Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2012, Article ID 193864, 15 pages
doi:10.1155/2012/193864

Research Article

A Mobile Computing Framework for Pervasive Adaptive Platforms

Olivier Brousse,[1,2,3] Jérémie Guillot,[1] Gilles Sassatelli,[1] Thierry Gil,[1] François Grize,[2] and Michel Robert[1]

[1] *LIRMM UMR 5506, Université Montpellier 2, CNRS, 161 Rue ADA, 34095 Montpellier Cedex 5, France*
[2] *Département des Systèmes d'Information, Faculté des Hautes Études Commerciales, Université de Lausanne, 1015 Lausanne, Switzerland*
[3] *LEAD-UMR 5022, Université de Bourgogne, CNRS, Pôle AAFE, Esplanade ERASME, BP 26513, 21065 Dijon Cedex, France*

Correspondence should be addressed to Olivier Brousse, olivier.brousse@u-bourgogne.fr

Received 15 June 2011; Accepted 16 September 2011

Academic Editor: Yuhang Yang

Ubiquitous computing is now the new computing trend, such systems that interact with their environment require self-adaptability. Bioinspiration is a natural candidate to provide the capability to handle complex and changing scenarios. This paper presents a programming framework dedicated to pervasive platforms programming. This bioinspired and agentoriented framework has been developed within the frame of the PERPLEXUS European project that is intended to provide support for bioinspiration-driven system adaptability. This framework enables the platform to adapt itself to application requirements at high-level while using hardware acceleration at node level. The resulting programming solution has been used to program three collaborative robotic applications in which robots learn tasks and evolve for achieving a better adaptation to their environment.

1. Introduction

Pervasive computing has been gaining attention due to the emergence of a number of ubiquitous applications where context awareness is of importance. Examples of such applications range from ad-hoc networks of mobile terminals such as mobile phones to sensor networks systems aimed at monitoring geographical or seismic activity. All these systems involve (i) monitoring and processing collectively environmental and platform information, (ii) adapting to time-changing scenarios.

Considering the similarity between living organisms and the adaptability needs of such platforms, drawing inspiration from biology appears a natural solution. Although a number of techniques such as genetic algorithms or artificial neural networks exist, pervasive computing opens a new dimension of opportunities for further extending bioinspiration.

There exist several theories that relate to life, its origins, and all its associated characteristics. It is, however, usually considered that life relies on three essential mechanisms that are phylogenesis, ontogenesis, and epigenesis (referred to as respectively P, O, and E throughout this paper):

(i) Phylogenesis is the origin and evolution of a set of species. Evolution gears species toward a better adaptation of individuals to their environment; genetic algorithms are inspired from this very principle of life.

(ii) Ontogenesis describes the origin and the development of an organism from the fertilized egg to its mature form. Biological processes like healing and fault tolerance are ontogenetic processes.

(iii) Epigenesis refers to features that are not related to the underlying DNA sequence of an organism. Learning as of performed by artificial neural networks (ANN) is a process which scope remains limited to an individual lifetime and, therefore, is epigenetic.

The Perplexus European project [1] (that last from september 2006 to march 2010) aimed at developing a platform of ubiquitous computing elements that communicate

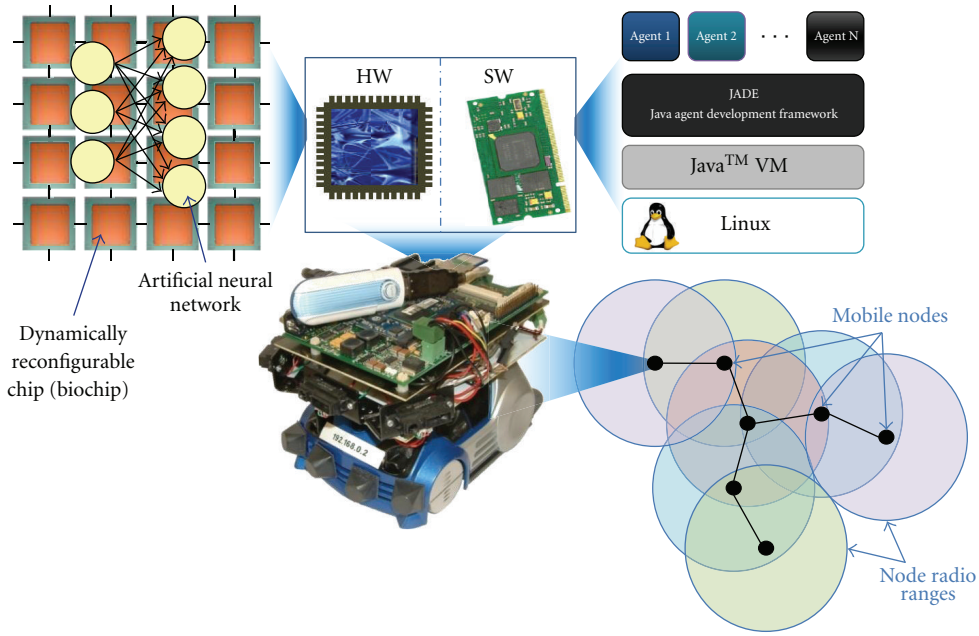

Figure 1: A pervasive sensor network example.

wirelessly and rely on these three principles of life. Intended objectives range from the simulation of complex phenomena such as culture dissemination [2] or biologically plausible neural networks [3] to the exploration of bioinspiration-driven system adaptation in ubiquitous platforms. As a consequence, this research dedicated platform has been developed keeping to explore the impact of bioinspired features in this context of omnipresent and present throughout computing. As a consequence, no particular efforts have been made to secure the platform as it is held in a research laboratory. In case a future evolution of the platform would aim to be disseminated on a larger scale, several solutions maybe envisaged to secure it. As we use standardize technologies, Wifi may be ciphered, additional network and communications ciphering may be used such as VPN, SSL authentication, and JADE-S services. Another consequence of this restricted deployment stands at the power consumption level. Once again no specific effort has been made to limit the power consumption of the modules. As a matter of fact at the exception of the collaborative robotic applications presented in this paper, all platform modules are used plugged to power grid.

Each ubiquitous computing module (called Ubidule) is made of a XScale [4] microprocessor that runs an embedded Linux operating system and a bioinspired reconfigurable device that essentially serves the purpose of running artificial neural networks (ANNs). This device is referred to as Ubichip [5] (Ubidule Chip) throughout this paper. The Ubichip supports two main operating modes:

(i) a native mode in which the chip behaves similarly to a FPGA [6], however, endowed with bioinspired features like automatic partial reconfiguration and self routing,

(ii) a single instruction on multiple data (SIMD) processor mode [7] that allows parallel computation of algorithms like neural network. This mode is presented in more details in the following.

Finally, ubidules are equipped with a wireless network adapter for internode communications, as well as sensors and actuators on an application-specific basis as illustrated in Figure 1.

This paper presents two contributions

(i) First, a generic agent-based infrastructure that provides native support for bioinspiration dedicated to pervasive distributed platforms is described. This bioinspired programming framework based on agent oriented programming allows synchronizing population-level mechanisms (evolution through distributed genetic algorithms) and node-level mechanisms (learning processes using the reconfigurable device).

(ii) Secondly, a means for transparently taking advantage of the Ubichip in SIMD mode is presented. This chip mode being dedicated to neural network simulations, it proves well suited to run learning mechanisms at the node level. The presented technique relies on a specific compiler that translates entire agent code sections into hardware executable binaries that speed up the execution.

The adaptability that results from these two contributions is demonstrated on three applications that use a fleet of autonomous vehicles; a lap race that uses Phylogenesis (evolution), an obstacle avoidance application that relies on collaborative learning as our first generation of applications and finally a robotic society evolution application that regroups phylogenetic and epigenetic aspects of the previous applications.

This paper is organized as follows:

(i) Section 2 describes the bioinspired agent programming framework used to the specification of pervasive and adaptive applications,

(ii) Section 3 details the techniques used for enabling the use of the reconfigurable bioinspired device which is at the heart of the Ubidule,

(iii) Section 4 presents the 3 applications in which Ubidules are embedded into small autonomous vehicles that learn and evolve,

(iv) Section 5 concludes on this work and draws some perspectives for future work.

2. Adaptive Mobile Computing Environment

The modular structure of the Perplexus platform offers scalability thanks to the decentralized network structure which avoids central bottlenecks. Modules are then connected to each other using point-to-point and infrastructureless connections. In the case of the Perplexus project applications, the network reactivity and reliability are important criterions. We estimate that in most Perplexus applications this latency should not exceed 10 seconds for communication reliability and performance as well as for buffer memory reasons. Networking in ad-hoc platforms constitutes a challenge because of the topology of the network that does not rely on a fixed structure with routers, DHCP, or DNS servers. This challenge becomes critical when nodes are mobile. Indeed, it induces the need of distributed adaptive features at the platform/network level.

2.1. Network Support. The emergence of smart mobile devices able to manage network-based applications and the associated ad-hoc network support shares the same challenge with the Perplexus platform. In the literature such a paradigm is known as MANET for Mobile Ad-Hoc NETwork [8]. This internet engineering task force (IETF) working group is in charge of proposing software solutions and standardizing IP routing protocols in the scope of wireless ad-hoc routing with either static or dynamic topologies.

MANET routing algorithms can be classified into two families.

(i) *Reactive MANET Protocols (RMPs)* that search for a route between nodes A and B when a communication is requested. AODV (for ad-hoc on-demand distance vector) [9] and DSR (for dynamic source routing) [10] are reactive protocols. Once a route has been found, communications are directly established until the topology of the network changes which results in the computation of a new route.

(ii) *Proactive MANET Protocols (PMPs)* in which nodes regularly exchange messages in order to maintain routes up to date and elect relay nodes. Optimized link state routing (OLSR) [11] is a proactive protocol complying with this principle. These protocols exhibit better performance and also prove more power consuming because of the constant route updating process.

Critical points for the routing scheme in the Perplexus applications are communication reliability and latency. Proactive protocols that offer a better latency/power consumption tradeoff when nodes are moving are well suited for our platforms.

The OLSR routing protocol is among the most popular and effective MANET solutions [12, 13]. This proactive routing protocol regularly sends 3 different types of messages to create and maintain automatically network routes (i.e., in a proactive way). This mechanism is well suited for most mobile applications as it provides reduced communication latency due to mostly up-to-date routes. This is all the more true in comparison to reactive protocols that are slower to establish a route before actually communicating meaningful data.

The chosen OLSR implementation offers the possibility to set the number of desired relay nodes or to use plugins to provide additional services such as name/address translation or link quality routing [14]. Additionally three references present studies about the OLSR power consumption and/or propose energy-efficient versions of this algorithm [15–17]. Using one of these versions in conjunction with the nameservice plugin of OLSRd, the communication power consumption may be reduced significantly. This has not been done but it will be investigated for the next generation of the Perplexus platform.

2.1.1. OLSR Validation on Perplexus Platform. For validating this solution, we conducted several experiments which confirmed that proactive protocols such as OLSR perform better with respect to latency. Figure 2 shows the experimental protocol we used for OLSR. The map of the premises shows four nodes, three being static and the last one (Ubidule 3) in motion along the path (illustrated by the plain arrow).

As suggested in Figures 2 and 3, different network topologies are observed as node3 moves along the path. The changes from one to another occur whenever a node drops out or comes in the radio range of another. Results presented in Figure 4 correspond to a representative experiment. In this experiment, we set the number of relays to 1 and disabled the link-quality routing to observe OLSR with the lightest solution in real conditions without any optimization.

Figure 4 shows the evolution of the communications bitrates received by the mobile node from the three other units; it can be clearly seen that a change in the network topology results in a break in one or more communication flows that lasts up to 5 seconds (dark arrows). In the case where packets are transiting through a relay node, they are not lost but temporarily stored in the relay and sent when communication is restored (light arrows). These results show that the OLSR protocol is compliant with the Perplexus platform. Results presented in Figure 4 also fulfill the latency constraint we estimated for Perplexus applications.

We consider these results satisfactory for the targeted applications; furthermore, the flexibility of the chosen OLSR implementation allows using a nameservice (DNS-like)

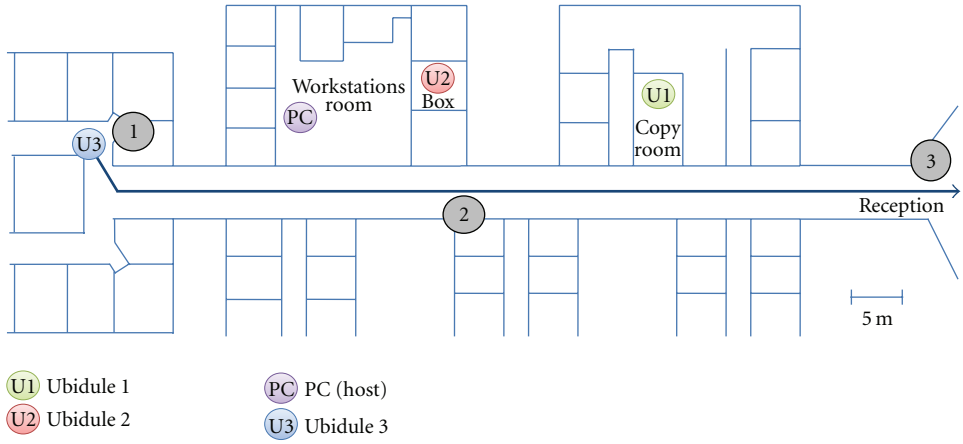

Figure 2: Mobile test protocol.

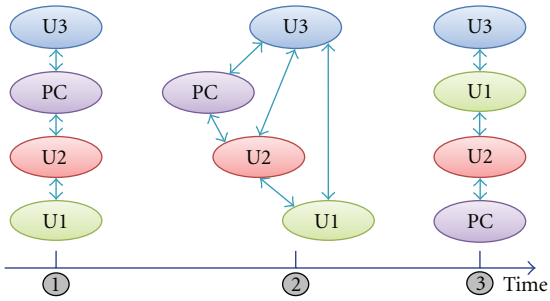

Figure 3: Time changing topology.

plug-in that proves mandatory in the following. The nameservice plug-in acts in two successive steps:

(1) the plug-in uses OLSR to broadcast messages containing IP and hostname information,

(2) It collects other nameservice message and stores received data in the hostsIP file (i.e., /etc./hosts for Linux OS).

Consequently, OLSR with nameservice allows each module to get a local routing table working with IP addresses and hostnames and make the ad-hoc network act as a standard structured network. OLSR and a slightly modified nameservice plug-in prove to be sufficient in term of network reactivity and efficient solution, in the case of our MANET application, as this solution perfectly fits our latency and performance needs and does not incur significant processing workload.

2.2. The FIPA Multiagent System. Programming distributed/pervasive applications are often regarded as a challenging task that requires a proper programming model capable of adequately capturing the specifications. Agent-oriented programming (AOP) derives from the initial theory of agent orientation which was first proposed by Shoham [18]. Agent-orientation was initially defined for promoting a social view of computing and finds natural applications in areas such as artificial intelligence or modeling of social behaviors. AOP consists in making agents interact with each other through typed messages of different natures: agents may be informing, requesting, offering, accepting, and rejecting requests, services, or any other type of information. AOP furthermore sets constraints on the parameters defining the state of the agent (beliefs, commitments, and choices).

These constraints essentially define the agent oriented computational system which is then viewed as a set of communicating software modules that exhibit a certain degree of independence making the whole system more adaptive than an object oriented (OOP) computational system. These characteristics naturally geared the Perplexus modeling framework toward AOP as a solution for adaptability in our pervasive architecture.

2.2.1. FIPA-Based Agents. The IEEE group named Foundation for Intelligent and Physical Agents (IEEE-FIPA) defines standards allowing for interoperability among various multiagent platforms. Figure 5 shows the FIPA standard structure of an agent platform (AP). Three main services ensure FIPA platforms reliability and functionality.

The agent management system (AMS) is in charge of the life cycle of platform agents; it can create, suspend, resume, or kill agents. The AMS also provides a white page service listing all agents "living" on the platform.

The directory facilitator (DF) is in charge of providing a yellow pages service. This service associates an agent to its offered services and a service to agents that provide it.

The message transport system (MTS) provides all communication functionalities at low-level. Therefore, agents can communicate with each other regardless their location (same or different APs).

Figure 5 shows that FIPA mandatory agents (i.e., AMS and DF) reside at the agent level next to user agents; they therefore behave as such and provide the above-mentioned functionalities. On the contrary, the message transport system lies at a lower level dedicated to communication protocols that provide a framework for interagent message communications. The FIPA standard does not include an AP search service that allows to discover FIPA peer APs as of

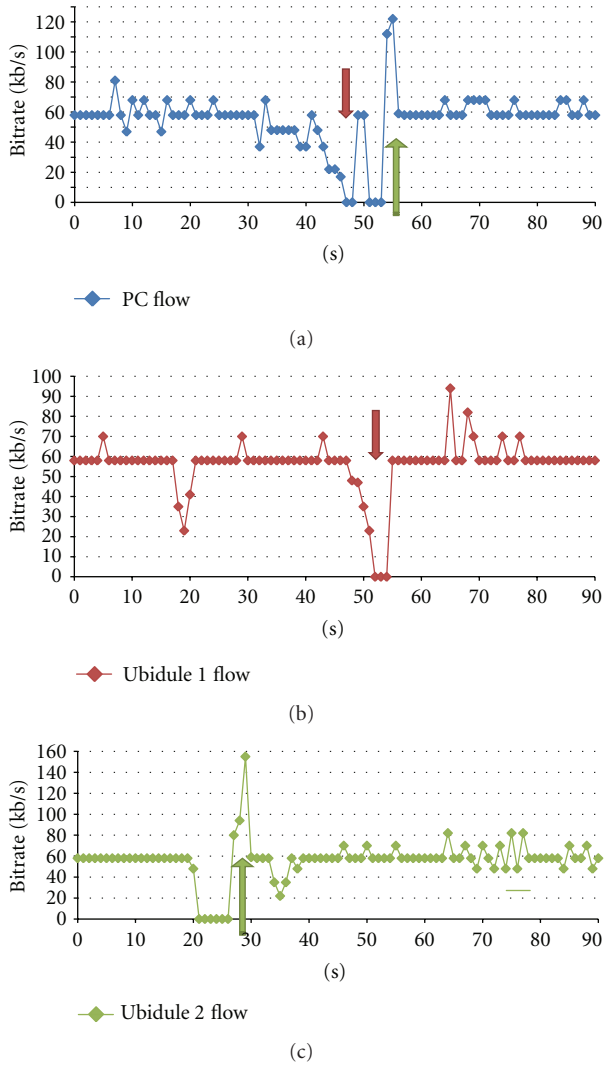

Figure 4: Mobile test: mobile node received message flows.

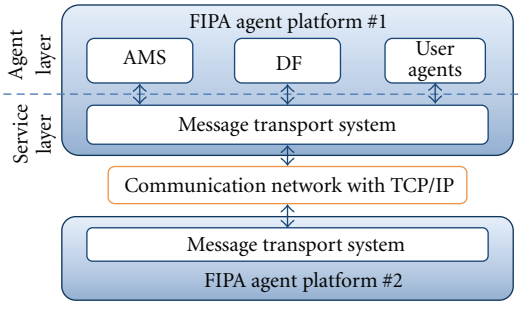

Figure 5: FIPA standard overview.

today. This feature exists in several protocols such as JXTA [19] or Kademlia [20] peer-to-peer protocols. This drawback of the standard does not ease the use of multiple platforms which is targeted in this work. In this paper, we propose a solution to tackle this problem, detailed in Section 2.3 and finally allow an FIPA compliant AP to search for peer platforms.

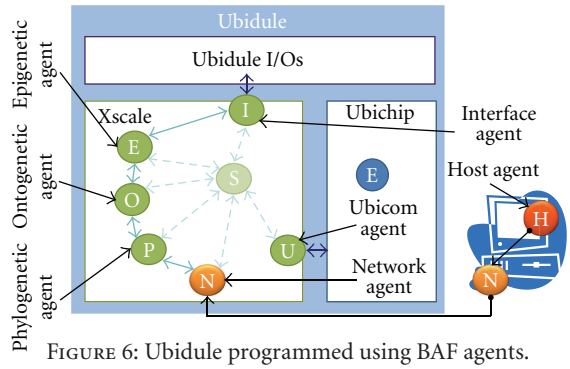

Figure 6: Ubidule programmed using BAF agents.

2.2.2. JADE: Java Agent Development Framework. There exists various multiagent platforms that allow developing agent-based applications, such as JADE [21], JXTA [19], FIPA OS [22], JAX [23], or MADKIT [24].

As the Ubidule XScale is a resource-limited embedded processor [25], many of the listed solution cannot be efficiently implemented. JADE was chosen for its portability (Java), FIPA compliance, and also because of the availability of a lightweight version called lightweight extended agent platform (LEAP) already used on embedded devices [26].

Agents in a JADE Framework "live" in containers. These containers exist either inside or outside of the original hardware hosting the JADE platform but are registered in the AMS and DF platform agents in an "original hosted main container."

Communications take place using a message transport protocol (MTP) which in turn uses TCP/IP protocols. In our case, we decided to use HTTP MTP in order to ease AP communications and unify AP name and address with hardware hostname. This point is discussed with more details in the following section.

2.3. Contribution 1: Bio-Mimetic Agent Framework. Previously described solutions at network level (OLSR) and programming level (LEAP) are used as basis for a bio-inspired agent framework (BAF) suitable for distributed, decentralized, and mobile platforms where adaptability is mandatory. This section focuses on two fundamental aspects of the proposed BAF: on one hand the description of the BAF and overview of the provided functionality, on the other the description of the POE specific agents.

Bioinspiration and the three fundamentals of life being at the heart of the project, the proposed framework extends JADE default platform, that is, mandatory agents (AMS and DF) by defining agents whose purpose relate to both interfacing and bio-inspired mechanisms support as well as pervasive computing platform management agents. Figure 6 schematically depicts the ubidule programming which is regarded as a mixed hardware/software entity: the Ubichip for hardware support and the XScale microprocessor for software side.

The BAF specifies 7 agents belonging to 2 families:

(i) application agents: phylogenetic, ontogenetic, and epigenetic agent(s),

(ii) infrastructure agents: UbiCom, interface, network, and spy agent(s).

All these 7 agents have been developed using LEAP classes to support a dedicated function. Therefore, they add the BAF mandatory features to the legacy JADE. Figure 6 shows both the infrastructure and application agents and their interactions (for the sake of clarity AMS and DF agents are not represented).

(i) P agent: the Phylogenetic agent is responsible of the execution of the distributed genetic algorithms: it calculates the local fitness of the individual (the local ubidule) and synchronizes this information with all other ubidules. It is responsible for triggering the death (end of a generation) and birth of the embodied individual hosted on the Ubidule.

(ii) O agent: the ontogenetic agent is tightly coupled to the P agent: it takes orders from him and has the capability of creating other software agents.

(iii) E agent: the Epigenetic agent embodies the individual and its behavior: it is either a software or hardware neural network.

Next to the three POE agents, four additional agents have been defined for interfacing and networking purposes.

(i) I agent: the interface agent provides a set of methods for issuing commands to the actuators or retrieving data from the sensors of the ubidule.

(ii) U agent: the UbiCom agent provides software API-like access to the Ubichip and manages hardware communications with the chip.

(iii) S agent: the spy agent provides information on the platform state (agent status/results, activity traces, bug notification).

(iv) N agent: the network agent provides a collection of methods for network-related aspects: time-synchronizing of data among ubidules, setting/getting clusters of ubidules, obtaining the list of neighbors, and so forth. For it requires access to low-level network-topology information, it also implements the MANET functionalities.

Finally, a host agent (H agent) instantiated on a workstation allows controlling remotely the Perplexus platform (Start/Stop/Schedule actions) through a graphical user interface.

Figure 7 shows the modifications applied to the FIPA platform for integrating the platform agents listed above. For the sake of clarity, only the P, O, E, and N agents are represented; all other agents reside on the agent layer with AMS and DF agents.

The additional features of the BAF (shaded areas) comprise the network agent and a low level service layer that handles the ad-hoc networking features. This layer includes OLSR and the nameservice plug-in. The hostname/IP table (periodically updated by the nameservice) can easily be accessed by other software entities such as JADE agents. Any agent can access these services through a specific Network agent.

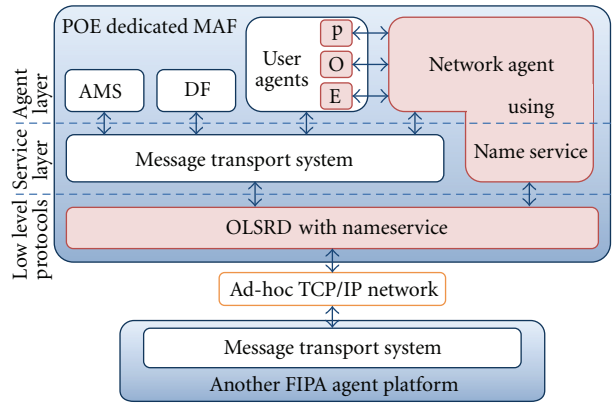

Figure 7: POE-dedicated BAF overview: white areas represent the classical JADE framework. BAF additional features to this framework appear in shaded areas.

The network agent is mandatory in a BAF platform. It allows FIPA platforms to communicate with each other and ensures the overall platform reliability. As this particular agent provides AP level services and low-level functionality (such as message broadcasting), it spans both highest level layers of the diagram. The use of the HTTP message transport protocol allows resolving the AP name and address in an ad-hoc network environment. Figure 8 describes this peer discovery mechanism.

Once the nameservice has edited the operating system Hostname/IP file (step 1), the network agent is able to create the peer platform list (step 2). Similarly, other agent lists can be created.

The host agent has been designed to provide a single interface for the platform management. This agent is able to remotely schedule applications from a host station thanks to the network agent services. A broadcast protocol is used for issuing global commands to the platform such as global *Service Search*, *Start Application*, *Stop Application*, or *Switch Mode* (switching from software mode to hardware accelerated mode, detailed later in Section 3.7). In this case, network agents sink command messages.

The main advantage of this method is that the host agent, and the user it represents, does not need to know addresses of all final receivers at design time allowing users to take advantage of the flexibility and scalability of the environment.

3. Hardware Acceleration

Bio-inspired features are heavy computational tasks that hardly fit with embedded devices such as the XScale processor used within the PERPLEXUS platform. The Ubichip has been designed to provide hardware support for such features. Figure 9 puts focus on the SIMD operating mode of the Ubichip used to accelerate parallel parts of PERPLEXUS applications.

In this specific mode, the ubicells are grouped by four to obtain an array of 16 bits processing elements (PEs)

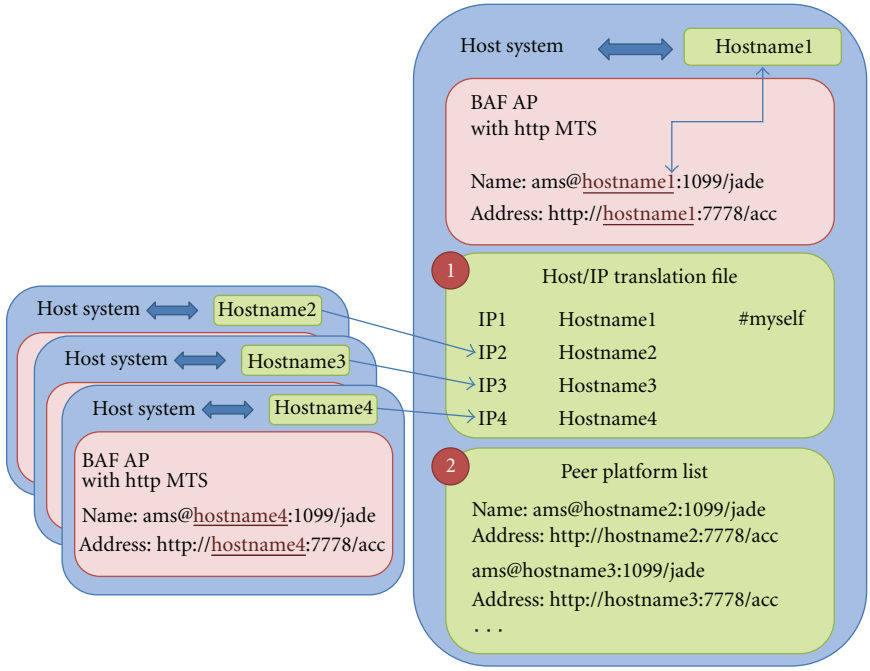

Figure 8: BAF AP address resolution.

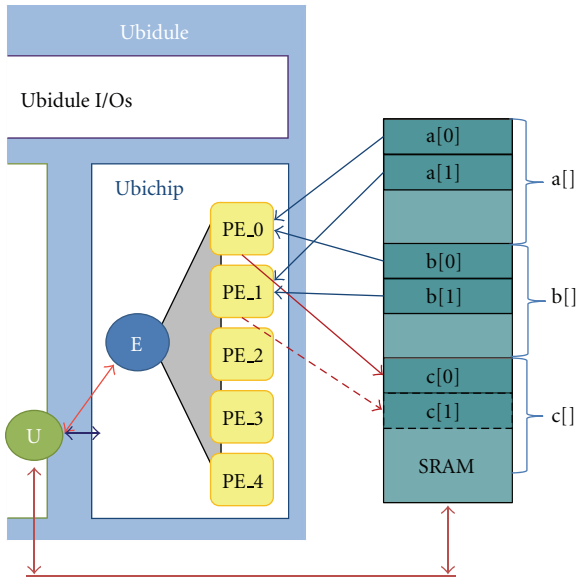

Figure 9: Ubichip SIMD mode architecture.

controlled by a sequencer. Program and data are stored in an external memory accessible by the sequencer and the XScale. The XScale is in charge of enabling or disabling the Ubichip allowing it to access the memory in a secured way. When the program ends, the Ubichip is able to interrupt the XScale allowing a proper result retrieval.

In this section, we present a solution to program this accelerator with the same programming language for both purely software and hardware accelerated agents.

3.1. Related Works. As agents in the BAF are captured in Java, we investigated the literature to find parallel oriented Java implementations. However, a classical Java virtual Machine (JVM) is by construction, executed on a single processor. Some Java hardware machines have been under study over the last decade, [27, 28] or [29], but none of them provide support for hardware parallelism as the original language was not intended to this.

Another approach is proposed by Manta [30] and relies on compiling Java threads to native x86 assembly code and run them on an x86 cluster through remote method Invocation (RMI). This solution removes the Java portability and does not target platforms such as the SIMD processor of the Ubichip.

Some software parallel classes like JPCL [31] add software parallelism to Java but JVMs are running on a single processor. Therefore, using this kind of libraries requires a framework that links several JVM running on several hardware targets and sharing the same global object space. The previously presented BAF ensures a similar software parallelism level but based on message passing scheme rather than shared memory.

The proposal is to provide a solution for easing the accelerator programming and consequently use a real- and fine-grain hardware acceleration in the PERPLEXUS framework.

3.2. Contribution 2: The Jubi Extensions. The fundamental concept behind the proposed approach relies on the use of directives for flagging parallel sections in a hardware-independent description based on Java: Java for ubiquitous or Jubi in short. Agent coded in Jubi can then be executed in

SW mode (in such case directives are ignored) or in hybrid SW/HW where flagged sections are compiled for parallel SIMD execution.

A flagged section of code presented as a component can be described with its inputs, outputs, and internal behavior. Adding this approach to Java requires setting firstly `in` and `out` keywords. An `NPE` keyword allows the user to specify the number of processing elements (PEs) that will be used for this application.

The following code where $c = f(a, b)$ gives an example of the applied transformations:

```
final int NPE = 4;
int a[NPE], b[NPE];
int c[NPE]
```

becomes

```
final int NPE = 4;
in int a[NPE], b[NPE];
out int c[NPE]
```

Then, to describe the behavior of the hardware block, we define the `#jubi` keyword that flags the code to be accelerated using the SIMD hardware. Finally, to enable the parallelization of software sequential loops in the hardware accelerated mode while keeping the sequential software execution possible, we introduce the `parallelfor` keyword. This keyword allows both software and hardware generating implementations from the same unified description.

The following code performs an addition on input vectors and illustrates the memory layout presented in Figure 9:

```
#jubi
parallelfor(int i=0; i<NPE; i++)
{
    c [i] = a[i] + b[i];
}.
```

3.3. Specific Tools. Figure 10 presents the compilation flow we propose in order to allow fast applications development in software or in hybrid HW/SW modes. In this figure, the software compilation flow appears on the left side whereas the hardware flow is on the right side.

Software applications execution only use software flow whereas hardware-accelerated applications require both sides to compile accelerated agents software part (named envelope) and hardware parts (named kernels). The agent envelope is a part of the Java file that triggers the hardware execution and feeds hardware kernels with appropriate data. This is done through the UbiCom agent that acts as a wrapper between sequential and parallel sections of the application code (i.e., software and hardware parts).

As presented in Figure 10, the processing of the Jubi file results in the creation of two distinct file types. One Java file that offers the possibility to start the application either in software mode or in hardware accelerated mode. For each `#jubi` block described in the Jubi file, one "Ubi" file is created. Every Ubi file encapsulates the code to be accelerated. Figure 11 details the splitting process that produces both the Ubi files and Java files with the required hardware calls through the UbiCom agent.

The Java file is compiled thanks to the standard Java compiler (javac), whereas Ubi files are compiled into associated hardware kernels, ".hw" files by the JubiCompiler and UbiAssembler. The last step of the compilation flow is the loading of the HW-accelerated code from a ".hw" file into the program memory of the Ubichip with up-to-date data. This is done at runtime by the UbiCom agent behavior.

3.4. JubiSplitter. The entry point of the JubiTool compiling environment is the JubiSplitter that splits the Jubi description into a Java description for the software part and several Ubi descriptions for the hardware-accelerated parts. The JubiSplitter tool generates "softwareBehaviour" and "hardwareBehaviour" classes. These JADE Behavior classes represent respectively the entire agent functionality in SW mode and the envelope of hardware accelerated kernels, which contains non-parallelizable (non `#jubi` flagged) code sections in the HW mode. The execution mode is then chosen when the platform is configured. Figure 11 gives an example of this code splitting stage which is the first stage of the application.

3.5. JubiCompiler. Once a Jubi code has been split into Java and Ubi kernels the JubiCompiler tool compiles every Ubi code into Ubichip assembly. As a result, the following example, where uninitialized values are set to 0:

```
int a [NPE] = {4,3,1,2};
int b [NPE] = {1,2,3,4};
int c [NPE];
c = a + b
```

is then compiled into

```
.data
    V01 = ''00030004'', ''00020001''
    V02 = ''00020001'', ''00040003''
    V03 = ''00000000'', ''00000000''
.code
    load   r1,V01
    load   r2,V02
    mova    r1
    add    r2
    movr    r3
    store  r3,V03.
```

Where the `r1`, `r2`, and `r3` represent the register of the Ubichip, `V01` corresponds to a, `V02` corresponds to b and `V03` corresponds to c.

The JubiCompiler is based on a flex and bison [32] description of the Jubi grammar. An array dimension set to `NPE` indicates that every processing element of the SIMD

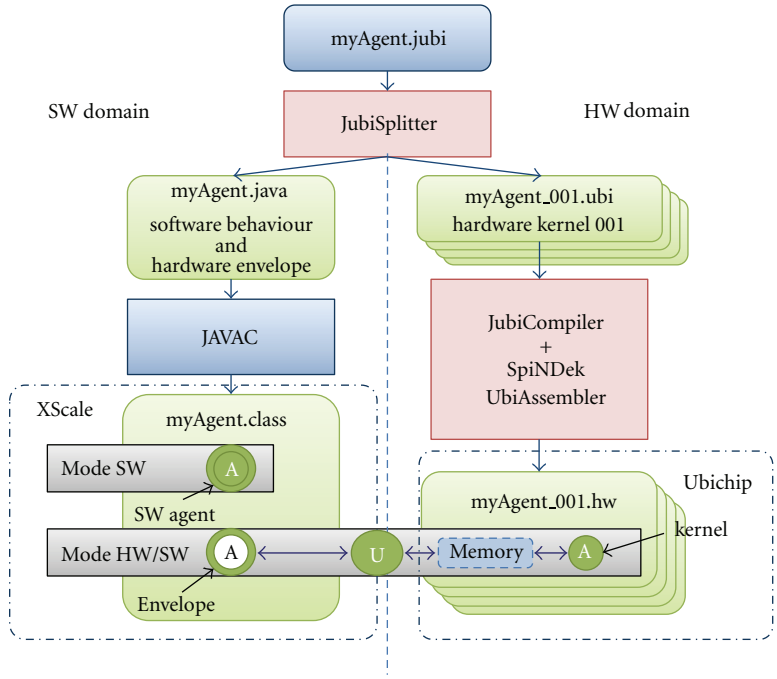

Figure 10: Unified compilation flow.

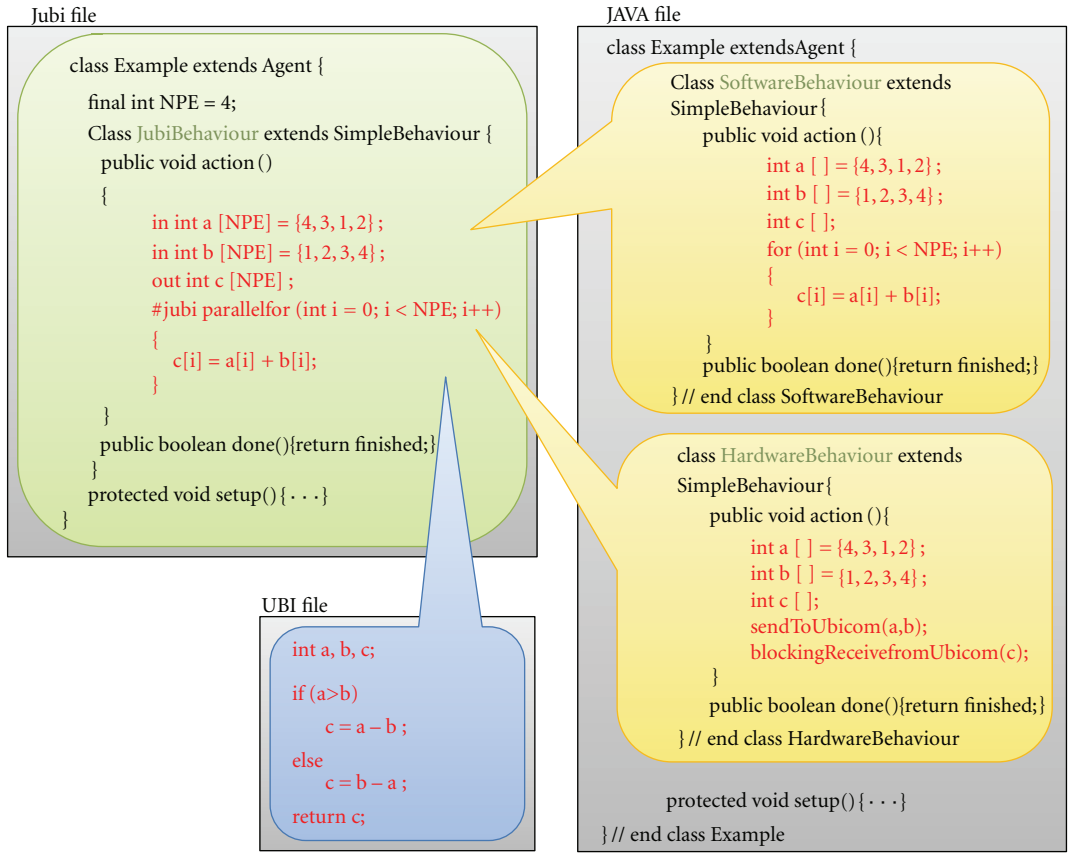

Figure 11: JubiSplitter: Code splitting and file creation.

processor will receive a different dataset. 1D arrays therefore become PE-local scalar variables. Hence, a 1D NPE-relative variable (in SW) is spanned over `NPE` PEs of the Ubichip architecture as `NPE` scalar variables (in HW).

3.6. UbiAssembler. The UbiAssembler is the final tool used in the flow. It is in charge of translating the assembly code generated by the JubiCompiler into Ubichip SIMD hardware binary executables. This tool is part of the SpiNDeK environment [33] and was developed to program the Ubichip using assembly. Two main features are provided in this tool in addition to the code translation: the memory layout setup that involves variable to physical address translation, and regular linking provided by label to physical address translation.

3.7. Toward an Adaptive Acceleration. The opportunity of running agents in software or hybrid hardware/software mode opens interesting perspectives in term of adaptability. Beyond enabling to assess the speedup resulting from hardware execution, this allows for online mapping of agents Ubi sections to the Ubichip. This may prove useful for adapting to changing performance requirements by migrating agents from hardware to software and the other way around.

The UbiCom agent has been designed for this purpose; it embeds Ubichip management functions and an interface that allows an agent to request a migration of a functionality (`#jubi flagged sections`) to the Ubichip. The U agent is able to start and stop the Ubichip and to load a binary file in the chip code memory. Then, after the loading phase the UbiCom waits until an interrupt is raised by the Ubichip to get data back and communicate them to the agent envelope.

The features presented in this section have been validated in VHDL simulations of the Ubichip model as the prototype was not available at writing time. Test programs that validate PERPLEXUS applications needed features have been compiled and fed into the Ubichip model as memory content files. The result assertion of these test programs proves the functionality of the proposed framework.

4. BAF Applications Validations

4.1. Introduction. Three case studies are presented in this section for, respectively, illustrating evolution and learning features of the proposed framework. These applications are not taking advantage of the hardware acceleration due to delays in the fabrication of the Ubichip accelerator.

Figure 12 schematically depicts the used robots, their sensors and actuators, as well as the framework agents presented previously.

The E agent is the main robot controller that reads data from the sensors and, depending on given or learned rules, issues commands to the wheel motors. The P agent is responsible of the robot controller evolution and therefore computes the robot fitness and runs the genetic algorithms together with the P agents of other robots. Ontogenetic agent instantiates the E agent based on the genome provided by the P agent. The N and I agents serve the purpose explained previously. The U agent is unused here as the chip is still under fabrication; therefore, presented applications only make use of software mode.

The use of either all these agents or only a subset of them is a design decision.

4.2. Test Application 1. In order to prove the reliability of the platform, a simple proof-of-concept application based on a race has been developed. This application relies on all framework agents; the robot controller (E agent) being here a simple feed-forward artificial neural network (ANN) that reads binary information from three proximity sensors installed on the front, front-left, and front-right sides of the robots, and issues speed commands to the two motors. For this application, robots are moving into a closed arena containing obstacles and a start/finish line. The goal of robots is to run one lap.

Figure 13 shows the principle of this genetic race the lap time gives the fitness of a given robot controller and hence its genome wich is the array of ANN weights. Therefore, there is no learning in this application, changes in the robot behavior being driven by evolution only.

These agents are crossed and/or mutated by the P agent to create the next generation replacing inadequate behaviors. Once a new individual genome is ready, the P agent forwards it to the O agent that instantiates the corresponding E agent. Generation after generation, robots exhibit better behaviors proving the reliability of the software and the possibility to handle POE problems via the platform.

A demonstration video that illustrates this application is available online at: http://www.lirmm.fr/ADAC/?page_id=9.

4.3. Test Application 2. Robots which participate use online learning (Epigenesis) for improving their performance. Figure 14 shows the robots that are enclosed in an arena scattered with obstacles (cylinders in Figure 14); collision avoidance is here the main objective. As this application only targets learning, the P and O agents are not used here.

The collaborative learning approach has been implemented in such a way that two networks are trained online in parallel (i.e., two individuals are running). Every-time a robot gets into an obstacle, its own network is trained to avoid this error in the future. The faulty robot also advices the other individual that the action it has just made, in the context it was, leads to an error. The other robot is then learning its proper network including this information. This scheme is repeated until robots are able to avoid obstacles exchanging their experience in live as represented in Figure 14.

Besides the previously described sensors, a bumper switch is added to inform the robot whenever a collision with an object occurs; it is located on the front side of the robot. The sensors here do not provide binary information but rather the distance with the nearest obstacle.

These robots move by issuing speed commands to each of the two motors. As depicted on Figure 14, an ANN is in charge of controlling the robot. Figure 15 depicts the principle of this application: the E agent is a multilayer perceptron ANN that uses a standard back-propagation learning algorithm.

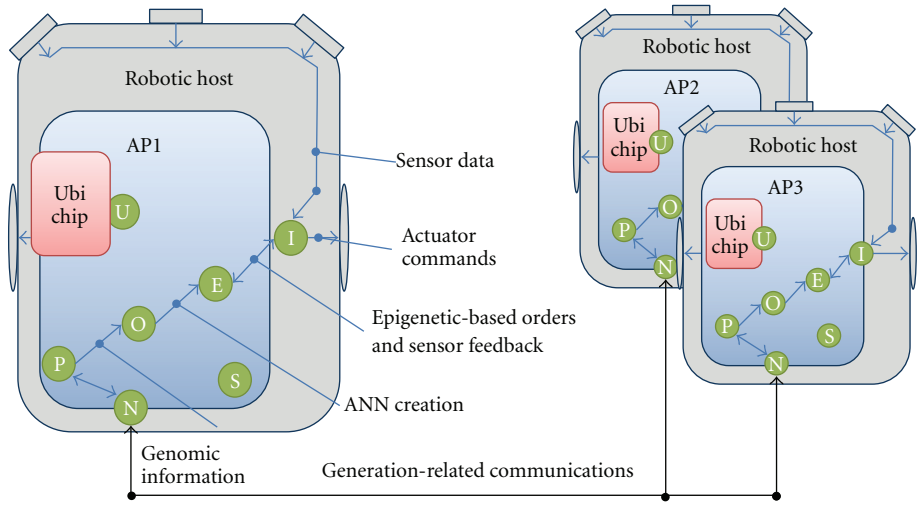

Figure 12: Application environment.

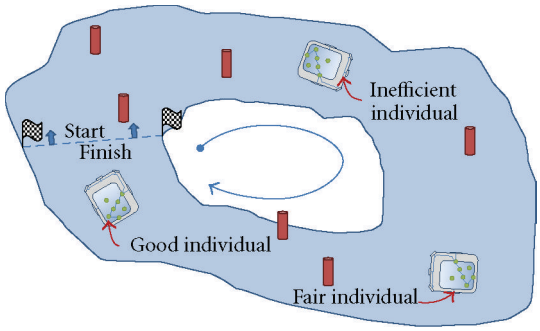

Figure 13: Evolution-driven application overview.

Table 1: Ultrasonic sensors areas.

Area	Distance range
0	0–200 mm
1	200–400 mm
2	400–600 mm
3	600–800 mm
4	>800 mm

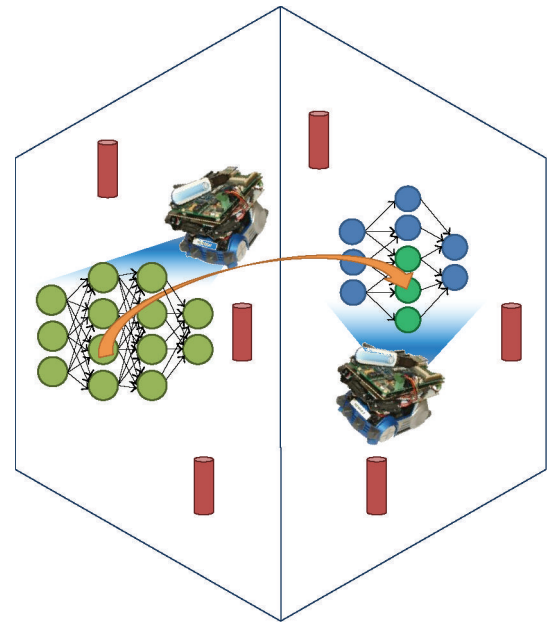

Figure 14: Collaborative learning application overview.

Inputs of the ANN are the three values measured by the infrared and the ultrasonic sensors, we have defined five areas for each ultrasonic sensor as depicted in Table 1.

The outputs of the ANN are speed values sent to the motors, each value is set as an integer value from −7 (i.e., fast backward motion) to +7 (i.e., fast forward motion). The robot can turn by applying two different speeds on the motors.

During a given period, each robot performs the following tasks.

Robots are moving in an unknown environment. Each time they collide into an obstacle, a random modification of the relevant learning pattern is applied and an ANN learning phase is triggered online. The robot then notifies all its peers that this pattern shall be modified; and the modification is registered by all robots therefore collectively speeding up the convergence toward a satisfying solution.

In this application, a host system that runs the H agent (host agent) is used for launching the application and to collect information throughout the execution of the algorithm.

As depicted in Figure 16, the host workstation and all the ubidules are running the BAF environment.

Our experiments show that this technique exhibits a speedup (versus a single robot) that is almost linear with the number of used robots. Furthermore, it has been observed that a convergence threshold is reached after a number of iterations which is a function of the complexity of the environment. Once this threshold is reached, adding some

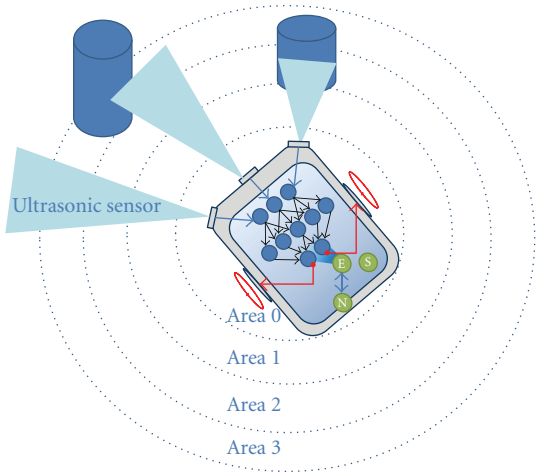

Figure 15: Collaborative learning application overview.

more obstacles in the arena retriggers learning until a new threshold is reached, demonstrating the adaptability potential of the proposed solution.

A video showing the runs of the above-presented individuals is available online at: ifundefinedselectfont http://www.lirmm.fr/ADAC/?page_id=9.

4.4. Evolving a Population of Learning Individuals. Based on the same idea to prove that bioinspired features are useful for distributed and pervasive systems adaptability, the third application relies on the association of P and E features. We designed it as a mixed evolution and learning robotic demonstrator. In this application, the robot behavior is set using a dynamically changing quality function created following the individual genome. This function is based on couples state/action and is modified using reinforcement learning [34]. As a consequence, robots are following move rules that depend on the present sensors state and actions are rewarded if the action is a success or punished if a collision occurred. The application runtime is described in Figure 17 and can be explained with the following steps.

(i) Robots move in the obstacle-scattered arena of Figure 14 learning form their errors during 2 minutes. When they collide with an obstacle, learning is triggered inducing a change in the quality function. To avoid wall stucking, we also make the robot to move backward on collision. Due to the limited time and various uncertainties induced by their genome, some individuals are not able to learn properly to avoid obstacle whereas good individuals are learning rapidly.

(ii) The individual fitness is then calculated using the number of collisions balanced with the global-recorded speed of the robot during the 2 minutes run.

(iii) The new generation is then created merging and mutating the genomes of individuals with the best fitness.

(iv) The simulation ends when an individual achieves to spend an entire run without colliding with any obstacles.

In this application, robots inherit their characteristics such as right/left and front sensors zones from their respective parents. Robots are also improving themselves using a collision-triggered process that allows online learning. In Figure 17 blue part represents step of the application where individuals interact with each others using the BAF communications features.

One additional aspect we introduced in this application is the PE cooperation effect called the parental education. We define the Parental Education as a merging of the following two aspects: innate for newborn inherited behavior characteristics and acquired for the transmitted knowledge.

(i) The first is the innate aspect that can be encountered in some species. This process similar to instinct allows newborn individual to walk within minutes, this is only possible because their parents and the whole species acquired this innate ability.

(ii) The second is the parent influence on children representing the childhood learning in the nature, parents of evolved species like humans are teaching their children.

To mimic these PE aspects, we chose to transmit a given percentage (between 20 and 50%) of the parent quality function to the child genome.

One of the characteristics of the reinforcement algorithm is the reflex latency value. It corresponds to the delay between a given move and its associated reward or punishment. This characteristic can easily be used as a genome parameter and then evolve with the species.

The individual genome is represented in Figure 18 with the three main transmitted characteristics namely ultrasonic sensor zone definition, parental education patterns and reflex latency value.

Using parental knowledge rapidly brings robots to an average species-level behavior that online learning further improves. Subsequent offspring will, therefore, benefit from species capabilities evolution through inheritance.

In this application, the quality function is stored in an array where every value of quality is associated with the corresponding state/action couple. Following quality function examples, are extracted from application results with three possible move forward (MF), turn Left (TL), and turn Right (TR) actions per sensor state. Three sensing zones are used leading to $3^3 = 27$ possible sensors states, for the sake of simplicity only some of the short-length and wide-range zones quality functions are exposed in Table 2. Presented values are obtained after the first generation run.

The quality function used in this application is based on the action score. The action with the highest quality is used depending on the current sensor state. These actions are represented for the two first sensor states with shaded cells. If this action provokes an error (i.e., the robot get into an obstacle), the action score is lowered. On the contrary, if the action is successful its score is raised. Table 2 shows two final

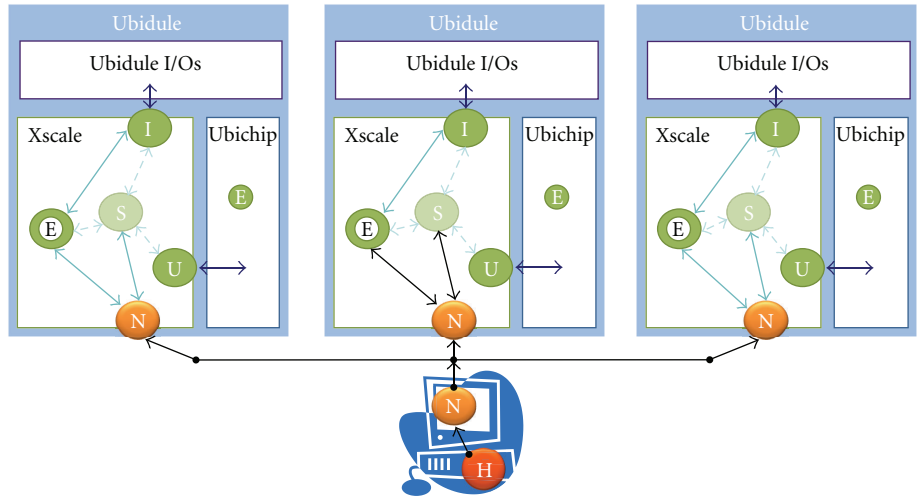

Figure 16: Application involved agents.

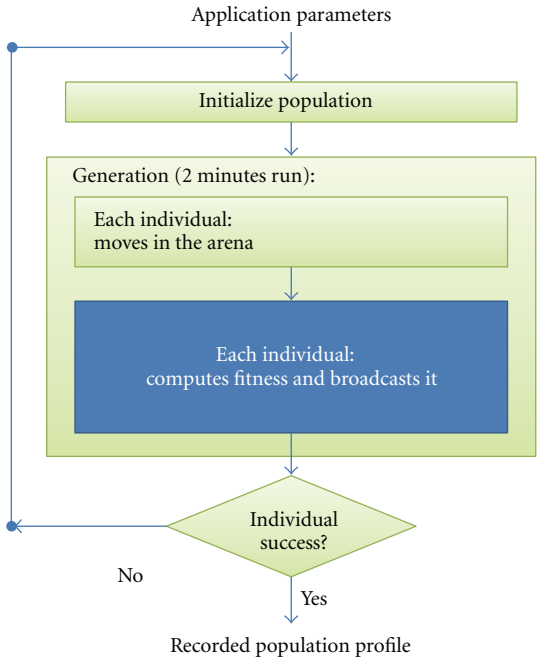

Figure 17: Second-generation application steps.

Table 2: Quality function example.

L/F/R sensor zones	Action	gen 1 QF	gen 4 QF
	MF	−4.0	−4.5
0 0 0	TL	−5.625	−4.5
	TR	−5.0	−2.0
	MF	−1.0	−2.0
0 0 1	TL	−0.0	0.0
	TR	0.0	0.0
	MF	−4.0625	−4.0625
0 0 2	TL	−4.0625	−4.0625
	TR	0.25	1.9921875
⋮	⋮	⋮	⋮
	MF	9	9
2 2 1	TL	0.0	0.0
	TR	0.0	0.0
	MF	1.9960938	2.0
2 2 2	TL	0.0	0.0
	TR	0.0	0.0

quality functions for different generation individuals. This demonstrates the generation after generation evolution that occurs even if 20% to 50% of the individual self-experience is transferred to children.

The fitness of an individual is defined in (1) where α is the reflex coefficient normalizer, S_{mean} the mean value of motion speed, and B_i the collision penalty of the ith move:

$$\text{Fitness} = \alpha \times \left(\sum_{i=1}^{n} S_{\text{mean}} \times (1 - B_i) \right). \quad (1)$$

With this fitness computation rule, we promote forward, moving individuals that avoid collision taking into account the number of moves during the 2 minutes run as well as their speed.

Table 2 shows resulting quality functions of individuals whose genomes main difference is their respective reflex latency value. The combining of this reflex value with the closest sensor zone reveals to be critical in various cases and had a great influence on the genome fitness. In the above-cited examples the reflex latencies were respectively, 185 ms and 195 ms resulting in respective fitness of 744 pts and 674 pts. One generation later, the recorded fitness of two of their children are 776 pts and 744 pts using the same reflex latency differentiator.

This demonstration shows the efficiency of a PE-based application in the robotic field, and the faculty of the proposed framework to run advanced bioinspired applications. Our experiments show that even if every generation is

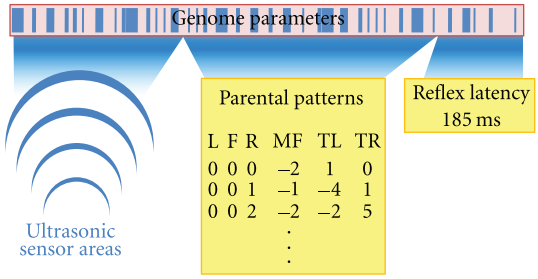

Figure 18: Robot genome example.

running only 2 minutes, the ability to provide information on the desired behavior to the next generation (PE effect) brings an interesting improvement compared with the classical P based application presented in Section 4.2.

A video showing the runs of the above presented individuals is available online at: http://www.lirmm.fr/ADAC/?page_id=9.

5. Conclusion

This paper presents a bioinspired agent-oriented framework dedicated to the prototyping of adaptive pervasive applications. Furthermore, this solution provides a means for taking advantage of hardware acceleration thanks to the use of language extensions associated with a specific compiler that generates code for the chip developed within the confines of the Perplexus European project.

The proposed proof-of-concept applications suggest that bioinspiration brings advantages for achieving adaptability in pervasive applications. To this end, dedicated robots with improved sensory capabilities are currently under fabrication. These robots will furthermore have the capability of hot swapping their depleted batteries autonomously thanks to a dedicated docking station, therefore, enabling to setup experiments lasting days or weeks.

Within the frame of the project, ongoing work focuses on demonstrating the combined advantages of the developed framework along with the bio-inspired device on a fleet composed of several tenths of robots running over long periods.

Although the adaptability features have been demonstrated on robotic applications, we believe that other application areas may benefit from the proposed solution. May it be for scheduling of communications for optimizing power in a sensor network, or devising techniques for transmitting data collected by distributed nodes to a gateway, the dependence to the environment makes such adaptive solutions attractive for coping with non deterministic scenarios.

Acknowledgments

This project is funded by the Future and Emerging Technologies Program IST-STREP of the European Community, under Grant IST-034632 (PERPLEXUS). The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

References

[1] Perplexus, Pervasive computing platform for modeling complex virtually-unbounded systems, 2009, http://www.perplexus.org/.

[2] J. Peňa, O. Jorand, H. Volken, and A. Pèrez-Uribe, "A connectionist, embodied and situated agent-based approach for studying the dissemination of culture," CESABM, UNIL.

[3] O. Chibirova, J. Iglesias, V. Shaposhnyk, and A. E. P. Villa, "Dynamics of firing patterns in evolvable hierarchically organized neural networks," *Lecture Notes in Computer Science* , vol. 5216, pp. 296–307, 2008.

[4] Intel corp., "Intel xscale microarchitecture," Tech. Rep., 2000.

[5] Y. Thoma and A. Upegui, "Specification of bio-inspired features to be supported by the device," hEIG-VD, Yverdon, Switzerland, Internal Report, 2006.

[6] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Uribe, J. M. Moreno, and J. Madrenas, "The perplexus bio-inspired chip," in *Proceedings of the 2nd NASA/ESA Conference on Adaptive Hardware and Systems(AHS '07)*, IEEE Computer Society, 2007.

[7] J. M. Moreno, "Specification of the ubicell," Tech. Rep., Barcelona, Spain, UPC, Internal Report, 2006.

[8] IETFMANET work group, "Mobile Ad-Hoc networks (MANET)," April 2009, http://www.ietf.org/html.charters/manet-charter.html.

[9] C. E. Perkins and E. M. Royer, "Ad-Hoc on-demande distance vector," December 1998.

[10] D. Johnson and D. Maltz, "The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4," February 2007.

[11] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, *Optimized Link State Routing Protocol for Ad-Hoc Networks*, INRIA Roquencourt, HiPERCOM project, 2001.

[12] T. Clausen, P. Jacquet, and L. Viennot, *Comparative study of CBR and TCP performance of MANET routing protocols*, Workshop MESAINRIA Roquencourt, HiPERCOM project.

[13] A. Huhtonen, "Comparing AODV and OLSR routing protocols," Seminar on Internetworking.

[14] A. TØnnesen, *Impementing and extending the optimized link state routing protocol, Tech. Rep.*, M.S. thesis, UniK University Graduate Center University of Oslo, 2004.

[15] F. de Rango, M. Fotino, and S. Marano, "Ee-olsr: energy efficient olsr routing protocol for mobile ad-hoc networks," in *Proceedings of the Military Communications Conference (MILCOM '08)*, San Diego, Calif, USA, November 2008.

[16] F. D. Rango, J. C. Cano, M. Fotino, C. Calafate, P. Manzoni, and S. Marano, "OLSR vs DSR: a comparative analysis of proactive and reactive mechanisms from an energetic point of view in wireless ad hoc networks," *Computer Communications*, vol. 31, no. 16, pp. 3843–3854, 2008.

[17] C. Taddia, A. Giovanardi, and G. Mazzini, "Energy efficiency in OLSR protocol," in *Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks (SECON '06)*, pp. 792–796, Reston, Va, USA, September 2006.

[18] Y. Shoham, "Agent-oriented programming," *Journal of Artificial Intelligence*, vol. 60, no. 1, pp. 123–129, 1996.

[19] L. Gong, "Jxta: a network programming environment," *Internet Computing Online*, vol. 5, pp. 88–95, 2002.

[20] The XLattice Project, "Kademlia: a design specification," Tech. Rep., The XLattice Project, 2003, http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html.
[21] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, Wiley Series in Agent Technology, Wiley, 2007.
[22] FIPA-OS project, FIPA-OS Agent Toolkit, FIPA-OS project , 2007, http://sourceforge.net/projects/fipa-os.
[23] F. Strauss, J. Schönwälder, and S. Mertens, Jax—a java agent x subagent toolkit, July 2000.
[24] G. Nguyen, T. Dang, L. Hluchy, M. Laclavik, Z. Balogh, and I. Budinska, "Agent platform evaluation and comparison," Slovak Academy of Sciences, Institute of Informatics, Pellucid 5FP IST-2001-34519, 2002.
[25] Wikipedia, Xscale, http://en.wikipedia.org/wiki/XScale#PXA-27x.
[26] J. Lawrence, LEAP into Ad-Hoc Networks, ACM Workshop on Agents in Ubiquitous and Wearable Computing, AAMAS.
[27] M. Schoeberl, *Evaluation of a Java Processor*, Vienna University of Technology.
[28] D. Hardin, "aj-100: a low-power java processor," Embedded Processor Forum.
[29] ARM, "Jazelle—arm architecture extention for java applications," white paper, 2002.
[30] J. Maassen, T. Kielmann, and H. E. Bal, "Parallel application experience with replicated method invocation," *Concurrency Computation Practice and Experience*, vol. 13, no. 8-9, pp. 681–712, 2001.
[31] T. Brecht, H. S., M. Shan, and J. Talbot, "Paraweb: towards world-wide supercomputing," in *Proceedings of the European Symposium on Operating System Principles*, pp. 181–186, 1996.
[32] GNU.org, "The gnu operating system," June 2009, http://www.gnu.org/software/bison/.
[33] M. Hauptvogel, J. Madrenas, and J. M. Moreno, "Spindek: an integrated design tool for the multiprocessor emulation of complex bioinspired spiking neural networks, submitted to congress on evolutionary computation," IEEE CEC, 2009.
[34] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.