Abstraction of the Communication in a Team of Robots with a Language Approach

Claude Gueganno, Dominique Duhaut

Proceedings of the 2006 IEEE
International Conference on Mechatronics and Automation
June 25 - 28, 2006, Luoyang, China

Abstraction of the Communication in a Team of Robots with a Language Approach

Claude Guéganno and Dominique Duhaut
VALORIA
University of South Britanny
Lorient, France
{*claude.gueganno, dominique.duhaut*}*@univ-ubs.fr*

***Abstract*— This paper presents, first, the `maam` [1] project which is a self-reconfigurable robotic architecture where each module is autonomous for energy and CPU. This robot provides a full software/hardware micro-architecture that underlies a language abstraction for command/control purpose. Because the synchronization between modules (or between groups) is a crucial aspect (docking, synchronized movements ...), we focus here on the communication aspects, in particular how we abstract them with a language based solution.**

***Index Terms*— Multi-agent, self-reconfigurable, communication, ad-hoc, language.**

I. Introduction

This project takes place in the more general fields of reconfigurable modular robotics. In these fields of research, we can mention several various experiments. The M-TRAN (Modular Transformer - AIST) described in [2], is a distributed self-reconfigurable system composed of homogeneous robotic modules. CONRO (Configurable Robot - USC), is a robot made of a set of connectible, autonomous and self-sufficient modules [1]. ATRON , is a lattice based self-reconfigurable robot [4], and also, PolyPod (Xeros) [3], I-Cube (CMU) [5]. These robots generally consist of modules working together and where each module is permanently linked to at least one other.

In the `maam` robot, the module can be linked together (making up a chain) or can be completely autonomous for motion, energy and processing unit. Some inspiration of the `maam` robot comes from the biological world of the ants: each one has a certain autonomy, but they can help each other for particular tasks (*e.g.* making bridges). The `maam` robot consists of several autonomous entities, called "robotics atoms" due to their physical look (see Fig. 1).

Our approach is organized along three main axis. The first axis concerns the distribution of the embedded command/control system between hardware and software. We systematically implement the low-level loops in a FPGA[2]. That allows us to use the resources of the CPU for coordination and programming of missions. More about this hardware can be found in [12].

[1]Molecule = Atom | (Atom+Molecule)
[2]Field Programmable Gate Arrays

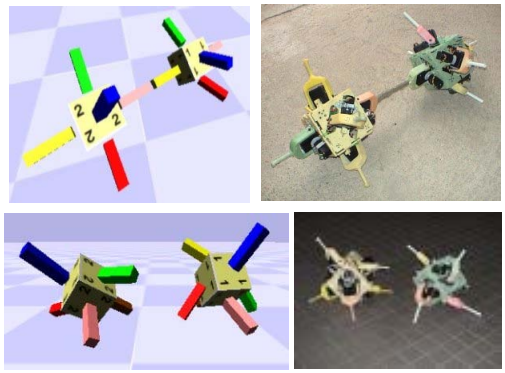

Fig. 1. **Simulation and real robots.** Top: two robotic atoms are walking. Bottom: the two atoms are linked and walk synchronously.

The second axis is to propose an universal abstract model for the robots and more generally for mechatronical systems. Each robot embeds an XML-description. When the host detects a robot, it must first request for its XML-interface. Exported on the host system, this file permits the automatic configuration of miscellaneous tools, for direct control/command, for task programming since parsing this file allows the host to learn the particular part of the remote robot. This aspect opens up our architecture to *ambient robotic* ([15]). Thus, the tools remain the same ones whatever the robots or the mechatronical systems are.

The last point aims at designing a flexible multi-robot system open either for full remote control or for temporary autonomous control. Moreover, we require that a robot can become master in a team. Our proposition is that the higher-level layer of the embedded software is an interpreter including native instructions for general communication purpose, and remote control/command of identical agent.

The following two examples shows the importance of the communication here. Assume a situation (see Fig. 2) where a group of n robots has to go from a point to another. To keep control of the group, this latter must sometimes give up an agent that becomes a radio relay. Another case is the reconfiguration pictured by the figure 3: during the reconfiguration, the team of robots is divided in subsets. The

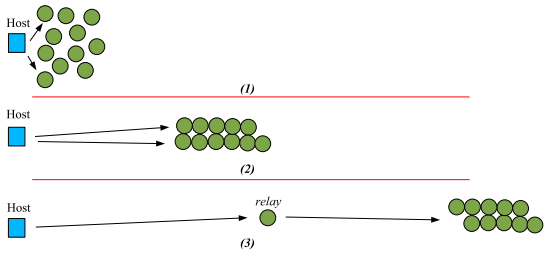

Fig. 2. **Displacement with safeguarding of contact with the host.** (1), the robots start the mission, (2) they use wireless communication for re-configuration and motion synchronization, (3) one robot acts as a relay for global communication between the control system and the others

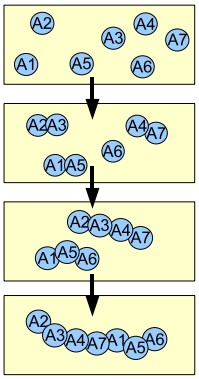

Fig. 3. **Reconfiguration.** Making up a chain involves parallel activities, local communication for docking and global communication for planification.

number of subsets and the number of elements in each one changes after each stage.

II. Overview of the maam project

This section is a quick presentation of some mechanical aspects of the basic module (atom) and next, a description of the hardware and embedded software.

A. Mechanical and hardware design

A robotic atom is composed of six legs which are directed towards the six orthogonal directions of space. They allow the atom to move itself and/or dock to another one. The frame of the atom consists of six plates molded out of polyurethane. A frame weights approximately 180g. The first walking

Fig. 4. Prototypes of the maam robot. They embed all the electronic and software described in this paper, but do not include the pincers.

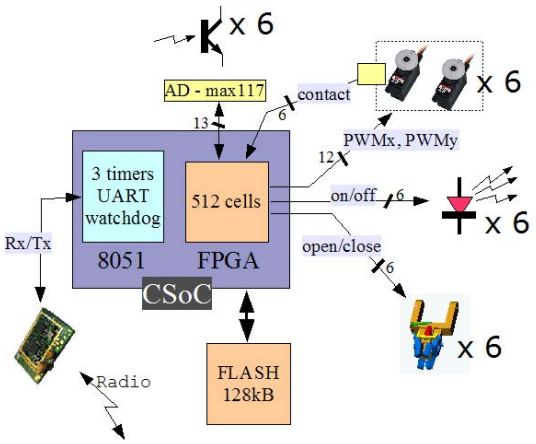

Fig. 5. **Embedded electronics an peripherals,** is built around a Triscend TE505 CSoC. The TE505 integrates a CPU 8051, a FPGA with 512 cells and an internal 16KB RAM. It is completed by an AD convertor component, 64kB of flash memory and an external *bluetooth* module for radio–communication

Fig. 6. **Embedded electronic:** CPU, extension card and industrial *bluetooth* module.

prototypes of atom are shown on the Fig. 4. The pincers are under development but are not integrated at this time. Each leg is driven by two servo-motors, and is fitted with an infrared transmitter/receiver. Some signals of the servos are processed in order to identify the legs at the touch of the ground. The embedded electronic is built around a CSoC (Configurable System on Chip) which integrates a micro–controller and a FPGA. Only 80% of the 512 cells of the FPGA are used to generate the twelve PWM-commands, the command for A/D converter (MAX117) driven in a pipeline mode and all basic inputs/outputs. The schematic of figure 5 summarizes the requirements of maam robot and the internal architecture of the CPU. All the electronic is embedded in a robotic atom in a cube which edges $\leq 50mm$.

B. Hardware for communication

Among the wireless technologies, *Bluetooth* gives us suitable responses for noise constraint, miniaturization of modules and low cost. In a *Bluetooth* network, all the units are identical for the hardware and also for the software interface. The only difference is the 6-byte address of each one. This

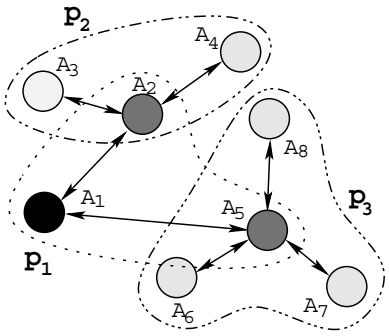

Fig. 7. **A network of atoms with *Bluetooth***. The atoms A_1, A_2 and A_5 make up a *piconet* p_1 whose master is A_1. Slaves A_2 and A_5 are the respective masters of *piconets* p_2 and p_3. The 8 atoms make up a *scatternet*.

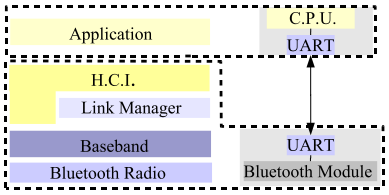

Fig. 8. **Bluetooth layers and implementation in the hardware.** The left side shows the bluetooth layers that we use in our robots: the application is written above the HCI layer, so we can control the complete bluetooth device. The right side shows the corresponding hardware: between HCI and application, we use an UART interface.

property is very interesting because the software developed for communication purpose on a centralized control-system can be transposed on robots with few modifications.

In the *Bluetooth* technology, when a module establishes a connection with another one, it becomes the *master* during the communication. A master can have up to seven open links at the same time The set of the slaves and the master is called a *piconet*. One slave of a *piconet* can be the master of another *piconet* (see Fig. 7).

Building an application just above the HCI[3] layer allows us to control the complete *bluetooth* device (link management and baseband). HCI packets are send from a *bluetooth* device to another one, containing either user data or command. All about HCI commands can be found in [7].

All layers from radio to HCI are implemented in the industrial module `Bluebird` constructed by `Inventel` [8]. This module is connected to the CPU via an UART interface (see Fig. 8). The host system uses exactly the same module.

C. Embedded software

The software embeds both a communication manager an an implementation of the abstract interface which was given for the robot. The main program runs an interpreter. The communication-manager can start, stop or replace on the

[3]Host Control Interface

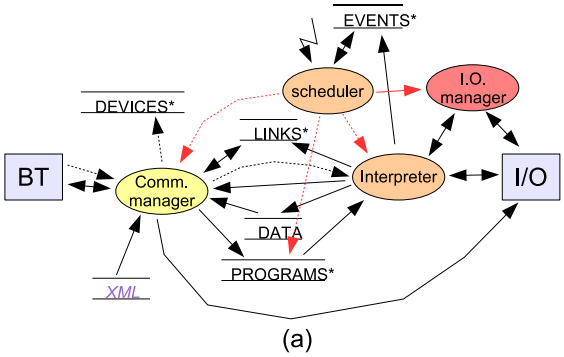

Fig. 9. **General architecture of the embedded software.** (a) Data-flow view: the communication manager receives orders and new programs. It also manages *bluetooth* connections and disconnections. The program interpreter is activated or stopped by the communication manager. (b) Layers view: the basic functions of the robot are reachable either from the instructions of interpreted program or directly from remote invocations incoming from other robots or hosts.

fly the program in progress. It also makes it possible to reach the basic functions of the robot. An overview of the embedded software is given by the Fig. 9. The interpreter and the communication manager are strongly connected.

III. Communication: Requirements and Abstraction

The communication is an inescapable aspect in a multi-agents architecture, and is the backbone in our architecture. Several reasons plead in favor of a wireless communication:

- effectiveness for the development: it becomes easy to replace the running program on the fly without manipulation; possibility of remote-control for parameter adjustment, and even of development on line in real-time with a real robot.
- cooperation of the robots: the wireless communication makes possible the implementation of distributed algorithms, even when the robots are separate and not line in sight.
- Constitution of a network of robots: The vocation of the robots an particularly in reconfigurable architectures is the exploration of difficult grounds. The radio communication allows to keep control on the whole of the robot.

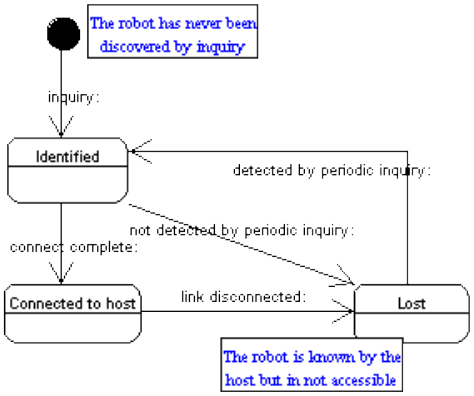

Fig. 10. **Robot states.** The initial state is defined at the beginning of the application: physical robots exist but are not known by the program which must inquire after them. During the execution, some robots can temporarily disappear or some other ones appear, so the host has to inquire periodically after robots.

We will now present general considerations about the requirements of communication and next the implementation in the `maam` micro-architecture.

A. Generalities

An atom must communicate with one or more of its neighbors, and/or with a host–system. Moreover, in the final application atoms must communicate inside little independent groups (for example by pairs of atoms when docking or by three when constituting an hexapod ...).

Each robot (or atom) may be represented by a finite state automate $\mathcal{A}_i$, and the set of agent by a directed graph $\mathcal{G}$ where the vertices are the previous automates and where the edges represent the radio-links between agents. The graph is directed because links are not symmetric due to technical reasons (master/slave connection), an edge

$$\mathcal{E}_{ij} = \mathcal{A}_i \rightarrow \mathcal{A}_j$$

means that $\mathcal{A}_i$ is the master and $\mathcal{A}_j$ the slave. Two agents can connect or disconnect each other when they receive such an order but, disconnection can occur randomly:

$$\mathcal{E}_{ij}(t) \in \{0, 1\}$$

The possible states of each $\mathcal{A}_i$ are (at least): *identified* (the agent is reachable in the working area), *connected* (the agent can receive orders and send data, the host can be a central system or one of the atoms), *lost* (for unspecified reason). The Fig. 10 describe these states and the transitions between them. This give the following graph:

$$\mathcal{G} = \{(\mathcal{A}_k)_{1 \leq k \leq N(t)}; (\mathcal{E}_{ij})_{1 \leq k \leq N(t)}\}$$

where $N(t)$ is the number of recognized agents at the instant t.

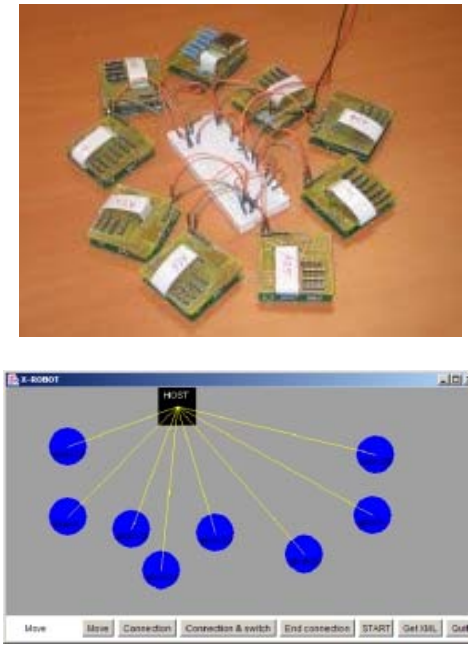

Fig. 11. **Eight CPU are communicating with the host.** Basic commands, and/or request for XML description can be performed.

Achieving the communication layer in this multi-agents application involves to have a real-time representation of $\mathcal{G}$ *i.e.* to listen for all asynchronous events.

B. Centralized management on host system

The planification of a mission often involves a global view of the team of robots. Using *bluetooth* allows three solutions:

1) the modules inquire after the host and request for a connection link when they found it. In this case, the host is a multi-session server. One disadvantage is that robots must know the bluetooth address of the host (or its user-friendly name), and we cannot easily change the host.
2) broadcast messages for all robots in the bluetooth area[4].
3) establish dedicated links with modules that have been detected in the bluetooth area. The link is possible only if the robot is waiting for a connection. The number of simultaneous links is theoretically seven, but inverting master/slave role after the connection with some robots allow to increase the number of simultaneous connections (Fig. 11).

On the host system, we have implemented the communication middleware in a `java` class called `SetRobot`. `SetRobot` includes a thread that searches periodically for new robots in the *bluetooth* area. Robots can produce events (incoming data, link disconnected ...) so the `SetRobot`

[4]According to bluetooth specifications[7], broadcasting is possible, but for the moment the firmware of our module don't take it into account.

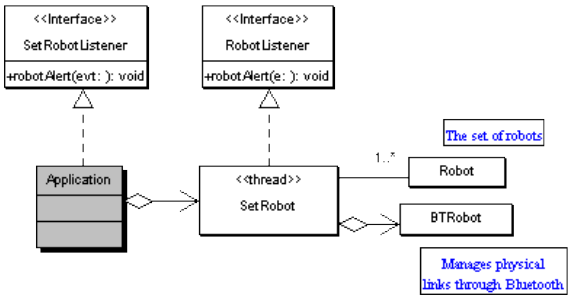

Fig. 12. The application uses a `SetRobot` object that abstracts the communication middleware.

object listens to them an forward them towards the application. For driving robots, an application has to instantiate a `SetRobot` class and to listen to `SetRobotEvent` incoming events. `SetRobot` provides operations for connecting, disconnecting, transmitting data (Fig. 12).

We have experimented the connectivity with a set of twelve robots $\{A_1, A_2, \cdots, A_{12}\}$. The fig. 13 describes one of the experiments. After some inquiries, the host creates links with several modules. It orders some of them to connect with another one to make a chain. When a link is established, the host can request for the standard XML description of the robot (assuming that the CPU drive a robot). When it has received the XML file, it instantiates an interface dedicated for this specific robot. More about the XML interface can be found in [15]

The effectiveness and the robustness of the communication have been tested in experiments like in fig. 14. The program running on the host establishes the connection with the robotic atom, and then uses it for scanning the scene. The host builds an image with the data requested to the robot. Thousands of exchanges were done without any loss. The same experiment was achieved with simultaneously four robots.

C. Abstraction of the communication inside the robot

The communication manager embedded in each robot is in fact a set of two finite state automates. The first is dedicated for links management, and the second realizes periodic inquiries in order to update a table of reachable devices.

We propose a full integration of the communication aspects in the local language of the robot. That involves that the usual services like connection, event on disconnection, wait for a connection and signals management constitute a part of the set of instructions in the local language. These instructions are illustrated by the example of figure 1 (bottom). Two atoms are walking synchronously. The first runs the following program:

```
SET(Id,open(0,7,58,0,39,155)); # open a link
*[TRUE](
     SIG(33,Id); # send signal to Id
```

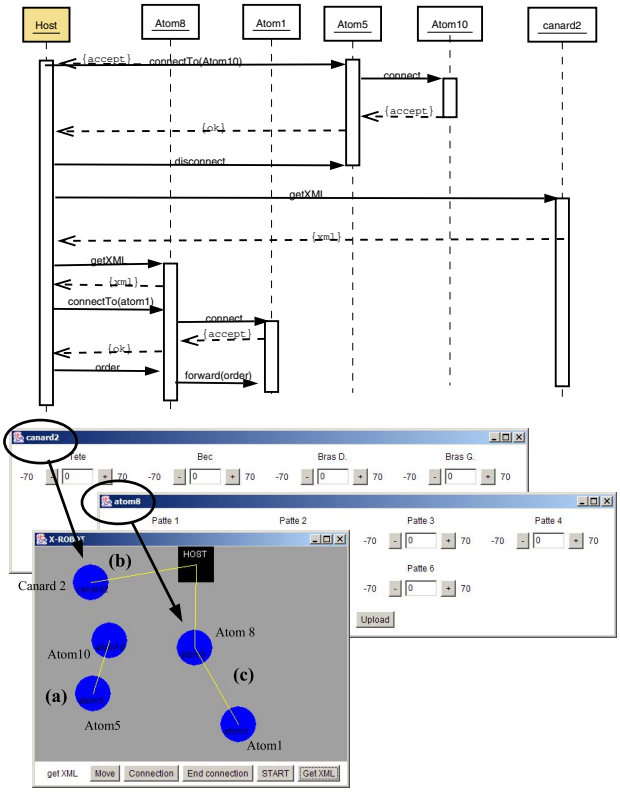

Fig. 13. **Links management.** This is a screen shot of an experiment where 5 robots were detected at this instant. The sequence diagram traces the successive actions. In the final state, two robots build up an isolated network (a); one is only connected to the host and has given its XML interface (b); at last, two others make a chain connected to the host (c), the robot directly connected to the host as also given its XML interface and forwards its orders to its slave.

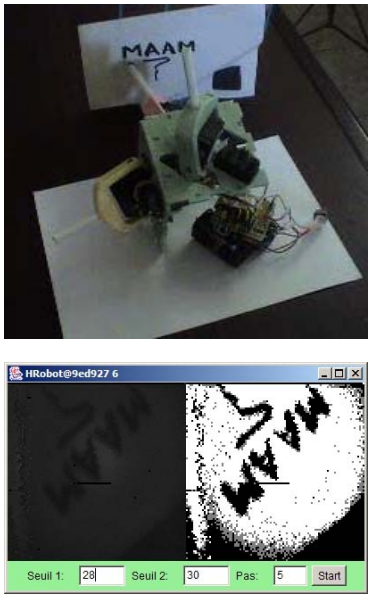

Fig. 14. **Communication between host and robot.** The host scans the space through the sensor of the robot.

Fig. 15. **Communication between two robots.** The left robot commands the other with remote invocations, during the alignment of legs.

```
     # ACTION
     ...          # walk
);
```

The second atom runs the following program:

```
SET(Id,WCX()); # wait for a connexion
*[TRUE](
     SYN(33); # wait a signal from semeone
     # ACTION
     ...       # walk
);
```

So, the exchanges between the two robots are reduced since they are limited to send (or receive) a signal.

Another point of the local language interpreted in the atom is the possible control/command of other *identical* agents. Assume that the set of IO instructions of the agent is

$$\mathcal{E} = \{\texttt{INS}_k\}_{k\in[0,N]}$$

then, this agent can appy all these instructions to a remote identical agent, after a successful connection. So the instructions of the set

$$(r \circ \mathcal{E}) = \{\texttt{rINS}_k\}_{k\in[0,N]}$$

are implicitly integrate part of the language. They take as first parameter the identifier of the target, and the normal parameters of $\texttt{INS}_k$ afterward. The figure 15 pictures an experiment where this property is widely used. In the `maam` robot, each leg is equipped with an infra-red transmitter/receiver to carry-out alignment and approach preliminaries before docking. We solve this problem easier thanks to remote invocation. Indeed, the communication makes possible for a robot to know, in real-time, the effect produced by an elementary displacement by questioning the sensor of the other robot, and to act on its affectors. So we can simply work in a closed-loop. The following code is extracted from the original program running in one of the atoms. The second one do nothing by its own.

```
...
SET(Id,open(0,7,58,0,39,171));
//(PwmA(4,X);,PwmB(4,Y);,);    # local PWM output
rPwmA(Id,4,AX); # remote PWM output
rPwmB(Id,4,AY);
...
*[INF(can4,15)](  # local infrared sensor
   [OR(SUP(AX,45),INF(AX,-45))]((
            INC(AY);
            rPwmB(Id,4,AY); # remote PWM output
   ));
...
```

IV. Conclusion and future works

We presented here a design for cooperation in a robotic system composed of several identical units. A language approach underlies our realization, including the communication aspects. Managing the team of robots can be done either by direct commands from the host or by uploading local programs in all (or some) of the robots. Each robot can access the hardware of each other so the collaboration in the team can be a cooperation between subsets of robots. The software/hardware partially presented in this paper must now be linked to the decision level. The `HoRoCoL` multi-agent language ([16]) has been designed for that, and the `java` software introduced here could underlie a part of its implementation.

Acknowledgment

The `maam` project is supported by the Robea project of the CNRS. All references to people participating to this work can be found in [10].

References

[1] M. Rubenstein, K. Payne, W-M. Shen, "Docking among independent and autonomous CONRO self-reconfigurable robot" in ICRA 2004

[2] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, S. Koraji, "M-TRAN: Self-Reconfigurable Modular Robotic System" in IEEE/ASME transactions on mechatronics, Vol.7 No.4 2002

[3] http://robotics.standford.edu/users/mark/polypod.html

[4] M.W. Jørgensen, E.H. Østergaard, H. Hautop, "Modular ATRON: Modules for a self-reconfigurable robot" in proceedings of 2004 IEE/RSJ International conference on Intelligent Robots an Systems (IROS 2004).

[5] I-cube CMU

[6] A. Kaminura et al, "Self reconfigurable modular robot", IEEE/RSJ, IROS conference, [Maui, Hawaii, USA, Oct 29-Nov 03, 2001 p 606-612]

[7] http://www.bluetooth.com : Bluetooth 1.1 specifications, (Part H: "Host Controller Interface functional specification")

[8] *Bluebird 2* Datasheet, Inventel, 2003.

[9] A. Das, J. Speltzer, V. Kumar, C. Taylor, "Ad Hoc Networks for localization and control", 41st IEEE Conference on Decision and Control (CDS 2002) [Vol. 3, p. 2978-2983]

[10] http://www.univ-ubs.fr/valoria/duhaut/maam

[11] D. Duhaut *"Robotic Atom"* CLAWAR, Paris, septembre 2002

[12] C. Guéganno, D. Duhaut, "A hardware/software architecture for the control of self reconfigurable robots", June 2004 [DARS-2004, France]

[13] V. Montreuil, Dominique Duhaut, Alexis Drogoul, "Study of the communication capacities in a reactive algorithm for self-reconfigurable robotics", submitted

[14] N. Brener, F. Ben-Amar, P. Bidaud, "Analysis of self-reconfigurable modular systems: a design proposal for multi-modes locomotion", ICRA 2004

[15] C. Gueganno, D. Duhaut, "Remote tools using wireless communication applied to `maam` project", CIRA 2005, Ospoo (Finland)

[16] Y. Le Guyadec, C. Guganno, M. Dubois and D. Duhaut, "Using HoRoCoL to Control Robotics Atoms", ICMA 2005, Niagara Falls (Ontario).

[17] http://www.w3.org/XML/