Distributed Robotic: a Language Approach

Claude Guéganno[1] and Dominique Duhaut[2]

[1] University of South Britanny – France claude.gueganno@univ-ubs.fr
[2] University of South Britanny – France Dominique.Duhaut@univ-ubs.fr

Summary.

In this paper we present a powerful and versatile architecture dedicated for robotic and mechatronic systems. The originalities of our study are, (i) that we consider a robot with a dynamic and explicit language approach and (ii) that the communication aspects are abstracted and take place as a natural part in the language. This approach allows easy transfers towards other fields of research like network of sensors, ambient intelligence and ubiquitous robotic. These works concern low-cost micro-system easy to embed in little mechatronic devices. For demonstration of the effectiveness of our architecture and developing tools, we have implemented it in the `maam` robot which is a reconfigurable robot composed of several modules autonomous for CPU, energy and motion. Some results can be found in the last part.

1 Introduction

Distributed architecture for control are widely studied ([1],[2],[3]) but generally address system fitted with operating system and network functionalities. The promising field of ambient intelligence with its variation in ambient and ubiquitous robotic has similar requirements but with reduced capacities. In the field of collective robotic, swarms or teams of robots or even in reconfigurable robotic with autonomous units, some of these requirements are: to distribute the roles for a mission, to take unforeseen events into account, to localize neighbors, to report status to the host. The behavior of the group must be reactive inside a planification itself function of time. Two examples of collaboration are shown in figure 1. In the first one, the robots are associating to make a chain in order to move together, in the second one, the chain is moving, but some agents are left as radio-relays in order to keep a data link with the host. So, the agents must be able to collaborate autonomously to achieve the mission or a part of it. In some cases, the robots must work

without any external help (host unreachable). Obviously, the communication aspect is very important in such systems. To take full requirements into account we generally need line in sight communication for solving localization and docking problems and, on an upper layer a network communication for full control purpose including request of reachable robots, broadcast of new plans, control/command of subsets of robots ...

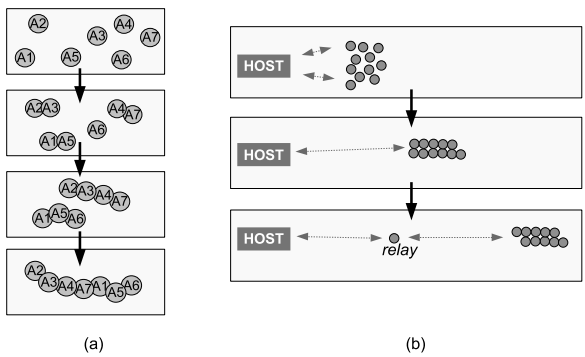

Fig. 1. Two kinds of collaboration. (a)The agents are building a chain. (b)They scatter some of them to ensure a communication between the host and all of them.

So we propose an architecture which gives answers to the following constraints:

- taking into account a centralized control system;
- distributing the intelligence between agents;
- giving possible autonomy for each agent, or subsets of agents;
- allowing collaboration between agents (autonomous group);
- reacting at events by possibly broadcasting new plans towards agents or subset of agents;
- each agent may control a group and in the same time be controlled by another entity.

Moreover, this architecture must not decide of the kind of control which will be apply to the society of agent. It could be deliberative or reactive, or one after the other.

Our proposition is that each agent is likened to a local language, so, controlling it involves (i) to request its own language (or, at least, its set of instructions) (ii) to generate and upload programs according to its particular capacities, and/or to remote-control it directly.

The next section presents the embedded part of this architecture.

2 An architecture for ambient and distributed robotic

2.1 External overview

As we have seen, managing a team of robots with effectiveness suppose to guess the actual robots reachable in the area, to learn the capabilities of each one, to be able to distribute an application between them, and, possibly to remote-control any of them.

The figure 2 summarizes the functionalities of our architecture, and show how we can use it from a control point of view. The first important aspect is that the control system is not supposed to know who are the robots in its environment. It has to inquire for them, an to achieve a mission, to compose with the robots which are actually presents and ready to use (Fig 2-a). Since robots could be different (family, or version in a family) the host requests for the identity of each of them (Fig 2-b). This is done by downloading an `XML` description from the robots. After a parsing stage, the host can generate dedicated user-friendly tools for each agent for remote control (Fig 2-c) and also programming tools (Fig 2-d). Indeed, the `XML` file permits to reconstitute the particular language of the robots, and also gives informations about remote invocations allowed towards the robot. So new programs can be uploaded in the agents (Fig 2-e). A new program replaces the current one instantaneously. All the previous operation from inquiry to uploading can be chained in a standalone program (Fig 2-f).

2.2 Internal overview

Embedded architecture

The embedded architecture is built around a configurable system on chip including a FPGA based hardware easy to adapt for many mechatronic and robotic applications and a micro-controller. It communicates with an industrial *bluetooth* module driven at the Host Control Interface level. More informations about the hardware can be found in [4].

The software include (i) a communication manager made of two finite state automates, one dedicated for permanent inquiry of neighbors, and the other for general communication purpose (remote control, up/download of files); (ii) an event scheduler that can launch some particular instructions of the main program; (iii) an interpreter which runs the main program of the agent (figure 3-left). The IO functionalities of the robot are directly accessed trough the communication interface as well as from the interpreted program (figure 3-right).

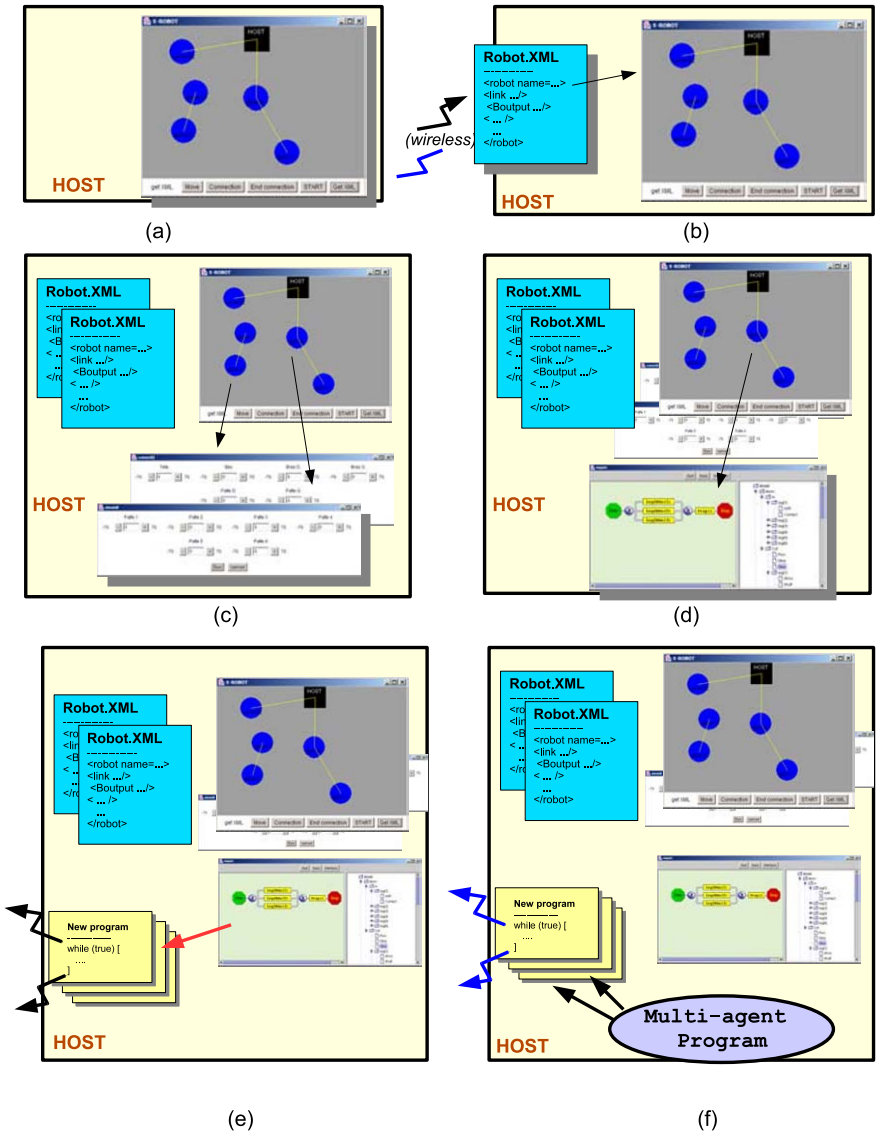

Fig. 2. Ambient behavior. (a) The host inquires for robots in its area. (b) The XML description of each robot is requested by the host. all the robots send their own XML files. With these informations the host (c) will be able to generate direct command on one robot or programs over a set of robots. (d) While the XML description contains elements of the local language of the robots, programming tools can be build. (e) New programs in local language are upload in the robots. (f) All these operations can be centralized in a global multi-agent program.

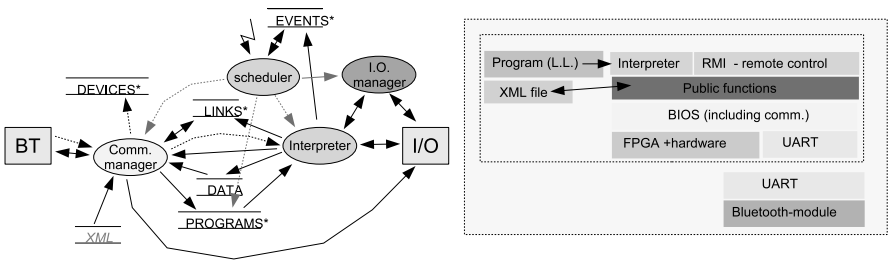

Fig. 3. BIOS. Data flow view (left) and layered view of the embedded software.

2.3 The highest layer: an interpreter

General structure of a program

A program is composed with a list of event, a list of variables and a list of instructions. Some of the instructions are specific to the agent and coincide with the embedded XML description. The control structures are the same.

Example

```
{K=0}
<SUP(can6,10):On2(1);>
<SUP(can3,20):On2(2);>
SET(K,-50);
EVT();
*[TRUE](   WAIT(50); INC(K);
           [EQU(K,50)]((SET(K,-50)!(PwmA(1,K);));
       );
```

In this example, two events are considered. `can6` and `can3` are internal registers implemented in the language. In this case they are the values of analog inputs, so these events aim at controlling the threshold of two sensors. The instruction `EVT()` starts the event scheduler.

Overview of the language

The syntax for instructions is the same as in many languages. All parameters and return values are integers. All basic operators are provided: arithmetic (`ADD`, `SUB`, `INC`, `DEC`, `MUL`, `DIV`); relational (`INF`, `SUP`, `EQU`, `NEQ`, `LEQ`, `GEQ`); logic (`AND`, `OR`, `NOT`) and affectation `SET`.

Control structures

The following examples gives three usual situations:

`{X=255, K=0}`	var: X ← 255 and K ← 0
`100*( PwmA(1,X); WAIT(10); DEC(X); );`	repeat 100 times
`*[LEQ(K,127)]( SET(K,Analog(1)); );`	while K(analog input)≤127 do ()
`*[TRUE]( On(1); Off(1););`	while (true) do ()

The instruction `BREAK` can end a loop, and the `CONT` (for *continue*) redirect the execution to the previous test.

The conditional instruction has the usual structure, as shown in the following example:

```
[EQU(X,0)](  (On(1);WAIT(10);)     // executed if X==0
           !(Off(1);WAIT(20);)     // else ...
         );
```

2.4 Instruction for communication

The major originality of this local language is to incorporate special native instructions for communication. They can be divided in four groups:

1. Control of local unit:
   `RSTBT();` → resets the *bluetooth* module;
   `ST("name/message");` → changes the user-friendly name of the *bluetooth* module; since this string can be read without opening a link between agents, we use it as a mail-box. An example of name should be `"a1/2,6,7,9"` meaning (i) that the name of this robot is `"a1"`, and (ii) that from `"a1"`, the four robots `"a2"`, `"a6"`, `"a7"` and `"a9"` are reachable. So the host can initiate an *ad-hoc* graph without any connexion at the application level.
2. Link management:
   `SET(Id,OPEN(address));` → opens a data link between the local module and the given address. The link number is stored in the variable `Id`.
   `WCX();` → waits for a connection, this agent has nothing else to do.
3. Signals management:
   `SYN(K);` → waits for the signal `K`, incoming from any agent;
   `SIG(Id,K);` → sends the signal `K` towards the opened data–link `Id`.
4. Remote invocations: assume that the set of IO instructions of the agent is

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

   then, this agent can apply 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 identification of the target. The figure 4 shows an example of distributed action where a robot controlled by the host (planification level) is ordered to control three other robots. We can note the local instruction `PwmA(x,y)` and the remote invocation `rPwmA(Id,x,y)`, where `Id` identifies an opened data–link.

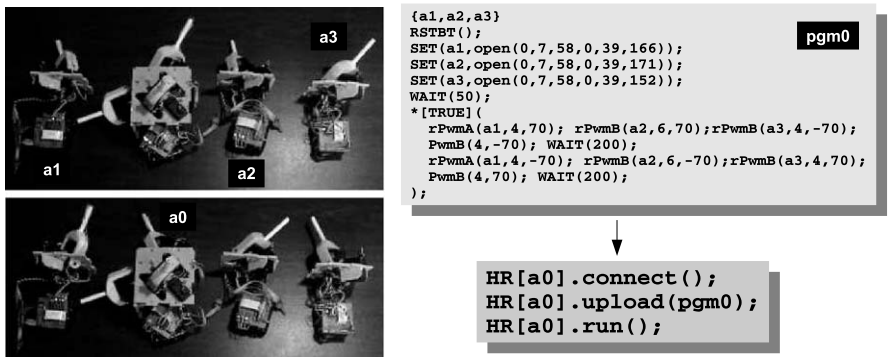

Fig. 4. Communication between agents. The host uploads a program in the agent a0 and launches it. Then, a0 take control of a1,a2 and a3.

2.5 Inner XML card

A DTD for mechatronical systems

The DTD is a set of grammar rules specifying the document generic logical structure. Thus in the DTD we enumerate all generic operations that concern the robots. Because we want easy transfers of this technology, we impose that (i) the number of kind of operation must be limited, and; (ii) the description can induce an object vision of the robot (a robot can be *composed* of subsystems themselves decomposable ...).

According to the first constraint, we propose only four kinds of functions:

$$\begin{cases} \text{binary outputs} \\ \text{outputs in a range} \in [min, \cdots, max] \\ \text{inputs} \\ \text{general procedures and functions} \end{cases}$$

A response for the second constraint is to add a concept of entity in the DTD and provide links between entities. So we can associate an operation with one of its subsystem. Because robots are often composed of many identical subsystems an entity can be link to n sub-systems (ex: hexapode, composed of 6 legs) Here is an extract of this DTD.

```
<?xml version='1.0' encoding='utf-8'?>
<!-- interfaceRobot.dtd -->
<!ELEMENT Robot (Link*,BOutput*,Output*,Input*,
                                Function*, Load?)>
<!ATTLIST Robot name CDATA #REQUIRED>
<!ATTLIST Robot BT CDATA #IMPLIED>

<!-- Logical link between objects -->
```

```
<!ELEMENT Link EMPTY>
<!ATTLIST Link name CDATA #REQUIRED>
<!ATTLIST Link parent CDATA #REQUIRED>
<!ATTLIST Link nb CDATA #REQUIRED>

<!ELEMENT Output EMPTY>
<!ATTLIST Output name CDATA #REQUIRED>
<!ATTLIST Output min CDATA #REQUIRED>
<!ATTLIST Output max CDATA #REQUIRED>
<!ATTLIST Output prot CDATA #REQUIRED>
<!ATTLIST Output parent CDATA #IMPLIED>

  ...
```

The `name` attribute concerns all the operations and entities of the robot. The `Link` element relies one system to n subsystems. When an operation addresses a subsystem, it fills the `parent` field.

The `prot` field appears for each operation element. Since we want to provide remote control tools and allow execution of distributed algorithms in a team of robots, we must access to all the internal basic functions of each robot. So this required field is necessary for defining the protocol of communication between the host and the robot.

XML example

The following code is an example of `XML` description using the previous `DTD`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Robot SYSTEM 'interfaceRobot.dtd'>
<Robot name="H2" BT="07003F273A">
  <Link name="leg" parent="H2" nb="6"/>
  <Output name="PwmA" parent="leg" min="-115" max="115" prot="A"/>
  <Output name="PwmB" parent="leg" min="-115" max="115" prot="B"/>
  ...
```

The control system of the team of robots requests for all the `XML` files of the detected robots, and parse them in order to generate instructions for them. By reading this file, we guess that the robot `H2` is composed of 6 legs, each of them is powered by two `PWM` commands. The next section shows the use of these informations.

3 Implementation in maam robot

3.1 Overview maam project

The maam [3] project is a self-reconfigurable robotic architecture where each module is autonomous for energy and CPU. The basic unit (called atom) is composed of six legs which are directed towards the six orthogonal directions of space. They allow the atom move itself and/or couple to another one. The first walking prototypes of atom appears on the figure 5.

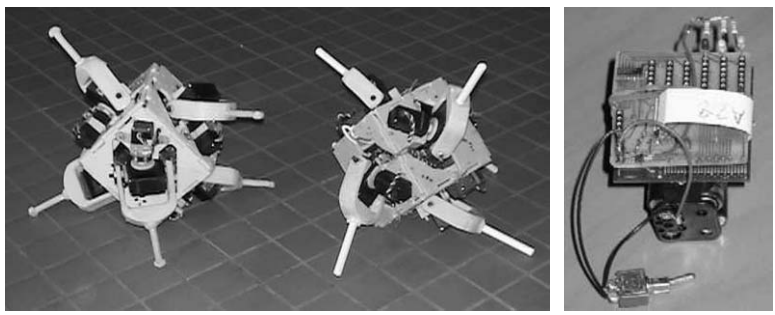

Fig. 5. The first prototypes of maam robot walking (right). These prototype embeds all the electronic and software functions described in this paper. They do not include the pincers.

The twelve PWM signals, and the command for the analog converter (driven in pipeline mode) are built in the FPGA. Some internal signals of the servo are processed in order to identify the legs in contact with the ground. Each leg is fitted with an IR transmitter/receiver for perception and docking. All the features described in this paper (XML inside, local language, wireless properties) are implemented and tested.

Because this robot do not take any inspiration from human or animal world a realistic simulator is written to study their behavior. The virtual robots have the same properties than the real robots and, moreover, are also driven by identical interpreted programs. The figure 6 reports a simulation were eight robots are searching for an attractor.

4 Conclusion

In this paper we presented an architecture for distributed robotic, completely abstracted by a language approach. Moreover, we proposed that every agent is fitted with an inside XML card, and that tools for control/command (including the dedicated local language) can be guessed thanks to this description.

[3] Molecule = Atom | (Atom+Molecule)

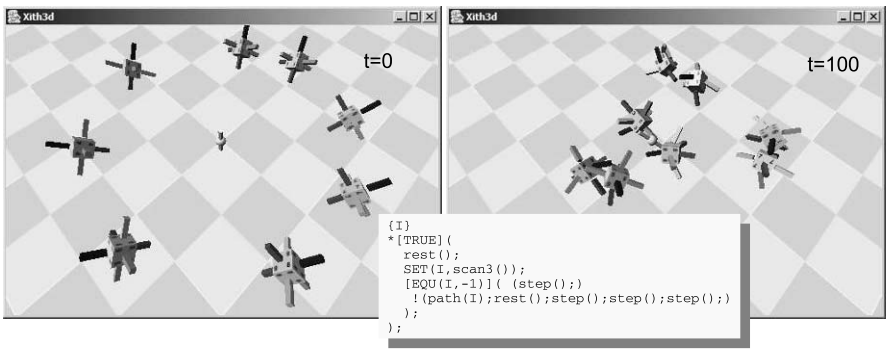

Fig. 6. Simulator. Like in the real context, the agents are independent and their own thread runs a program written in the interpreted local language.

The usual requirements in synchronization an remote invocation from one agent to another appear as a natural part in the language, since the middleware in charge of communication acts as a background task. The efficiency of this approach has been proved during its implementation in the `maam` robot which is a complex mechatronic system.

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 [5].

References

1. Raja Chatila, "Control Architecture for Automous Mobile Robots",From Perception to Action Conference (PERAC'94), Lausanne , 1994, pp.254-265.
2. Medeiros, Adelardo A. D. "A survey of control architectures for autonomous mobile robots." J. Braz. Comp. Soc., Apr. 1998, vol.4, no.3. ISSN 0104-6500.
3. Dominik Henrich, Thomas Höniger, "Parallel Processing Approaches in Robotic", ISIE (1997)
4. Claude Guéganno and Dominique Duhaut , "A Hardware/Software Architecture for the Control of Self-Reconfigurable Robots", DARS 2004, Toulouse (France).
5. http://www.univ-ubs.fr/valoria/duhaut/maam