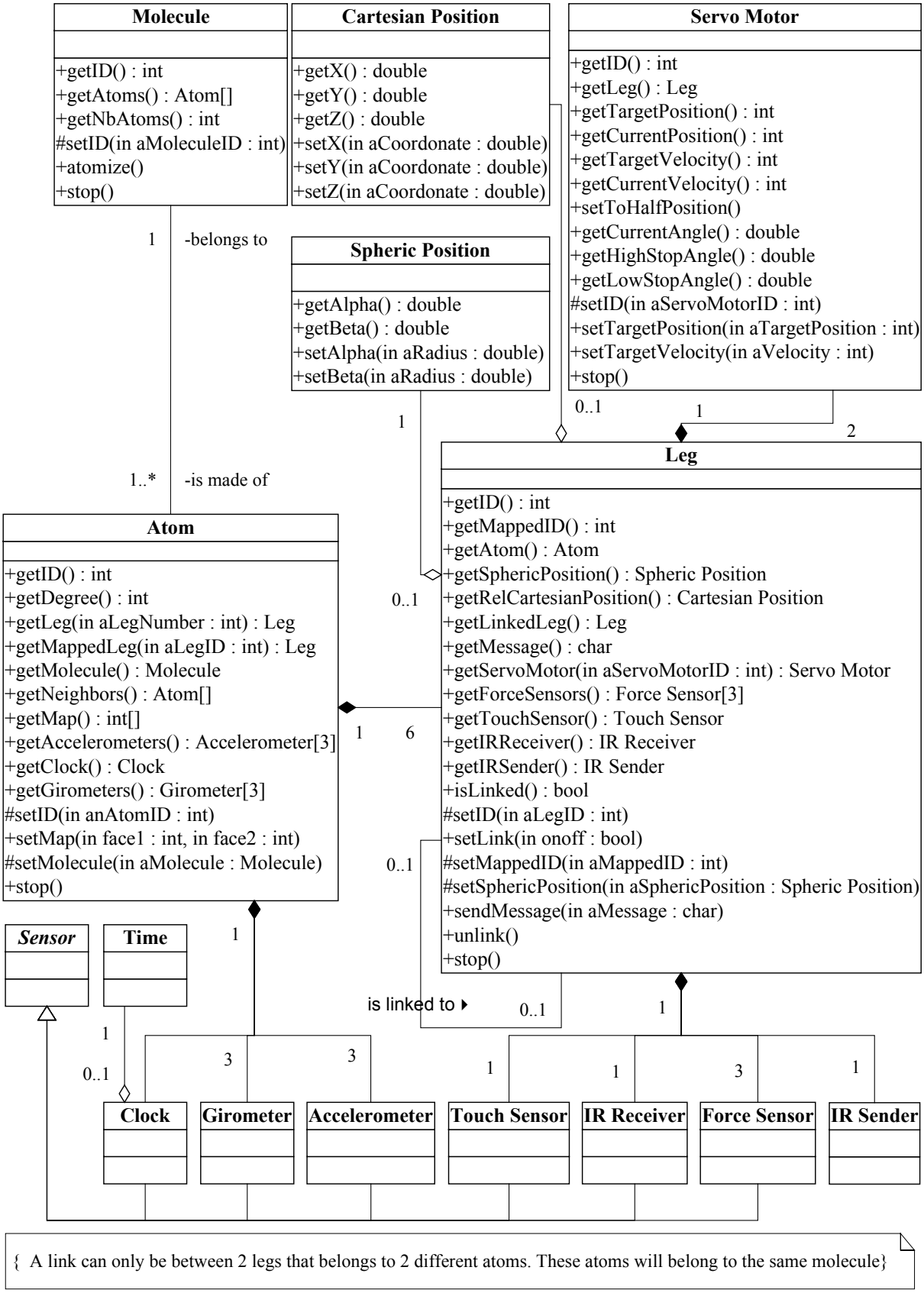

Fig. 4: Class diagram of the API

The Reactive Object Model (ROM) (11) merges the Reactive approach with objects. In this model, objects run reactive behaviours and communicate by method calls. Calls can either be accepted or rejected (for instance if a call already arise for this method during the same instant). Accepted calls are processed at the same instant they are issued.

The extension to distribution of the model, called DROM (12), basically introduces the notion of a reactive area in which all reactive objects share the same logical clock. In a distributed system, realizing global logical clock may be costly: components communicate within the same area by instantaneous orders, and between distinct areas by usual unbounded delay orders.

4.2 Application to the API

In our framework, each (physical) atom embeds objects that run reactive behaviours and communicate by (remote) method calls. The Molecule objects can either be replicated on each atom or migrate to an elected one. The DROM model naturally leads to decompose a set of atoms into several reactive areas (area = molecule), each of them defining its proper common time. The reactive paradigm is well adapted to synchronously program these reactive areas, while asynchronous communications are reserved for inter-molecules communications.

The API presented above is built on top of synchronous and asynchronous calls. Accessors (get prefixed methods) can proceed synchronously because a) they simply return the value of an encapsuled attribute (*getDegree*, *getNbAtoms*, …), b) they return a reference on a reactive object located in the same atom (*getLeg*, *getServoMotor*, …) or c) they concern reactive objects located within the same molecule (*getAtoms*, *getNeighbours*, *getLinkedLeg*). In these cases, the caller will resume its execution when the synchronous call returns. In case of inter-area call, the return proceeds at a different, a priori unknown, future instant. Modifiers (set prefixed methods) are non-blocking methods where the client continues its execution without waiting for the method to complete the order. Note that no value is supposed to be returned by modifiers, so we don't have to focus on future values to send back to the client. Modifiers are implemented using reactive statements: a new event is simply broadcasted to the local reactive machine to apply the current order.

5 DISCUTION ON A PROGRAMMING MODEL TO CONTROL ATOMS AND MOLECULES

Programming such reconfigurable structures is a very difficult task since the traditional method to generate moves cannot be applied because of the high redundancy of the system. To help the programmer to express reconfigurations on its topologies, the language we are currently defining deals with data-parallelism (14), symmetries of the topology, scalability and the distributed reactive object model of computation (15).

Programming models offer us two main different models of local computations (control of an atom) to exploit these assumptions: communicating asynchronous threads (or tasks or active objects) and the synchronous reactive model. Compared to the thread approach (16), the synchronous reactive model benefits of a light underlying execution model, a causality property that implies determinism, a built-in synchronization mechanism and a communication model based on event broadcast. One benefits from semantic properties that permit the programmer to control the inner parallelism (between legs into an atom) merely.

Global computations (control of a molecule) over the molecules implies to manage dynamically reconfigurable scalable structures. The problem with the synchronous reactive model is that is was not specifically designed to support scalable structures. Embedding scalable structure in a synchronous data-parallel model combines the advantages of the synchronous and asynchronous parallel style. The data-parallel model offers a way to express sequences of parallel computations on scalable data-structures. Each atomic entity in the structure can be enabled/disabled to perform the current statement on its local data. The underlying execution model support either massively parallel architectures, either a network of heterogeneous computers (including parallel ones) or a sequential computer which interleaves the statements intended to the enabled entities. The compiler has to fill the gap between a synchronous reactive, centralized and scalable expression and asynchronous distributed computations. For instance, to express walking over a row of atoms, we simply express the parallel computation of a distributed reactive walking program (fig. 5)

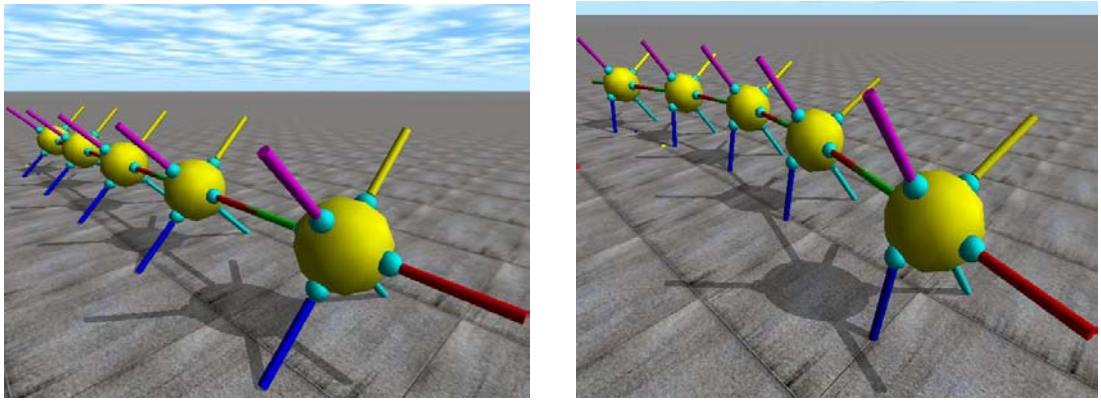

Fig. 5: a row of walking atoms under simulation

6 ANALYSIS AND DESIGN OF THE SIMULATION TOOL

In this section, we describe the motivations about the simulation engine we are designing. It offers an environment supporting meaningful code development that can easily be transferred to the real robots, realistic dynamics and collision model and compatible with the synchronous reactive model. To achieve these goals, two alternate ways are studied: extending our reactive objects with simulation capabilities or integrating our system on top of a high-fidelity physics simulation library.

6.1 Dynamics

For a physical simulation tool, a lot of Multibody Dynamics Software are available (17). Open Dynamics Engine (ODE) is a rigid body simulation engine initiated by Russel SMITH (18). It is reasonably documented, and has reached a maturity level: accurate methods are fast and stable. They are based on following works: Mirtich's method (19) for collisions, Stewart/Trinkle (20) and Anitescu/Potra (21) for integration and Baraff (22) for the solver. ODE is a cross platform development. This feature is interesting for a multiple team project. It was chosen in the Übersim simulator (23) for RobotCup robot soccer. The goal of Übersim is to create a simulation environment that enables control systems to be developed rapidly and transferred to the real system easily. A comparison between a simulated robot and a real robot validates the Übersim simulator.

Put together, ODE and Übersim meet the requirements for our simulation engine. Figures 2, 3 and 5 are screenshots of the simulation render in its current state. The event driven model with a global clock and the class design of Übersim is used to develop our simulator.

6.2 Centralized reactive implementation

The simulation process has to take in charge the notion of area presented in section 2. Each area manipulates a clock that permit to extend the synchrony property to interconnected reactive machines. Distinct area are not implicitly synchronized. A natural way is to extend our reactive objects with physical simulation capabilities. In (16), the reactive approach is applied in simulation and modelling of physical systems. A complex system can be represented as a set of linked but independent entities with a common discrete time. The influences of a physical entity on another one (forces between particles) are translated into events. The numerical algorithms of integration are adapted to have many parallel synchronized processes of integration communicating by broadcast events. Such a reactive simulator has two main advantages: modularity and support of distributed simulations. Since this research work is in an early stage of development and we deal with centralized simulation, we have chosen another approach : we simulate the distributed reactive machines embedded in the true atoms into the integration process.

To merge several reactive machines with distinct asynchronous clocks in the simulation loop, we simply interleave integration steps with tops of reactive machines. Fluidity and accuracy of the simulation is ensured by giving the simulation loop a high frequency. Tops of the reactive areas are alternatively started at different simulation step. It comes down to merge the reactive machines of the atoms that belong to a given area.

7 CONCLUSION

In this paper we have described our atomic component and its underlying execution model. We have discussed on a programming language exploiting concepts of the synchronous reactive and the data-parallel programming models. We have presented some results on a simulation tool based on ODE and Übersim.
Regarding the definition of the programming language, regular topologies like 2 or 3 dimensional grids, meshes, trees are easily supported by the data-parallel model. Some specificities of the dynamic reconfiguration of the topologies are under study regarding topologic languages like MGS (24).

REFERENCES

[1] http://www-2.cs.cmu.edu/~paredis/rmms C.J.J. Paredis, and P.K. Khosla, *Fault Tolerant Task Execution through Global Trajectory Planning*, Reliability Engineering and System Safety (special issue on Safety of Robotic Systems), Vol. 53, No. 3, pp. 225-235, Sept. 1996.

[2] http://golem03.cs-i.brandeis.edu H. Lipson and J.B. Pollack, *Automatic design and Manufacture of Robotic Lifeforms*, Nature 406, pp. 974-978, 2000.

[3] http://www.swarm-bots.org

[4] http://www.cs.dartmouth.edu/~rus/self-reconfig.html

[5] C. McGray and D. Rus, *Self-Reconfigable molecule robots as 3D metamorphic robots*, IEEE/RSJ, IROS conference Victoria, BC Canada, pp 837-842, Oct. 1998.
[6] http://www.isi.edu/conro, W.M. Shen and P. Will, *Docking in self-reconfigurable robots*, IEEE/RSJ, IROS conference, Maui, Hawaii, USA, pp 1049-1054, Nov. 2001.
[7] http://www-2.cs.cmu.edu/~unsal/research/ices/cubes C. Unsal and P.K. Khosla, *A multi-layered planner for self-reconfiguration od a uniform group of I-cube modules*, IEEE/RSJ, IROS conference, Maui, Hawaii, USA, pp 598-605, Oct. 2001.
[8] http://www.ai.mit.edu/~vona/xtal D. Rus and M. Vona, *A physical implementation of the self-reconfiguring crystalline robot*, ICRA conference, San Francisco CA, April 2000.
[9] http://www2.parc.com/spl/projects/modrobots
[10] http://staff.aist.go.jp/e.yoshida/test/top-e.htm A. Kaminura & all , *Self reconfigurable modular robot*, IEEE/RSJ, IROS conference, Maui, Hawaii, USA, pp 606-612, Oct. 2001.
[11] F. Boussinot, G. Doumenc and J.B. Stefani, *Reactive Objects*, INRIA Research Report RR-2664, Oct. 1995.
[12] J.F. Susini, L. Hazard, F. Boussinot, *Distributed Reactive Machines*, INRIA Research Report RR-3376, March 1998.
[13] http://www.univ-ubs.fr/valoria/Jacques.Malenfant/ARM, J. Malenfant and S. Denier, *ARM : un modèle réflexif asynchrone pour les objets répartis et réactifs*, , Actes de LMO 2003, Revue L'Objet, Hermès/Lavoisier, vol. 9, no 1-2, pp. 91-103, 2003.
[14] Luc Bougé, *The Data-Parallel Programming Model: a Semantic Perspective*, INRIA Research Report RR-3044, Nov. 1996.
[15] Albert Benveniste, Paul Caspi, Stephen Edwards, Nicolas Halbwachs, Paul Le Guernic, and Robert de Simone, *The Synchronous Languages Twelve Years Later*, Proc. of the IEEE, special issue on Embedded Systems, 2002, to appear.
[16] *Reactive Programming in Mimosa project main site*: http://www-sop.inria.fr/mimosa/rp
[17] J. Trinkle, *Trinkle's Survey of Multibody Dynamics Software*: http://www.cs.rpi.edu/~trink/sim_packages.html
[18] R. Smith, *Open Dynamics Engine main site*: http://opende.sourceforge.net
[19] Brian Mirtich, *V-Clip: Fast and Robust Polyhedral Collision Detection*, ACM Transactions on Graphics, Vol 17, No 3, pp 177-208, July 1998.
[20] D. Stewart and J. Trinkle, *An implicit Time-Stepping Scheme for Rigid Body Dynamics with inelastic collisions and Coulomb Friction*, International Journal of Numerical Methods in Engineering, 1996
[21] M. Anitescu and F. Potra, *Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementary problems*, ASME Nonlinear Dynamics 14, pp 231-247, 1997.
[22] David Baraff, *Issues in computing contact forces for non penetrating rigid bodies*, Algorithmica, pp 292-352,Oct. 1993.
[23] B. Browning and E. Tryzelaar, *Übersim: A Multi-Robot Simulator for Robot Soccer*, Autonomous Agents and Multi-Agent Systems (AAMAS'03), under submission.
[24] J.L. Giavitto and O. Michel, *MGS: a Programming Language for the Transformations of Topological Collections*, LaMI technical report N° 61, May 2001.