A Bio-Inspired Agent-Based Programming Environment for Pervasive Platforms

PhD Thesis N° XXXX (2010)

Presented at the :
Laboratoire d'Informatique, Robotique et Microelectronique de Montpellier

Directed in coordination by :
Université de Montpellier II / Université de Lausanne
École Doctorale Information Structures Systèmes

to obtain PhD grade in sciences

speciality : Systèmes automatiques et micro-/'electroniques

Manuscript in English with English and French abstracts

By

Olivier Brousse

Jury

Prof. M. Robert,	Univ. Montpellier 2	PhD thesis director
Prof. F. Grize,	Univ. Lausanne	PhD thesis director
Dr. G. Sassatelli,	Univ. Montpellier 2	PhD thesis co-director
Prof. E. Sanchez,	EPFL + HEIG-VD	Reviewer
Prof. B. Garbinato,	Univ. Lausanne	Reviewer
Prof. M. Paindavoine,	Univ. Bourgogne	Examiner
Dr. A. Gouaich,	Univ. Montpellier 2	Examiner

Montpellier, UMII
4 février 2010

Remerciements

Table of Contents

Remerciements **3**

Table of Contents **5**

1 General introduction **9**

1.1 A brief history of electronics and computer science 10

1.2 Programming paradigms 13

1.3 Systems go pervasive 14

1.4 Bio-inspiration for artificial systems 17

1.5 The Perplexus project 19

2 State of the art and proposal background **21**

2.1 Parallel Computing 22

2.1.1 Parallel platform hardware 23

2.1.2 Parallel platform Programming 27

2.2 Distributed Computing 31

2.2.1 Distributed platform hardware 31

2.2.1.1 High Performance Computing platforms 32

2.2.1.2 Grid computing 32

2.2.1.3 Wireless Grid 33

2.2.2 Distributed platform programming 34

2.3 Toward Ubiquitous Computing 36

2.3.1 Sensor Networks examples 38

2.3.1.1 SECURI*food* for cold chain management 38

2.3.1.2 Zebranet and Wildlife tracking 39

2.3.1.3 Modified MICA2 and Sniper localization 40

2.3.2 Programming Sensor Networks 41
2.3.2.1 Message oriented frameworks 41
2.3.2.2 Database frameworks 42
2.3.2.3 Application Driven frameworks 42
2.3.2.4 Agent oriented frameworks 43
2.3.2.5 Virtual Machine frameworks 44
2.3.3 Pervasive platform Challenges 44
2.4 Pervasive platform programming proposal 46
2.4.1 Agent Oriented Programming 46
2.4.2 Adaptability through Bio-inpiration 47
2.4.3 Bio-inspiration and Agent-Orientation in the literature 48
2.5 Case study platform 50
2.5.1 UBIquitous CHIP: Ubichip 50
2.6 Conclusion 53

3 Bio-inspired Agent Framework 55
3.1 A bio-inspired framework 57
3.1.1 Agent-Oriented Programming endowed with bio-inspiration . 57
3.1.2 BAF overview 59
3.1.2.1 JADE 59
3.1.2.2 JADE limitations 62
3.1.2.3 BAF: Agent organization 62
3.2 The BAF implementation 65
3.2.1 Bio-inspired agents 65
3.2.1.1 Phylogenetic agent 65
3.2.1.2 Ontogenetic agent 66
3.2.1.3 Epigenetic agent 66
3.2.2 Platform and service agents 67
3.2.2.1 Network support: Network agent 68
3.2.2.2 User interface: Host agent 76
3.2.2.3 Debugging and tracing activity 81
3.2.2.4 HW\SW communication 82
3.2.2.5 BAF in POE applications 83
3.3 Applications 86
3.3.1 Evolving simple robot controllers 86
3.3.2 Online collaborative learning 88
3.3.3 Evolving a population of learning individuals 91

3.4 Conclusion on the BAF environment 95

4 Jubi toolchain **97**
4.1 Jubi methodology and language 100
4.1.1 Unified HW/SW description file 101
4.1.2 The Jubi Language 102
4.2 Perplexus accelerator: Ubichip 105
4.2.1 Kernel loading 105
4.2.2 SIMD operating mode 107
4.2.2.1 Control instructions 110
4.2.2.2 ALU instructions 111
4.2.2.3 Architecture limitations 112
4.2.3 Ubichip modifications 113
4.3 Programming tools and validation 115
4.3.1 JubiSplitter 115
4.3.2 JubiCompiler 116
4.3.3 Code Analyzer 118
4.3.4 UbiAssembler 118
4.3.5 Functional tests 119
4.4 Experimental results 120
4.4.1 SNN simulator algorithm 120
4.4.2 GP-GPU solution: nVidia architecture 122
4.4.3 Compiling applications for GP-GPU 123
4.4.4 Experimental results 125
4.4.5 Results interpretation 125
4.4.5.1 GPU driven acceleration 126
4.4.5.2 Architectures comparison 126
4.4.5.3 Ubichip impact 128
4.4.5.4 Performance improvements 129
4.5 Conclusion on the Jubi Methodology 133

5 Conclusion **135**

A Ubibot:Moving TankBot with a Ubidule **139**
A.1 Hardware Implementation 139
A.2 Software driver implementation 139

B MANET protocol of the BAF **143**

B.1 OLSR and AODV detailed selection 143
B.2 Experimental protocols and obtained results 145
B.3 Conclusion 150

C Validation test suite **151**
C.1 Test 1 151
C.2 Test 2 152
C.3 Test 3 153
C.4 Test 4 153
C.5 Test 5 154
C.6 Test 6 155
C.7 Test 7 156

List of Figures **159**

List of Tables **163**

Bibliography **165**
General introduction 165
State of the art and proposal background 166
Bio-inspired Agent Framework 172
Jubi methodology 173
Conclusion 175
Personal Contributions - Publications 175

Index **177**

Acronyms **183**

A BAF for Pervasive Hardware Accelerated Platforms **188**
Abstract in English 188
Résumé en Français 188

Chapter 1

General introduction

Contents

1.1 A brief history of electronics and computer science 10
1.2 Programming paradigms 13
1.3 Systems go pervasive 14
1.4 Bio-inspiration for artificial systems 17
1.5 The Perplexus project 19

CHAPTER 1 ABSTRACT

This chapter presents a brief evolution in computer science and computer systems. We show how from centralized and sequential architectures, these systems have changed to parallel and distributed systems. This evolution is naturally moving toward pervasive computing, the cornerstone of the Perplexus project that is introduced in the final section of this chapter.

La Perplexité est le début de la connaissance.
"Perplexity is the beginning of knowledge."
Khalil GIBRAN .

1.1 A brief history of electronics and computer science

Since the first mechanical computing machines like the “Pascaline”, made in 1642 and presented in Figure 1.1, improving computing machines performance, increasing the number of their functionalities while reducing their size has been the leitmotiv of researches. As an example, Leibniz in 1673 improved the Pascaline to compute multiplications and divisions in an easier way using cylinders with indexes of different lengths.

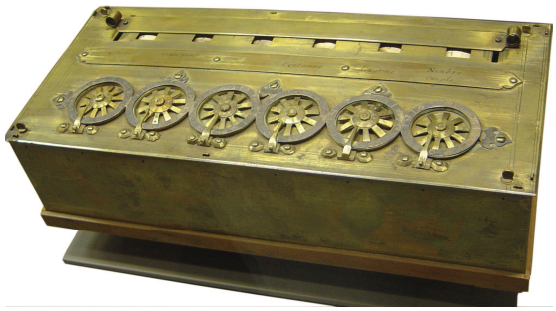

Figure 1.1: *Pascaline (Blaise Pascal, 1642)*

These improvements result from successive technological jumps and lead to several generations as presented in [14]. Early mechanical computers like Mark I, were replaced by huge electrics systems like Mark II and Mark III [2, 3] or ENIAC (19000 Tubes, 30 tons, 200 kW) [21]. Such systems of the first electronic generation used vacuum tubes.

Then, bipolar transistors-based architectures (like the Mark IV) reduced system size and power consumption while improving performance. This second generation made of large printed circuits and soldered transistors, was more reliable compared to electronic tubes. The main drawback of the bipolar technology is its inherent lack of scalability. Size of printed circuits could not be increased in a scalable manner. In addition the limited bipolar transistor size reduction did not allow successive performance

improvements and therefore reduced the interests of this technology for computer systems conception.

The last physical evolution appeared when Field Effect Transistors (FETs) have been adopted at the end of the 60's, it is known as the third generation. The FET, and obviously Complementary Metal Oxide Semi-conductor (CMOS), integration capability thanks to the transistor lithography scaling properties, led to the real emergence of computer science and microelectronic systems as power consumption, computer size and cost decreased while performance increased. For example 1MB of memory costs 76000€ in 1973 and only 0.004€ in 2005. As hardware systems became more efficient and complex to program, the need of hardware abstraction to implement applications appeared. Therefore, this third generation also marks the emergence of programming languages such as Formula Translator (FORTRAN), Beginner's All purpose Symbolic Instruction Code (BASIC), PASCAL, C and their respective compilers. UNIX OS also appeared in this generation and offered an interface to use the hardware without knowing its specificities. Between 1970 and 1975 transistor size reduced from 10 μm to 3 μm while chip size remained mainly constant.

The fourth generation (1975 - 1985) marks the starting point in using the computer science for office automation with spreadsheet, word processing or database applications. During this period the transistor feature size decreased from 3 μm to 1 μm.

The fifth generation (1985 - 1992) is associated to the appearance of three major changes: Graphical User Interface (GUI) and mice as human machine interfaces, as well as Local Area Network (LAN) to provide inter machine communications. The size of transistor decreased to 0.35 μm during this period.

The sixth generation (1992 - 2000) adds one communication level with the Internet that was primarily intended to connect LANs together. This generation marks the beginning of Personal Computer (PC) expansion's in the home environment. The fast increase of system performance led to the support of a rapidly increasing number of applications. Video games, multimedia applications, image and video processing or Computer-Assisted Design (CAD) applications are examples of the functionalities offered by computers. Once more, system performance was increasing while standard transistor channel size was decreasing to reach a size of 130 nm.

An example of this evolution is the improvement in cellular phones. Early cellular phones were exclusively able to make phone calls and weighted at least 1 kilogram.

Nowadays this kind of devices is capable of doing various things such as taking photos, videos, sending text messages or emails and running various applications (video games, Personal Navigation Assistant, etc.) while being ten times lighter.

The seventh generation gears electronics and computer systems to the invasive level with device miniaturization and the development of dematerialized interfaces such as speech processing/recognition, video detection/recognition. Since year 2000, transistors' size shrinking continued and technology based on 90 nm are progressively replaced by 65 nm (since 2007) and even 45nm since the end of 2008 (Intel core i7 processors [8]). A noticeable change occurred, due to physical constraints the chips frequency stopped to increase and even decrease a little. Multiple core processors appeared to allow performance improvements that are frequency independent. As a matter of fact, it is more feasible to integrate two or four identical computing cores - or Internet Protocols (IPs) - than to double or quadruple the device frequency. The 2007 International Technology Roadmap for Semiconductors (ITRS) forecasts that transistor channel size will continue to reduce as shown in Figure 1.2. However some experts believe that the FET technology integration capability is slowly reaching a physical limit [9].

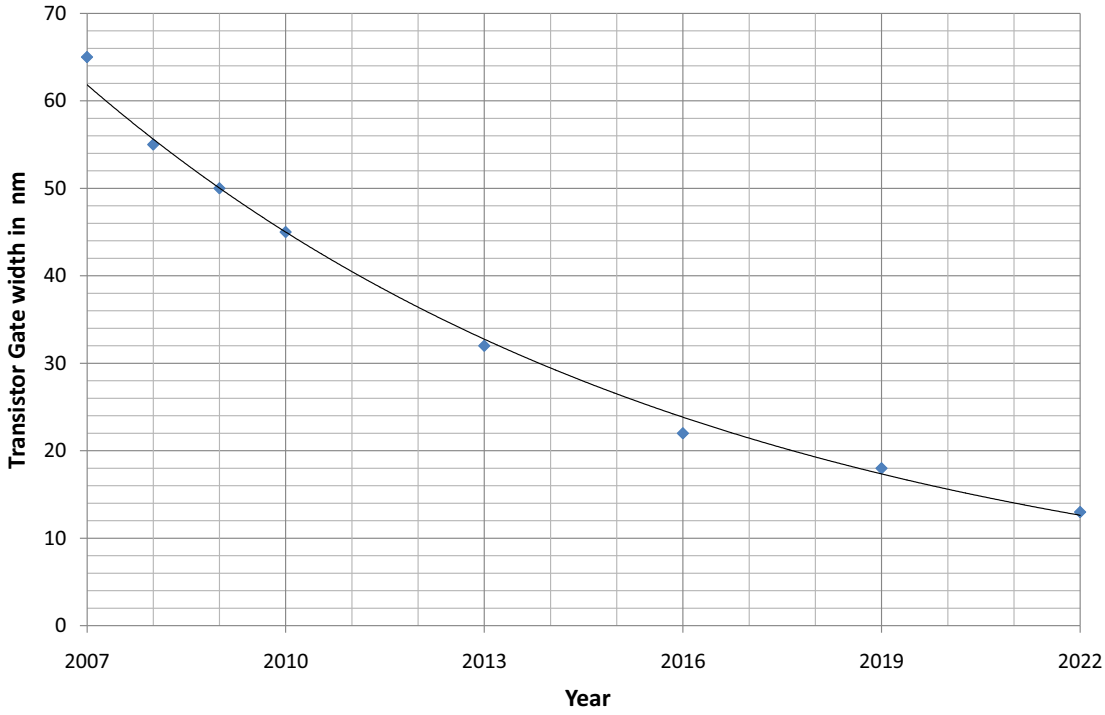

Figure 1.2: *ITRS 2007 CMOS technology forecast.*

As a consequence, future improvements might be limited. To cope with this is-

sue, different solutions are investigated, both at the technological level, such as carbon nano-tubes, photonics, DNA transistors [18], as well as at the systems level such as System In Packages (SIPs), die stacking or even distributed architectures. In this thesis the focus is put on this last solution mainly because it offers the interesting possibility to scale with the number of computing elements, and therefore copes with the numerous devices promised with the emergence of the seventh generation.

1.2 Programming paradigms

Programming parallel, distributed and even ubiquitous or pervasive architectures is not as simple as programming sequential systems. Specificities of parallel architectures have to be taken into consideration for an efficient programming of the platform. This parallelism may imply to describe applications in separated parts that are then executed on different hardware hosts. Application coherence therefore involves data and program controls to be exchanged between hosts. Dealing with these communications increase programming complexity, essentially because most of industrial programming languages have not been designed with parallelism in mind.

As an example, imperative languages are focused on the sequential aspect of a program; functions are called successively to be applied on elementary data. Data parallelism is not exhibited and requires efforts to be exploitable; moreover system level functionalities have to be used to allow task parallelism. The more recent Object-Oriented Programming (OOP) paradigm eases the task parallelism as data are abstracted into objects; they are therefore easier to manipulate from different software parts. Nevertheless, parallelized application still requires being described with dedicated functionality; as most languages still not include parallelism support.

The evolution brought by Agent-Oriented Programming (AOP) paradigm is an implicit support of these communications. In 1993, Yoav Shoham first proposed to describe applications as sets of cooperating independent agents. Based on a sophisticated communication model, AOP presents a new programming paradigm promoting a social view of computing where agents are loosely coupled and interact in a similar manner as individuals [16]. Agents are therefore designed as elementary parts of an application; they have their own functionality and data set. The opportunity to share data and programming controls is offered to agents thanks to message exchanges. As

communications are based on messages, whenever agents are hosted on different hardware or not, it is therefore possible to transmit them over distributed platforms using adequate communication media, such as shared memory, wired or wireless networks. The sum of these agents and their runtime interactions produce the global application functionality.

Complex parallel applications can be modeled using the AOP as every agent is independent and may exhibit its own characteristics. Therefore by multiplying the number of agents, one may represent complex interactions schemes, such as social interactions or swarm behaviors. This may be done in sequential, parallel or distributed systems using all the available platform parallelism. Therefore AOP is an interesting solution to have massively parallel applications executed.

The AOP offers then an interesting solution to program distributed and even pervasive platforms. This solution has been exploited with success in the first contribution presented in this thesis.

1.3 Systems go pervasive

Figure 1.3 shows the evolution of computing eras in time and their corresponding programming paradigm. It represents the different steps that led computer systems from "mainframes" of the first to fourth generations, to upcoming Ubiquitous Computing era of the seventh generation. During the mainframe era several people used one computer as depicted in the rectangle 1 of figure 1.3.

The generalization from scientific, business and administrative electronics to consumer electronics also influenced the evolution of computing systems. This trend signals the beginning of the personal computing era where one person uses one computer as shown in figure 1.3. This consumer model is always demanding new applications or services and increased performance. The reverse exists as new electronics and computer science systems are creating new markets that will, in turn, demand for more performance; video games and cellular phone markets are meaningful examples of this duality. Progressively, and thanks to the electronic miniaturization, microelectronic systems have been embedded in daily used devices, and mobile computing era started

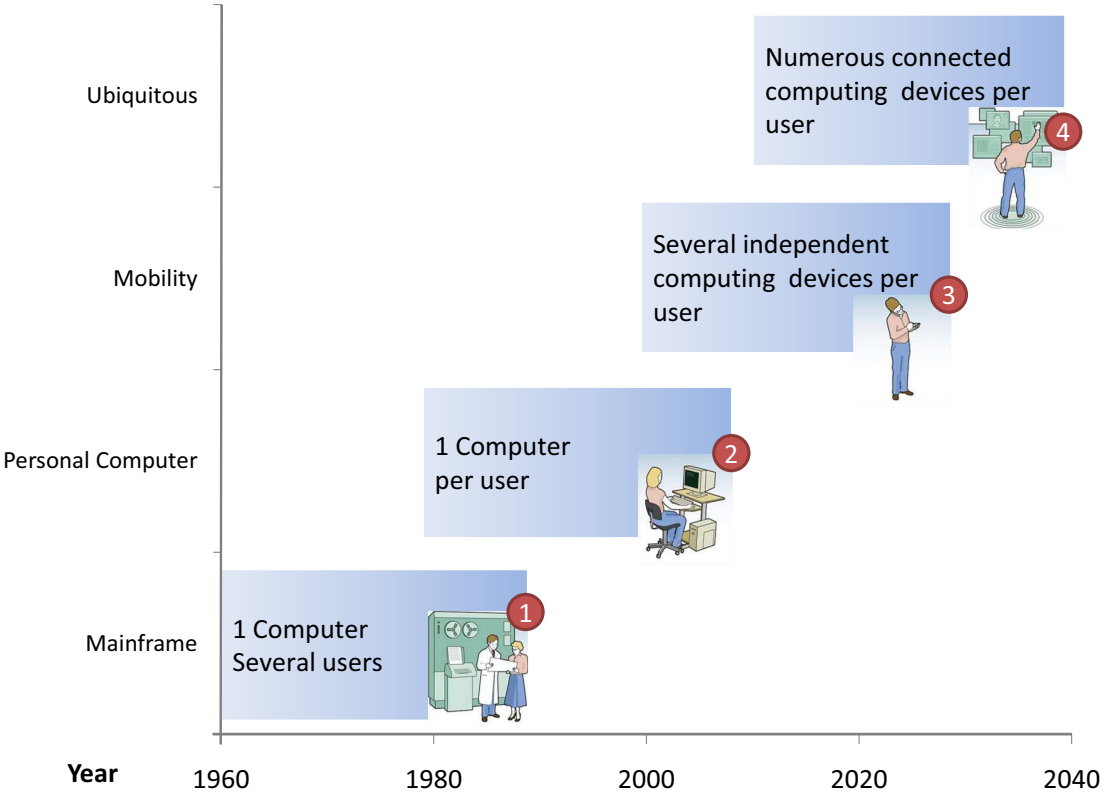

Figure 1.3: *Evolution of the computing paradigm.*

as one person uses several distributed and even mobile devices (cf. rectangle 3 in figure 1.3). Nowadays computing systems have invaded human environment: everyone is using more than 200 chips in a day without knowing it.

Therefore with the increasing mobility and connectivity demand, with Internet 2.0 or mobile Internet examples, and telecommunication evolutions (Bluetooth, Universal Mobile Telecommunications System (UMTS), Internet Protocol version 6 (IPv6), Worldwide Interoperability for Microwave Access (WIMAX), Power Line Communication (PLC) [22]), a perspective may be that every device will be connected. This network of computing nodes should be able to support applications addressing the modern human needs like intelligent control of the environment or permanent and immediate social interactions. We may therefore consider that a new computing era is emerging.

Therefore ubiquitous computing seems to be the natural evolution of mobile computing. This corresponds to the current situation where everyone uses electronics in everyday life with a still growing number of devices, connectivity, interoperability and

environmental integration. All of these notions can be abstracted by an evolved and active Sensor Networks (SNs).

Numerous applications are using sets of sensors distributed in the environment. Such systems that support these applications are referred to as Sensor Networks in this document. These systems made of small communicating modules may be used to monitor environmental or physical conditions by cooperating in the SN deployment area [13].

From a set of nodes a huge pervasive platform may arise if inter-node communications are possible allowing the platform to achieve tasks that only one element would not have been able to do. Such tasks are referred to as complex systems. Ant colonies, human economies and social structures, climate, nervous systems and habitat monitoring are examples of studied complex systems.

This pervasive computing scheme, also called ubiquitous computing, brings capabilities to simulate complex systems compared with pure SNs that are more dedicated to data gathering and signal processing tasks [1]. Ambient intelligence also brings new challenges either at informatics (Software (SW)) or at microelectronics (Hardware (HW)) levels due to their heterogeneous characteristics.

Based on this observation pervasive platform can be considered as a high-end generalization of Sensor Networks, where motes (SN elements) can provide more computing power, a larger memory capacity, interoperability and connectivity. Contrarily to a classical SN, such ubiquitous systems are made of potentially heterogeneous elements (referred to as nodes in this document). Those nodes therefore cooperate to achieve given tasks sharing services and data in an adaptive way.

Since nodes may be mobile, as for example cellular phones, they can join or leave a platform at anytime. The result is then an unstable platform topology with probably unreachable nodes and unavailable services at a given time. Platform adaptability becomes therefore a key factor for efficient pervasive systems because deteriorated mode may occur for various reasons like environmental influence, or defects concerning: unavailability of a desired service (SW), unavailability of a node hardware (HW) or cooperating node (HW, SW, Communication) for example.

As a consequence, several questions arise concerning the capabilities and management of ubiquitous platforms:

- How to program and manage them?
- For what kind of applications?
- How to improve the platform adaptability?
- How to improve node performance in such context?

These questions are studied since the early 2000's on SN platforms but pervasive systems require dedicated answers due to their specificity. The work presented in this document gives an approach that attempts at answering these questions. This study has been conducted during the Perplexus[1] European project ubiquitous computing platform [12].

1.4 Bio-inspiration for artificial systems

For centuries, scientists are interested in how life appeared and what the specific mechanisms that make life possible are. From these basic questions a new discipline called bio-inspiration emerged. It relies on the study and the imitation of natural processes to achieve tasks. In such a domain different biological process granularities are used. As an example, Swarm behavior or fly vision are coarse grain processes and provides direct functionalities (network adaptive packet routing, motion adaptive vision system, etc.). Finer grain processes rely on life elementary principles.

In [17] authors present an artificial life model based on three major biological principles. This model is applied to computing systems as a proof of concept that nature may be a bridge between natural sciences and engineering sciences disciplines. Moreover it is intended to improve performance of computing systems when simulating natural processes as well as systems adaptability.

The three presented principles of life are:

- Evolution of species allows species perpetuating their existence. This is achieved by adapting the species individual characteristics to the environment. This process relies on the natural selection and is usually referred to as Phylogenetic (P).

[1] PERvasive computing framework for modeling comPLEX virtually Unbounded Systems (Perplexus)

- The successive divisions of a mother cell that contains the individual characteristics allows the individual growth and self healing. This process where cells are divided and differentiated is referred to as Ontogenetic (O).

- Individual genome involved in both previous levels cannot specify the whole individual. Acquired experience both at the conscious or unconscious levels (nervous or immune and endocrine systems) is the third essential aspect of life and changes during the whole individual lifetime. This is known as Epigenetic (E).

Artificial Phylogenetic processes can be illustrated with the example of Genetic Algorithms (GAs) introduced in [7] and [4]. This family of algorithms is based on a Darwinian view of evolution. As a consequence, natural selection principle is applied to a population of individuals (i.e. a set of solutions for the given problem) to find what are the better adapted individuals. This general principle is well suited to investigate problem with a huge space of solutions. As for natural evolution, only the better adapted individuals of a generation are selected for reproduction. Therefore every individual is evaluated and compared to other what requires a *fitness function* to evaluate the solution domain. Then best individuals are crossed (solutions are fused) and or mutated (solutions are slightly modified) to create a new generation. This new generation is then evaluated until an acceptable solution is found. This large solution space investigation proved to be faster than basic or random complete explorations of solution space.

To illustrate the third axis, the Artificial Neural Network (ANN) example may be mentioned. The concept of a neural network has been proposed by Alan Turing in 1948 and published in [19]. This principle is based on real neuron studies of the 19th century. The learning capability of a set of interconnected artificial neurons has been demonstrated with the Perceptron model in 1957 by Frank Rosenblatt [15]. This model was able to execute during a *recall phase* or *execution phase* relatively complex logical equation after a *learning phase*. Several neuron models and learning algorithms have been proposed as Spiking neurons and back-propagation, other models are referenced in [23].

The association of these three principles is referred to as the Philogeny Ontogeny Epigenesis (POE) model and is extensively described in [17].

1.5 The Perplexus project

Among the several targeted features of the Perplexus[2] platform, scalability and ubiquity are of utmost importance for the contributions presented in this thesis. Such a platform is dedicated to the simulation of large-scale complex phenomena. These complex systems are composed of interconnected simulation modules, that as a whole, exhibit one or more properties not obvious from the properties of the individual parts. Therefore such systems are not deterministic and exhibit emerging behaviors.

Large-scale phenomena simulations are typically out-of-reach of classical sequential simulations due to their high complexity. Therefore they imply using massively parallel or dedicated simulation platforms. These platforms traditionally either belong to the supercomputing class [10] [5] or to the emerging network-centric distributed simulation class [20] [11] with a continuously growing number of computing resources. Therefore the Perplexus platform will require scalability to allow the platform to grow and be adapted to the targeted applications.

Simulating complex phenomena often requires modeling environmental interactions which is a source of inaccuracy and computing overhead (statistics, random number generation, etc.). The Perplexus platform overcomes this issue by having interactions directly taking place at the module (or node) level through application-specific sensors and actuators. As a consequence, ubiquity obtained by spreading modules over the environment associated with the knowledge and interactions with this environment are key factors for the platform.

To reach the presented objectives, scalability and ubiquity are required. As a consequence, the proposed platform is composed of many ubiquitous computing modules called Ubidules. Scalability is ensured thanks to the distributiveness of the platform and its decentralized communication scheme. Nodes are considered as peers and join others with a peer-to-peer scheme. This distributed specificity also provides ubiquity since the computing modules are not only interacting with each other thanks to a wireless network but also with the environment through sensors and actuators.

These characteristics make the system similar to SN [1] with the corresponding identified challenges such as the difficulty of achieving a global synchronized time

[2]PERvasive computing framework for modeling comPLEX virtually Unbounded Systems (Perplexus)

reference and reliability of the communications. The platform can be considered as an EveryWare enabled platform as presented by Adam Greenfield in [6] and consequently requires adaptability.

To ensure the system adaptability, bio-inspired features are embedded in the platform either at SW or HW levels. Nodes are based on a bio-inspired reconfigurable chip (Ubichip). It provides bio-mimetic capabilities to the platform. This point will be developed in Section 2.4.2.

The approach to simulate complex systems as well as the inherent constraints of the Perplexus project are innovative. To find reliable solutions that allow ubiquitous platforms programming requires studying the state of art. In addition some pre-requisites definitions have to be introduced to present the guidelines of this thesis work. Chapter 2 deals with both of these points.

Chapter 3 presents the programming framework we have developed for ubiquitous computing platforms. In this chapter we also demonstrate how this framework operates on the Perplexus platform through three robotic applications.

Chapter 4 will relate to the work done at the module level for optimizing application using their intrinsic parallelism.

Finally chapter 5 will draw some conclusions and perspectives that come with the presented contributions.

Chapter 2

State of the art and proposal background

Contents

2.1 Parallel Computing 22
2.1.1 Parallel platform hardware 23
2.1.2 Parallel platform Programming 27
2.2 Distributed Computing 31
2.2.1 Distributed platform hardware 31
2.2.2 Distributed platform programming 34
2.3 Toward Ubiquitous Computing 36
2.3.1 Sensor Networks examples 38
2.3.2 Programming Sensor Networks 41
2.3.3 Pervasive platform Challenges 44
2.4 Pervasive platform programming proposal 46
2.4.1 Agent Oriented Programming 46
2.4.2 Adaptability through Bio-inpiration 47
2.4.3 Bio-inspiration and Agent-Orientation in the literature . . . 48
2.5 Case study platform 50
2.5.1 UBIquitous CHIP: Ubichip 50
2.6 Conclusion 53

Chapter 2 abstract

In this chapter the evolution of computer systems toward Ubiquitous Computing platforms is illustrated. Parallel computing and distributed computing platforms are then presented to introduce these thesis contributions. As Sensor Networks are at the boundary between distributed and pervasive platforms their programming is presented as a background context of the work presented in this thesis. Additional definitions introduce the background for the presented contributions. To conclude this bibliography we introduce the case study platform used for testing the proposed solution.

La recherche est un processus sans fin dont on ne peut jamais dire comment il évoluera. L'imprévisible est dans la nature même de la science.

"Reseach is an endless process which no-one could tell how it will evolve. The unpredictable is in the nature of science."

François JACOB .

As introduced in Chapter 1 with mobile and ubiquitous eras, the trend in systems improvement is no more based on frequency increase and micro architecture optimization. Due to physical constraints device frequency is no more increased to improve system performance. The optimization of micro architectures is really costing and do not increase performance significantly. Thus other solutions already used in supercomputing are investigated and integrated to increase microelectronics systems computing power. The following section presents, firstly some of those solutions known as parallel architectures. Secondly, some distributed computing systems are introduced. Finally, focus is put on sensor networks and their computing framework, as they are the link between distributed and pervasive architectures. Based on identified pervasive computing challenges, some solutions are proposed to program ubiquitous platforms; the background of these solutions is therefore presented. In the final section of this chapter the case study platform is presented.

2.1 Parallel Computing

Since 2005, multiple cores General Purpose Processorss (GPPs) have been proposed to improve system performance. The principle is then to use the die surface to connect through unified cache memories several cores that are able to execute several applications in the same time. For instance Intel core, core 2 and core i7 architectures [8] or AMD Athlon64, Phenom and Phenom 2 [26] are 2, 3 or even 4 core architectures with up to 8MB of unified cache memory. Embedded processors follow the same trend. As

an example we can cite the ARM MPCore configurable IPs with up to 4 cores [27] implemented in Qualcomm snapdragon QSD8672 [75], TI OMAP 4 family [86] and nVidia Tegra [70]. This is relevant of the changes in microelectronics design paradigm and therefore of computing era. Even if parallel architectures exist since decades in supercomputers they are now emerging as a standard solution to improve performance in general purpose and embedded computing systems.

This hardware trend has a great influence on application programming. As a matter of fact, the parallelization of an application requires separating it into pieces. Such pieces are known as threads or processes and will referred to as SW modules in this thesis. Additionally, communications between these SW modules have to be ensured to obtain the final parallelized application. Two general principles are then used in the literature: implicit and explicit communications. Communications through a *shared memory* can be considered as implicit as data accesses are globally possible at any time without actual communications between SW modules. Explicit communications take place in the case SW modules actually exchange data by sending messages. Architecture using this communication trend are referred to as *message passing* and usually do not rely on shared resources. The following sections then present some parallel solutions and the associated programming methods.

2.1.1 Parallel platform hardware

Micro architectures such as Single Instruction Multiple Data (SIMD), Multiple Instructions Multiple Data (MIMD), superscalar and Very Long Instruction Word (VLIW) are examples of literature parallel processors solutions. Their respective specificities are listed here after:

- **SIMD** architectures are able to execute one instruction over several different data. The processor is then using data parallelism as one instruction is applied over the whole Processing Elements (PEs) of the architecture. SIMD architectures are usually used as instruction set extensions for GPP: MultiMedia eXtension (MMX), Streaming SIMD Extension (SSE) and equivalent instruction set families are typically SIMD processors applied to multimedia processing inside GPPs.

- **MIMD** processors are more flexible than SIMD as these architectures may use data and tasks parallelisms at the cost of an increased complexity and resources. Data and instructions memory(ies) may be shared or distributed between PEs. These PEs may be interconnected using buses or hierarchies for shared memory architectures. In the case of distributed memory architectures meshes or hypercubes are used as interconnect. MIMD principles are used either on-chip or off-chip. Multiple processors PCs, MACs, and servers are therefore examples of shared memory MIMD architectures. The HS-scale platform is an example of on-chip MIMD architecture [79].

- **Superscalar** micro-architectures are made to exploit the instruction level parallelism. Such processors are based on duplicated functional units and therefore accept multiple instructions per clock cycle. These instructions come from a sequential instruction stream. Therefore data dependency between parallelized instructions is processed by the hardware. Internal architecture of current x86 processors, since the first generation of Pentiums is superscalar.

- **VLIW** architectures are an alternative to superscalar architectures with a simplified structure as data dependency is no longer processed by the hardware. Data dependency is then verified at compile time therefore allowing the compiler to parallelize the program. Parallel instructions are concatenated into very long instruction words that are executed in the processors parallel datapaths. As dependency checking is achieved during compiling, the compiler performance and the generated code have critical impacts on architecture performance. NXP TriMedia media processors family, SHARC Digital Signal Processor (DSP) by Analog Devices, C6xxx DSP family by Texas Instruments, and ST Microelectronics ST2xx family are contemporary examples of VLIW processors. They are used as embedded media processors for consumer electronic devices.

Such parallel architectures that use the Instruction Level Parallelism (ILP) are nevertheless limited by the increasing difficulty of finding enough parallelism in a single instructions stream to keep a high performance single-core processor busy according to [45]; this limitation is called the *ILP wall*. This limitation can be easily overcome while using the Task Level Parallelism (TLP) that enables significant speedups and proves more flexible than ILP. As a matter of fact, more parallelism is available - and usually less coupled - in a complete application than is a small instruction stream.

Parallel architectures are also characterized with respect to their memory management. Usually macro-architectures with small amount of computing devices are based on shared memory as presented in Figure 2.1. This memory management provides fast inter-processor data communications with overhead and delays. These parameters depend on the number of devices sharing this memory. Two different threads running on different hardware hosts of a parallel cluster can easily share data. Accesses to shared data are secured thanks to semaphores that prevent data inconsistency with a minimal overhead.

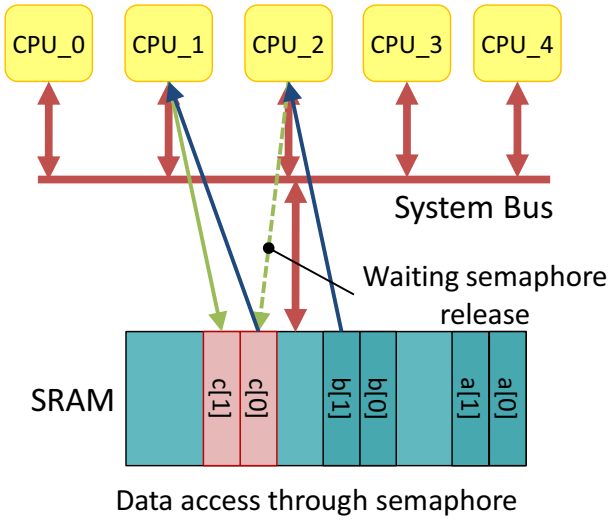

Figure 2.1: *Example of parallel architecture with a shared memory*

The lack of scalability in shared memory architectures is a major drawback. Hierarchies of memory, such as multiple level cache memories, are then used to limit the impact of the shared memory bottleneck. Figure 2.2 shows an example of memory hierarchy that includes 3 cache levels and one system memory. In the presented case each core has its own *L1* cache, shares its *L2* cache with the neighboring cores and has an access to a *L3* cache shared by all 4 cores of the processor. During runtime when a cache miss occurs request are sent to the cache upper level (dark red arrows) until the data is found (light green arrows). System memory accesses are reduced if the cache policy is efficient reducing the memory bottleneck.

The same problem is encountered at the communication level. Bus based communications can be used for parallel architectures with a small amount of connected devices, Direct Memory Access (DMA) accesses and bus mastering limit communication bottleneck and obviously improve system performance.

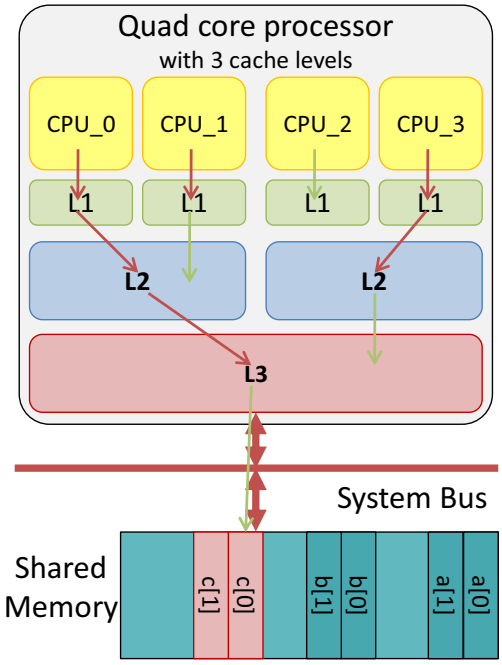

Figure 2.2: *Example of parallel architecture based on memory Hierarchy*

Analogous to memory hierarchy, hierarchical bus based architectures are less sensitive to congestions at the expense of increased delays in inter-level communications. An example of parallel such architecture is the general personal computer architecture as presented in Figure 2.3. Generally in these platforms Graphics Processing Units (GPU) are connected to GPPs. General Purpose on Graphics Processing Units (GP-GPU) allows using the GPU to accelerate applications taking advantage of the system Central Processing Units (CPU) and GPU parallelisms [24, 72].

Figure 2.3 depicts how operates a system based on bus and memory hierarchies. In this case a GP-GPU application is considered. CPU 1 hosts the main program of the application. It is able to initiate the copy of data from system memory to graphical board memory. This command makes use of several system drivers (Northbridge, Peripheral Component Interconnect Express (PCI-E) and GPU) to have data transmitted from the system memory to the Northbridge then to GPU through system and PCI-E buses.

The cost/performance ratio of such devices as well as the opportunity to accelerate personal and scientific applications such as MatLab [71] explains the growing interest for the GP-GPU solution in numerous scientific fields [78, 93, 41]. Devices such as nVidia GTX 295 are theoretically able to provide 1192.3 Gflops [69], as the AMD

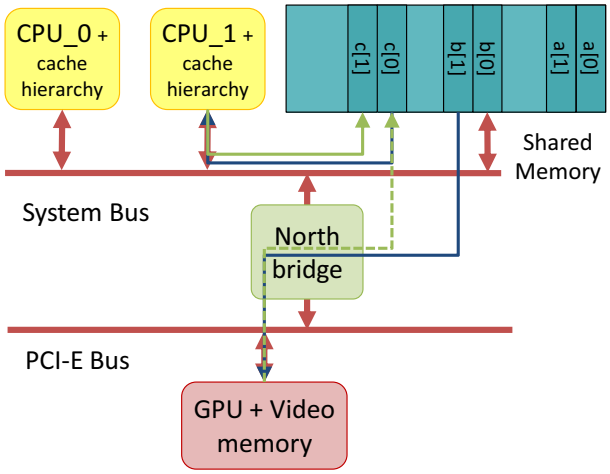

Figure 2.3: *Example of a parallel architecture with bus and memory hierarchies*

HD4780 X2 equipped with 2 GPU achieves 2400 Gflops [25] for price below 500€ while high-end current GPPs perform around 70 Gflops.

The real alternative to shared memory and bus based architectures, that cannot scale in term of number of computing devices, are *private memories* architectures where data are exchanged through Message Passing (MP) architectures. Figure 2.4 shows an example of parallel architecture made of interconnected elementary systems (i.e. CPU and its own private memory). Such systems are connected through different variations of interconnection networks for example mesh, torus, rings and hyper cubes.

Current research in the Multi-Processor System on Chip (MPSoC) field are using such architectures inside of a chip to improve the processor level scalability. These architectures will be useful to replace cache hierarchies in the context of a processor that embeds hundreds to thousands of cores.

Contrarily to sequential programming, parallel platform programming requires to exhibit parallelism to map in an efficient way the application on the architecture. The following section presents different state of the art solutions to program parallel architectures.

2.1.2 Parallel platform Programming

In order to program parallel platforms, applications have to be split into parts that can be executed simultaneously. These parts are commonly known as threads [34]. Compared with sequential application description, multi-threaded application programming

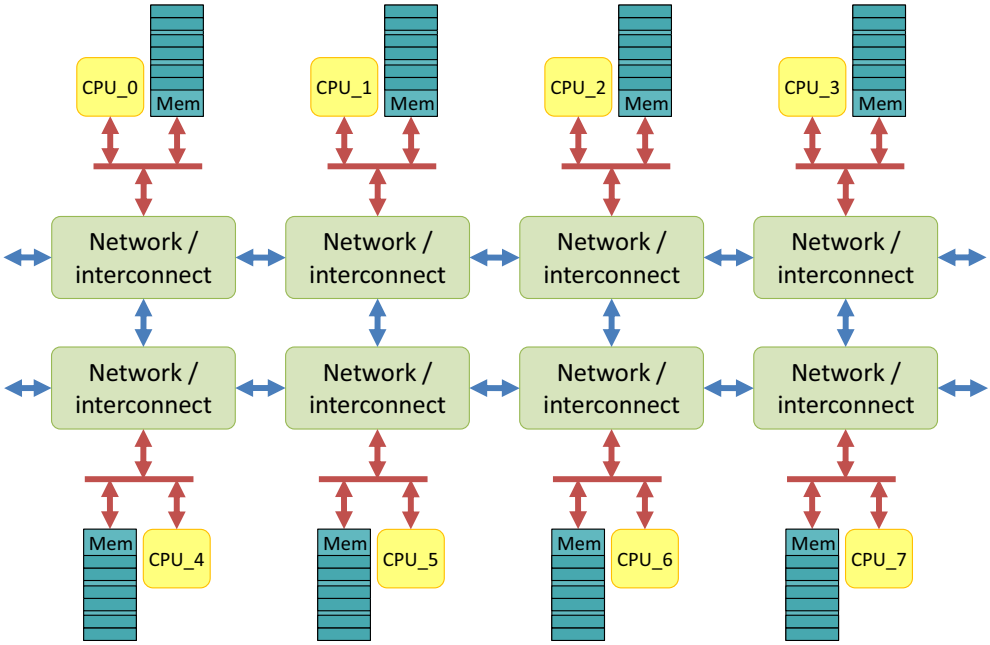

Figure 2.4: *Example of parallel architecture based on private memories and network based communications*

requires additional features. These features are presented in Figure 2.5. Data dependency and protection have to be ensured as data are accessed from several independent sources. The *semaphore* mechanism introduced by Edsger W. Dijkstra [37] to provide control on processes that intend to access to a shared resource, in parallel or concurrent programming contexts. This principle also allows synchronizing threads by establishing *barriers*. A barrier for a group of threads or processes is a wait point that cannot be proceed until all other threads or processes reach this barrier. This synchronization is therefore useful to describe application with nested sequential and parallel sections and inter threads data dependency as shown in figure 2.5.

To efficiently use the available parallelism of these multiple processor in shared memory systems, Application Programming Interfaces (APIs) such as Open Multi-Processing (OpenMP) are good alternatives to the classical thread programming [35]. OpenMP supports parallel programming in C/C++ and FORTRAN on all classical operating systems, including UNIX, Linux, Mac OS and Windows platforms. OpenMP is a portable model but is limited to shared-memory parallel systems ranging from the desktop to the supercomputer at the cluster level.

A common problem with threads and OpenMP programming is the difficulty to guarantee that the application is deadlock free. User should therefore take care of this

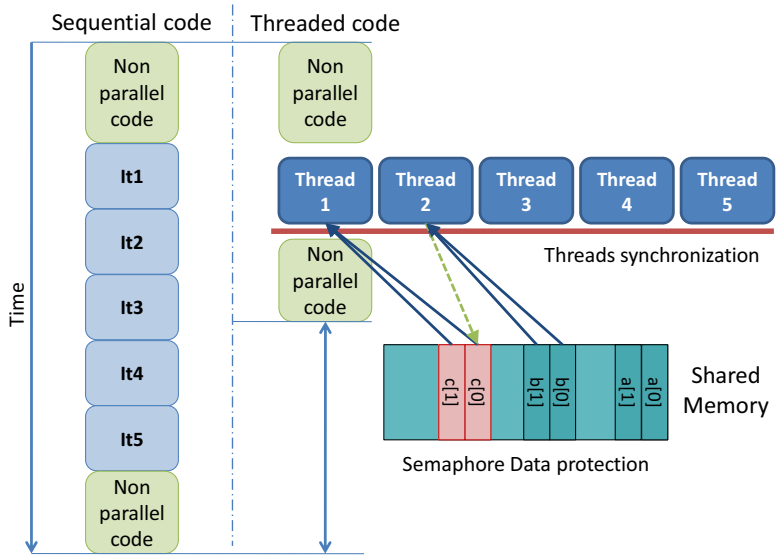

Figure 2.5: *Thread programming on parallel architectures*

common problem in multiprocessing. In the literature, there exist solutions to avoid most of deadlocks. Systems such as the *Banker's algorithm* [36, 38] or Petri nets [73] are used to avoid deadlocks. Readers interested in finding additional solutions are invited to refer to [34] and to the bibliographic article by Zöbel [92].

The use of Message Passing Interface (MPI) (i.e. explicit data communication between SW modules without using a shared memory) in parallel platforms is somehow limited to the programming of specific platforms that are usually intended to execute dataflow applications. Such applications are usually executed on hardware platforms that offers message optimized communication links between the platform PEs.

Describing applications with threads, OpenMP, MPI or using higher level programming such as Synchronized Distributed Executive (SynDEx)[54, 55], usually rely on classic or Object Oriented programming. OOP was introduced in the early 1960s' at the Massachusetts Institute of Technology (MIT). OOP is a programming paradigm based on an "objects" view of data and their dedicated methods. Objects interactions allow designing applications. This paradigm has been introduced as a way to maintain software quality when system complexity increased and to allow methods interoperability and source code reuse. The use of OOP impacts the application performance, as an abstraction layer is added compared to standard computing. Nevertheless it offers a more secured programming scheme as data gathered into larger objects are easier to use or reuse.

As introduced in section 1.2, programming languages have been designed mainly for sequential computing. The emergence of GP-GPU parallel platforms led graphical board manufacturers to investigate how to program their platforms to provide efficient GP-GPU computing systems. nVidia CUDA and ATI Stream Computing solutions have been proposed as APIs that allow users to use the computing resources of their GPU in addition to their CPU computing power. Nevertheless strong proprietary restrictions limit the use of an API to platforms of a given manufacturer. On the contrary, the Open Computing Languge (OpenCL) standard [52] intends to give a unified API to program heterogeneous platforms made with multiple core CPUs and GPUs. Based on C/C++ this API is closed to CUDA language and offers clear parallel programming directives. It aims to ease multiple core and multiprocessors CPU programming in addition to the GPU programming. The Microsoft DirectCompute is an alternative to OpenCL but as it is included in DirectX11 it is then limited to windows applications.

Figure 2.6 summarizes this programming and hardware paradigm. As a conclusion, parallel computing allows performance improvements at the cost of an increased programming complexity. Nevertheless, these improvements are limited by the small amount of devices that can be assembled inside a system due to the shared resources. In the following these parallel systems are referred to as clusters.

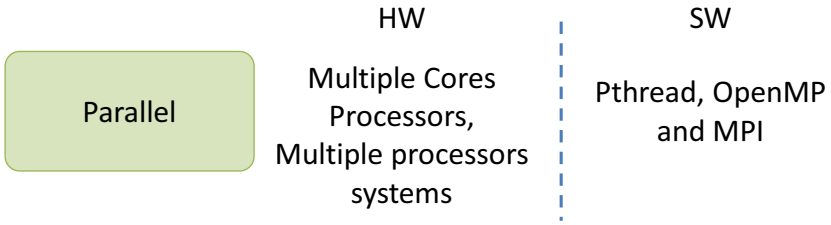

Figure 2.6: *Parallel Computing paradigm*

The problem with multiple core processors, System on Chip architectures and more generally parallel architectures is their lack of scalability either at integration level or at memory level. If the CMOS technology finally reaches its integration limits then the number of computing elements will not be increased in scalable manner. The shared memory paradigm may also become a bottleneck issue as the memory bandwidth has to be shared between computing elements. As a consequence the on-chip and even on-board systems scalability may be limited in the future.

2.2 Distributed Computing

Another architectural level has been used in supercomputing to overcome the lack of scalability of parallel cluster. It increases the number of assembled computing elements by using the spatial distribution of parallel computing resources is presented in Figure 2.7. This allows gathering several parallel platforms into a more efficient system. Distributed computing is a solution to the parallel computing scalability issue and therefore offers more computing power. High Perfromance Computing platforms (HPCs) and computer grids distributed platforms are based on this principle of cluster interconnection to obtain massively parallel and high performance platforms.

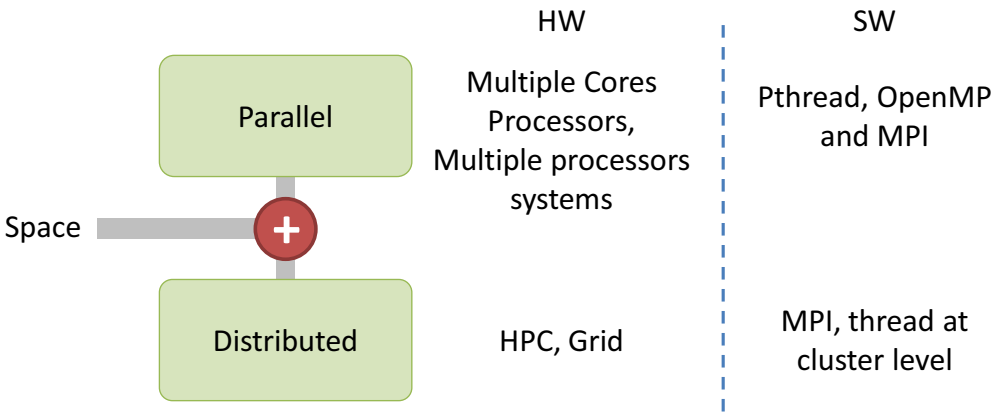

Figure 2.7: *Distributed Computing solution*

2.2.1 Distributed platform hardware

The main critical point in distributed architectures is at the communication medium level. A distributed architecture is made by assembling several systems (referred to as clusters) thanks to a network. Every cluster has its own memory but no global shared memory is possible for scalability reasons as exposed previously. Nevertheless, some mechanisms are used to duplicate data over the architecture at the cost of decreased performance. Therefore these architectures are based on the MP methodology. Different types of distributed platforms exist; they are classified in the following sections depending on their communication media and the scale of the platform.

2.2.1.1 High Performance Computing platforms

High Perfromance Computing platforms (HPCs) are usually made of clusters connected through high speed and short length network links. The global principle of HPCs distributed platforms is shown in Figure 2.8. The scale of such platforms is generally restricted due to several reasons such as the size of cluster, its cooling mechanism, the cost and length limitations of the high performance interconnect. Clusters are generally based on shared memory architectures and therefore programmable thanks to threads as exposed in the previous section.

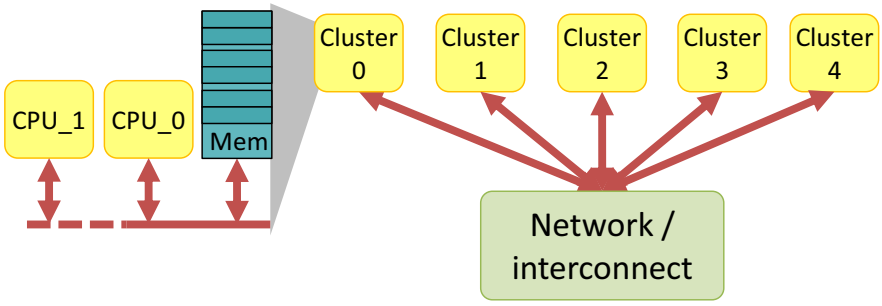

Figure 2.8: *HPC general principle*

These powerful clusters are usually interconnected through very high bandwidth network links such as 10 Gigabits Ethernet or InfiniBand links [49]. These offer respectively 10.3125 Gbps and up to 96 Gbps with range from 1 m to 10 km and transfer latency in the order of 1 μs. They allow to create static and efficient MP architectures within a short area range. Therefore parallel applications may be executed on such platforms with a reasonable communication overhead using threads at cluster level and MPI at the HPC level.

2.2.1.2 Grid computing

In [95] Ian Foster and Carl Kesselman defined the grid concept as follow: *"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."*.

In comparison with HPC, the grid computing also called *cloud computing* extends the platform scale, network supports and spatial distribution. HPCs are commonly located in a same place and composed a homogeneous platform. When dealing with

computer grids resources are more likely heterogeneous and geographically distributed as presented in Figure 2.9. Such systems form of network-distributed parallel processing architecture.

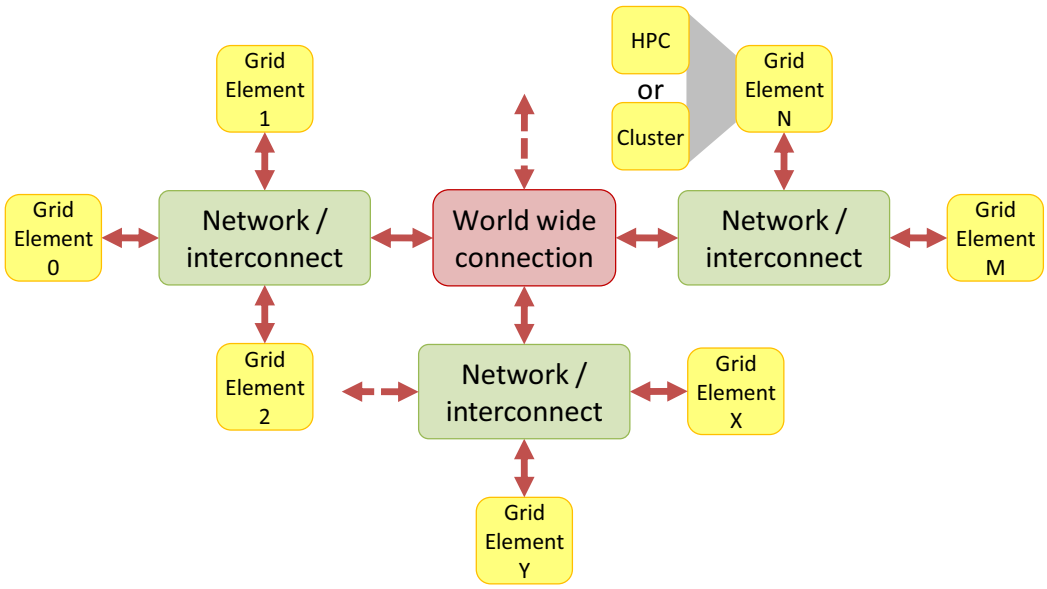

Figure 2.9: *Grid general principle*

The size of grid computing may vary from a confined network of computers to large public collaboration across many companies and networks. The scale of such platforms in term of number of computing element may be larger than in HPCs as it involves more computing nodes. Nevertheless network links that support inter-cluster (or inter-HPC) communications in the grid are slower as long range communications are involved. Such communications are usually done through internet links.

This reduced communication bandwidth associated to the large number of computing nodes make grids suitable for massively or ultra parallelizable applications with minimal communications. Therefore, these applications must have a very low amount of data dependency.

2.2.1.3 Wireless Grid

With the arrival of the mobility computing era, networking becomes wireless. With the emerging of wireless and even mobile networking connections, the classical grid computing can be extended to the wireless scheme as shown in Figure 2.10. Wireless

grids are therefore made with mobile embedded devices with less computing power and a reduced bandwidth. Therefore the grid topology is dynamic, and may require dedicated network features to allow topology to change dynamically and therefore mobile nodes may be easily included in the grid.

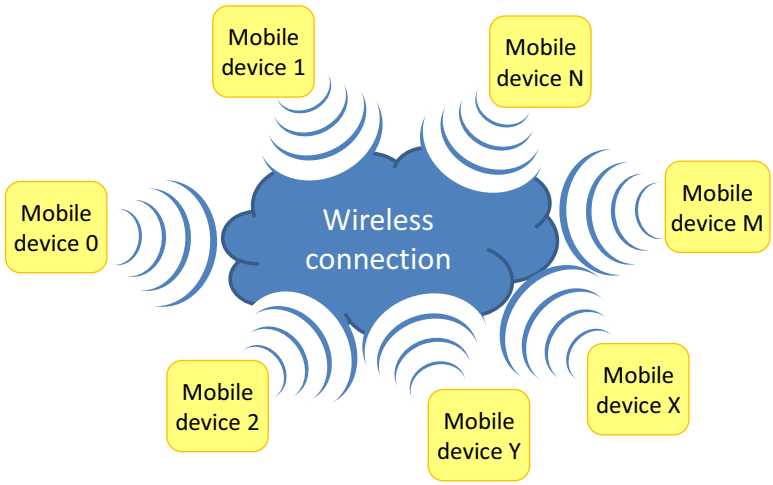

Figure 2.10: *Wireless Grid general principle*

Even if platforms scales, and network supports differ, the programming paradigm of such platforms is common as exposed in the following section.

2.2.2 Distributed platform programming

The absence of shared memory and resources allows increasing platform scalability but requires using another solution to share data in distributed applications. The Message Passing (MP) principle is a form of communication that is used to share data in this distributed context. In MP systems, threaded programs can communicate through messages that include function invocation, signals, and data packets. "Actor Model" [46] and "process calculi" [28] models of computations are based on MP. Message passing is implemented in systems like ONC RPC, Corba, Java Remote Method Invocation (RMI), DCOM, SOAP, .NET Remoting, QNX Neutrino RTOS, OpenBinder, D-Bus and similar. The term is also referred in High Performance Computing to as Message Passing (MP).

MP is usually coupled with Object Oriented Programming. In such systems, a message is the single means to pass control to an object. Therefore objects should be

described to allow sending messages from within their method bodies. In this way, MP enables runtime thread binding and synchronization. This therefore improves systems adaptability to runtime context.

Application mapping on distributed platforms using Object-Oriented Programming, local thread and global Message Passing Interface is really complex and greatly affects application performance if these principles are not mastered. Another abstraction level may be useful to ensure application programming quality and performance. As an example, some Java parallel classes like Java Parallel Class Library (JPCL) [32] add software parallelism to the classical Java concept but even in this case Java Virtual Machines (JVMs) are running on a single processor. Thus, the use of such libraries requires a framework that links several JVM running on several hardware targets and needs a shared global object space. Another approach is proposed with Manta [60] and relies on compiling Java threads to native x86 assembly code. These threads are then executed on a x86 cluster through RMI with an MPI communication. Therefore the whole application becomes hardware dependent with a strict limitation on the available type of architecture.

An evolution of grid computing emerged with web-based applications like Berkeley Open Infrastructure for Network Computing (BOINC) class [20], or folding@home [53]. Such applications are built around the volunteer computing principle and rely on Internet as platform link. No dedicated grid computing system is required as volunteers offer their unused computing power. One or more hardware dependent client applications are developed targeting for example x86 CPUs with different operating systems or even GPU (BOINC and nVidia, folding@home and AMD GPU clients). The application client is then offered to volunteers who install it on their hardware host.

When the application is running it connects to a server and receives a part of the global simulation to execute. These parts are usually called compound. Data contained in the compound are then treated locally. When this part is done the application sends back results and asks for a new compound. Application compounds have no data dependencies with others and can therefore be computed without time or resource constraints.

As no infrastructure is required, potential computing power is then only limited to the willingness of Internet users. In April 2009 the folding@home project exceed 8

Pflops using CPUs, powerful GPUs and Playstation 3 (IBM cell processor) unexploited resources. In the same time the entire BOINC averages get over 1.4 Pflops.

In the section shows that distributed computing may be used as a scalable solution but they are far from the environmental integration that pervasive platforms are intended to reach. In addition, distributed application description is complex as thread programming should be use at cluster level and mixed with some message passing programming at the platform level to use the available computing power.

2.3 Toward Ubiquitous Computing

The emerging trend in computing relies on the integration of computing systems in the environment this is referred to as Ubiquitous Computing. In this context, the prominence of the computing performance aspect is replaced by environmental integration aspect; platforms are therefore qualified of pervasive or ubiquitous. Figure 2.11 proposes Wireless Sensor Networkss (WSNs) platforms as early Pervasive platforms. WSNs are referred to as Sensor Networks (SNs) in this document. Compared to classical distributed platforms, SNs and obviously pervasive platforms are no more intended to offer more computing performance but to observe, manage and interact with their environment.

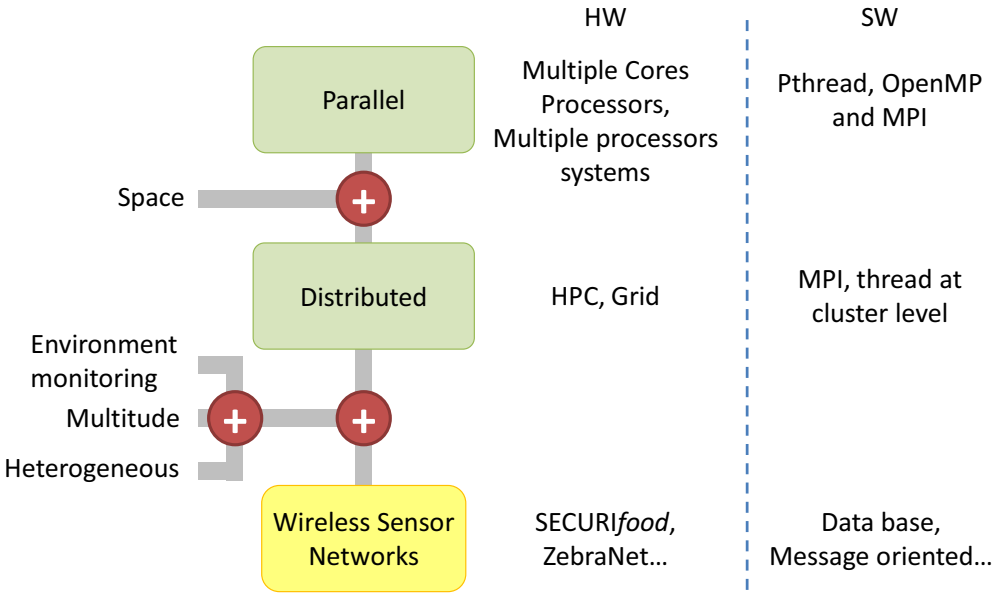

Figure 2.11: *Sensor Network solution*

This allows avoiding modeling the environment that is required in some heavy scientific simulations. Obviously, this also allows observing natural and potentially large scale behaviors. As a consequence large scale platforms may be required. Therefore pervasive platforms are composed with a multitude of computing nodes that are able to monitor physical parameters in their direct surrounding environment. In the case of SN these environmental interactions are passive.

As in the case of grids, SN may involve heterogeneous hardware modules. Heterogeneity becomes a required characteristic if the different nodes of a platform have different computing resources. Additionally, these nodes may be intended to monitor different parameters and compute data differently depending on their tasks and with regards to their computing resources. As a consequence, key factors in SN design are mainly dependent on the kind of application the platform is built for [76]. As an example the ZebraNet platform dedicated to wildlife tracking is intended to be robust, lightweight, energy-saving and only devoted to store animal position, move, and some biological signals with only little communications infrastructure [51], [94]. Other systems like the tsunami detection NOAA DART platform are able to communicate vital information to their base station via satellite [43]. Such systems are subject to physical constraints but should overcome them. Their HW and SW are therefore designed accordingly with these constraints.

The programming of such platform is critical as it must be adapted to the environment constraints in addition to wired or wireless grids programming constraints. As an example in resources constrained SN it is difficult to achieve a global time synchronization that can be useful if not required for a given application. In classical computing grids a global time reference can be easily established thanks to Internet or using message-based synchronization, this is no more true in the SN platforms context due to power and computing cost constraints. Therefore, efforts have been done on the SN programming.

In the following sections we present some state of the art motes hardware and their applications to give an overview of the diversity of available solutions. More references and examples can be found in the PhD thesis of Jason Lester Hill on *"System architecture for wireless sensor networks"* [47]. The next section then focuses on the different programming methods used in the SN context. Finally, based on these observations Section 2.3.3 presents some challenges inherent with pervasive computing platforms we propose to tackle.

2.3.1 Sensor Networks examples

Each of the three following sections presents an example of SN that exhibits these characteristics.

2.3.1.1 SECURI*food* for cold chain management

The SECURI*food* platform is a complete cold chain management system that is intended to early detect inadequate food storage conditions with respect to temperature [77]. This system then allows reducing the risk and cost to remove the goods from sale stalls. Such systems have to be reliable, with ultra-low power and cost hardware.

This complete system should fit with the logistic requirements of the retail groups (one producer with several distributors with several shops) where data are collected on every site and information are then centralized using Internet. This hierarchical grid uses several Wireless Sensor Networks connected through a virtual wired grid thanks to Internet.

The platform front-end is composed with thousands of Sensor Units (SUs) based on an I2C temperature sensor, a couple of XEMICS ultra-low power microcontrollers, radio chips and 512 bytes of data memory. The microcontroller is responsible of data conditioning before transmission. The standard program size is 8kB with more than half of the program dedicated to the communication protocol. The radio is able to send packet at a rate up to 64kbps. The radio protocol packet size is 72 bytes and data recovery is made every 15 minutes. The SU global power consumption is around 3μA. Sensor Unit lifetime with a 250mAh button cell is then up to 8 years.

Data from SUs are recovered by Relay Unitss (RUs) covering an area of 1000 m^2. Multiple hop communications allow using several RUs ensuring a scalable connectivity at the local level.

One Access Box (AB) per site is mandatory to provide a gateway between the local gathered data and online centralized applications. Applications are therefore based on database and take place outside of the sensor network for example on a client station that checks for inadequate storage conditions in a given place and get advices if such condition are encountered.

As a conclusion this platform uses a low cost front-end with information relays to gather data and feed them into a database. Computations rely on the platform back-end allowing using low cost and low power hardware saving energy at the node level. From this point of view this platform cannot be considered as a pervasive platform. A more powerful hardware should be used to allow application to really run into the sensor network and achieve a pervasive platform. SECURI*food* uses a hierarchical architecture with different hardware for every hierarchy level while large scale pervasive platforms may take advantage of the peer-to-peer inherent scalability. Moreover data are gathered during goods storage and therefore no dynamic topology support is needed. The passive environment monitoring is the essence of SECURI*food* therefore no active control of the warehouse and sale stalls temperature is provided.

2.3.1.2 Zebranet and Wildlife tracking

The ZebraNet platform is made of small and low power motes. The application principle is to install "collars motes" on individuals of the tracked zebra population. Motes are collecting data concerning their host such as heart rate, temperature, and activity. This information is logged with redundancy in the network and transmitted to biologists. The zebra activity is evaluated thanks to Global Positioning System (GPS) and individual speed computation creating daily migration patterns of the population. A GPS system is then embedded on every module. To save power this device is activated every 8 minutes to get a GPS fix. A timeout of 1 minute is set to save power in case the fix cannot be obtained.

This system is therefore a typical example of Wireless Sensor Network. The embedded hardware is based on an 8MHz microcontroller, a 900MHz XStream radio, a u-blox MS1E GPS and 4Mb of off chip memory for long term storage. The mote power supply is achieved using one rechargeable battery and 14 solar cells (7mA at 5V). It is able to satisfy the mote power consumption for the one year experiment duration. Additionally XYZ orientation and accelerations sensors can be included to provide more information on zebra behavior.

Data are transmitted through the network when a zebra comes into another individual radio range progressively flooding the network with its local and already collected information. Therefore the manned mobile base station only has to come in range of one individual to gather the population data.

This platform is dedicated to collect data and support only minimal computations such as GPS fix reading and individual speed, pairwise communications and information flooding (redundancy filtering). The hardware is suited to improve application lifetime as well as data precision and data recovery. Pervasive applications, as proposed in Chapter 3, have more computing requirements and consequently need a more powerful or additional hardware. The next example follows this later option.

2.3.1.3 Modified MICA2 and Sniper localization

Based on modified MICA2 motes [48], the Wireless Sensor Network proposed in [82] is used to locate snipers as well as the trajectory of bullets within seconds, after the shot, providing essential data for law enforcement. To achieve these objectives acoustic sensors are used to recognize and measure the muzzle blast and the bullet shock waves. The localization accuracy is about one meter with latency of less than two seconds if the number of node is important enough. As an example an experiment using 8 nodes provides an error of 8 m when 32 nodes reduce localization error to 2 m and 56 nodes to 1 m.

The application accuracy also relies on the precision of global node time synchronization. In [82] authors are using the Flooding Time Synchronization Protocol to obtain a synchronization variance of 1.4μs. Another solution could have been to use Global Positioning System (GPS) timing chips such as u-blox LEA-5T that are able to provide global synchronization with an accuracy of $\approx$15 nanoseconds (using the GPS atomic-clock time). Results presented in [82] show that a decrease of one order of magnitude of the time variance results in doubled localization accuracy. The GPS solution provides a 100 times more precise synchronization that can be used to enhance the localization precision with fewer motes while freeing the communication bandwidth for data transfers instead of using it for time synchronization. In this case counterparts should be the use of an additional chip and its inherent power consumption.

This kind of application is demanding a consequent computing power that the basic MICA2 mote, based on ATMEL ATmega128L microcontroller, is not able to cope with. Therefore a multi-purpose acoustic board has been developed as the acquisition board. It is based on three independent acoustic channels each including a microphone, a configurable amplifier and a 1MHz Analog to Digital Converter (ADC). To process

incoming information the platform motes also embed a Spartan II FPGA that is configured to classify sound as muzzle blasts, bullet shock waves or none of the above. A data consistency function is used to allow a sniper localization and bullet trajectory convergence at the platform level.

This application requires ensuring a quality of service at anytime with real time constraints, without any power saving objectives. Therefore hardware requirements are more important than in previous cases making this platform closer to pervasive platforms.

The software side of the three presented cases also greatly relies on the intended applications. In addition, the way a sensor network platform is programmed and its inherent capabilities are based on the low level software supporting the application also called middleware (MW) . In the next section we present some of the middleware that are used to develop wireless sensor network applications. This overview of the existing frameworks is necessary to make appropriate choices in our pervasive platform framework design.

2.3.2 Programming Sensor Networks

As for distributed platform, SNs have no shared resources. As a consequence MP is an obvious programming paradigm for such platforms. Nevertheless specific constraints of SNs require designing adapted programming frameworks. The following sections present some examples of frameworks that are used to program SNs.

2.3.2.1 Message oriented frameworks

Message-oriented middleware uses the publish/subscribe mechanism to allow reliable message exchanged between nodes. Asynchronous communications are naturally supported by this kind of middleware, allowing the sender and the receiver to be almost independent. Therefore this approach is quite suitable in pervasive environments such as wireless sensor networks, where most applications are event based.

Mires Middleware [83] has been designed for fixed topology wireless SN and provides a message-oriented framework. The publish/subscribe service communications between senders and receivers are established using a multi-hop routing algorithm. Three phases are required for communications in Mires:

1. Senders publish their data.
2. Data are transmitted over the network thanks to a flooding algorithm.
3. User applications are then able to subscribe to desired data (also called topics).

During the application lifetime only subscribed topics are transmitted. Mires services regularly update these topics creating events to be used by the application.

The restriction to fixed SN makes Mires unsuitable for mobile sensor network where network topology may change.

2.3.2.2 Database frameworks

In this SN programming scheme the whole network is considered as a virtual database system. Nodes update their data without any global time synchronization. Data of interest are queried to the network by the user through an interface. Major issues of this approach are: the spatio-temporal approximation of provided results, and the lack of real-time applications support. As an example the Cougar middleware implements management operations in the form of SQL-like language that allows the use of algebra operator for the sensor signal processing [50]. Using Abstract Data Type (ADT), Cougar represents all sensors of the same type in one data.

Such solutions are suitable for really constrained resources SN running data gathering applications with no critical tasks or real-time tasks objectives. The SECURI*food* example described in section 2.3.1.1 is a typical database based approach.

2.3.2.3 Application Driven frameworks

In this approach applications are closely connected to the network protocol stack. Thanks to this connection, applications dictate network operations management, providing Quality of Service (QoS). The programmers are then responsible to tune the network depending on the application requirements. However, the tight coupling with applications might result in applications dedicated middleware.

As an example the Middleware Linking Applications and Networks (MILAN) provides support to build applications that actively affect the entire network [44]. This

Middleware is based on primitives that cope with application requirements by selecting a set of sensors and network resources. The applications mapping is decided using SN information like: applications respective time dependent QoS requirements, applications priority or network available resources. Node communications are fundamental in this centralized management middleware.

Application performance but also results are therefore dependent on network state and topology. Changes in the network topology are then critical as applications mapping is centralized. This restricts the use of such middleware to static platforms with a limited number of nodes.

2.3.2.4 Agent oriented frameworks

Agent-oriented (or so called modular) middleware approaches are based on two principles:

1. Applications are designed as modular as possible to allow a high flexibility in application mapping and application code transmission.
2. Modules are injected in the network under the form of mobile codes.

This feature eases code implementations and migrations. Additionally to broadcast elementary SW modules, that update a given functionality, consumes considerably less energy than transmitting an entire application.

Update process also benefits from this. The Impala framework provides mechanisms that are able to support dynamic applications [57]. The agent based autonomic behavior increases the platform fault tolerance as well as network self-organization. Intended applications can take advantage of the flexible framework and are generally more complex. However, its native code and the agent overhead do not allow hardware heterogeneity and makes it unsuitable for devices with highly limited resources. The Zebranet SN example presented in section 2.3.1.2 is based on such middleware and greatly benefits for its fault tolerance and flexibility.

2.3.2.5 Virtual Machine frameworks

Virtual Machines (VM) based middleware exhibit high flexibility level as they associate the agent modularity with the system virtualization. Developers write their applications in separated small modules as for agent based middleware. The system injects and distributes the modules through the network using tailored algorithms, so that overall energy consumption and resource use are minimized. The VM then interprets the modules inducing an additional computing overhead compared with the classical modular approach. This is the main drawback of the hardware abstraction but it offers the possibility to span an application over heterogeneous platforms while only providing one application modular source code.

The Scalable Wireless Sensor Query Machine (SwissQM) programming framework described in [59] is a example of virtual machine based modeling framework. The SwissQM framework offers a platform independent programming abstraction that is oriented toward data acquisition and processing in the network. The virtualization provided by SwissQM ease the programming of SN running TinyOS by abstracting the NesC low level programming that is cumbersome and error-prone.

Other Virtual Machine (VM) based solutions may be experienced as current VM such as sun JVM propose Just In Time compilation (JIT) of interpreted bytecode to native code. This property reduces the interpretation cost for frequently used functions therefore reducing the global application overhead.

This approach is suitable for heterogeneous platforms such as Ubiquitous Computing systems as demonstrated in Chapter 1. Our objective is to provide a flexible framework for pervasive computing platforms; therefore the proposed middleware will be based on this kind of approach that offers the necessary flexibility and hardware independence.

2.3.3 Pervasive platform Challenges

Ubiquitous computing platforms are intended to interact in an active manner with their environment. Additional constraints, due to this activity therefore require improving nodes autonomy and adaptability to their surrounding conditions. Obstacle avoidance,

node failure, critical battery level reached and connection loss are a set of examples of possible problems that can prevent applications to end in a proper manner. As presented in Figure 2.12, wireless sensor networks endowed with these features may be able to offer the researched ubiquity and therefore design a pervasive platform.

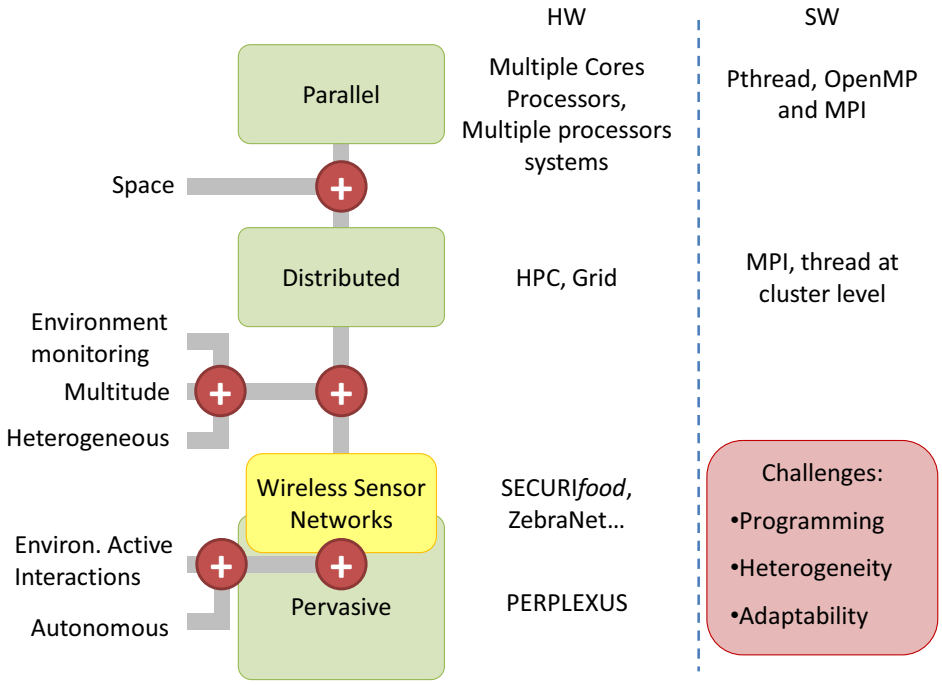

Figure 2.12: *Pervasive Computing challenges*

Some challenges in the platform programming and management emerge from these requirements:

- How to **program** such distributed and autonomous platforms to use the available ubiquity?
- How to manage and take advantage of the platform nodes **heterogeneity**?
- How to provide the required **adaptability**?

This thesis is proposing an approach to tackle these challenges. The following section introduces two fundamental principles, used to precise the general frame of the proposed contributions. In a third part the case study platform used to validate the approach is presented.

2.4 Pervasive platform programming proposal

The bibliographic study on SNs programming highlighted the AOP approach as a good candidate to program distributed platforms. As presented in Section 2.3.2 this solution is particularly efficient for platforms such as SNs which require adaptability while proposing a high level programmability on heterogeneous platforms. The proposed approach is therefore to use the AOP for the design of our pervasive framework.

Other concepts for pervasive framework have been proposed in the literature. In [63] authors propose a classification of ubiquitous computing frameworks. Referenced systems are mainly based on low level network protocols that support services for applications. For example Universal Plug and Play (UPnP) [91], the GRID [40] and MediaBroker [64]. These concepts share a major drawback; they have been conceived with wired or fixed wireless networks, this does not cope with the Ubiquitous platforms mobility. They are based on service offering with dedicated communications interfaces but do not propose a global programming scheme as AOP.

Nevertheless if AOP flexibility eases the platform programming this solution is not adaptive enough to ensure the platform autonomy. Researches using bio-inspired features proved that it is an interesting source of adaptability that can be tuned for various application fields [31, 68, 80]. We therefore choose to investigate this kind of solution to improve the platform adaptability.

The final proposal is to merge advantages of AOP and bio-inspiration inside a framework that proposes:

- An accessible programming methodology.
- A software executable onto heterogeneous hardware platforms.
- A high adaptability to the environment.

2.4.1 Agent Oriented Programming

Agent-Oriented Programming (AOP) derives from the initial theory of agent orientation which was first proposed by Yoav Shoham [81, 16]. This programming format infers Object-Oriented Programming (OOP) by endowing objects with additional

characteristics; they are viewed as entities which exhibit behaviors, capabilities and are entitled to take decisions. Agent-orientation was initially defined for promoting a social view of computing and finds natural applications in areas such as artificial intelligence [74] or social behaviors modeling.

An AOP computation consists in making Agents interact with each other through messages of different natures: they may be informing, requesting, offering, accepting, and rejecting requests, services or any other type of information. AOP furthermore fixes constraints on the parameters defining the state of the Agent (beliefs, commitments and choices) and constraints on the type of messages exchanged between Agents.

These constraints actually define the Agent-oriented computational system which is then viewed as a set of communicating and cooperating modules that exhibit a certain degree of autonomy. Agents are intended to offer and request services and still continue to operate if some of the requested functionnalities are not available; they are independent. Therefore AOP is naturally well suited for parallel, distributed or pervasive platforms as agents can be distributed and even migrate over the platform. These characteristics naturally geared our proposed modeling framework toward AOP which fits perfectly the characteristics and objectives of the test platform. Moreover the high inherent level programming allows defining applications within a large spectrum like collective robotic, culture dissemination or brain development simulation.

Thanks to the agent concept we can describe an application as the set of interactions between parts that are dedicated to specific tasks. Therefore it is possible to describe agents responsible of the platform management and adaptability.

2.4.2 Adaptability through Bio-inpiration

Ubiquitous platforms are intended to be closely linked to their environment and interact with it. Bio-inspired systems have proved their capability to adapt themselves to unknown situations [85, 87]. In addition platforms endowed with bio-inspired features may provide a unique support for the simulation of large-scale and bio-inspired systems.

The three main axis of life introduced in the POE model [17] (c.f. Section1.4) are:

- **Phylogenetic (P)** 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. This axis is therefore software implemented with processes running on SW.

- **Ontogenetic (O)** 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 qualified of ontogenetic. Some electronics devices like Field Programmable Gate Array (FPGA) are able to recover from errors using their reconfiguration capability [87].

- **Epigenetic (E)** refers to features that are not related to the underlying Deoxyribonucleic Acid (DNA) sequence of an organism. Learning as of performed by neural networks is a process which scope remains limited to an individual lifetime and therefore is Epigenetic. Therefore learning applications such as neural network provide epigenetic capabilities to computing systems.

An important drawback of the bio-inspiration integration is its important computing cost. To avoid this, bio-inspired features such as ontogenetic and epigenetic processes may rely on a bio-inspired device. The platform of the Perplexus project offers such hardware feature in its pervasive platform. Therefore it is an interesting solution to test and validate our pervasive computing framework.

2.4.3 Bio-inspiration and Agent-Orientation in the literature

Pervasive platform may be made of high-end nodes compared with classical sensor networks motes. These nodes may integrate dedicated hardware to accelerate heavy features such as bio-inspired applications. Such platforms are intended to run a large set of applications from robotic to neuro-sciences or social behavior modeling. Therefore it requires a reliable, flexible and accessible programming framework. As viewed in Section 2.4 agent oriented platforms are efficient solutions for this kind of platform. Some investigations have been conducted about using bio-inspired mechanisms in multiple agent computing frameworks.

- In [109] an agent deployment policy based on cell transforming and crawling locomotion is proposed. This work is implemented with Java on a wired network

of PC therefore it offers neither mobile nodes nor environmental interactions. However this work proposes an interesting collection of agent migration policies based on Ontogenetic observations.

- In [112] authors propose a task scheduler based on swarm division of labor observations. This scheduling is then used in a multiple agent programming environment on a distributed system. The objectives are to increase application throughput while decreasing the average flow time. This is done by optimizing dynamically the use of platform resources. In this framework, tasks are proposed to a node level authority that computes its *intention* to achieve the task depending on several parameters. For example: the task heaviness, node current workload and node ability to achieve the task are used to compute the *intention*. Therefore on heterogeneous platforms the most efficient hardware can be used for task they are specialized for if they estimate that they have time to achieve these tasks. The three major problems are:

  - The centralized task bank where applications are proposed make the system centralized.
  - The unpredictability of application mapping that prevents this system to be used for tasks where environmental interactions are spatially significant.
  - Finally the tasks have to be profiled before their implementation to provide a knowledge base of performed operations. This task profile is required for comparison with the host operation efficiency. This requires to know architecture capabilities of the different nodes and profiling all tasks.

- In [111] bio-inspired agents for priority routing are proposed to solve complex routing problems. The proposal is to use the ant routing enhanced with agent adaptability, multi-path routing and priority routing. Routing agents are moving in the network and color links with different types and concentrations of pheromones like ants tag their way. Additionally a genetic algorithm has been used to tune algorithm parameters resulting in a specification that allows agent searching better routes even if a very good one is already used. Drawbacks of the proposed routing algorithm are that it requires agent mobility, and the storage of pheromones concentrations, both points are lowering available bandwidth and memory. Moreover such routing algorithms are well suited for static network topology but the pheromone life time makes it unreliable for dynamically changing topologies such as those we can find in pervasive platforms.

Nevertheless to the best of our knowledge, in the literature, there is no pervasive platforms middlewares that provide support for low level bio-inspired mechanisms (i.e. POE). Given that such services may help user to define adaptive applications on pervasive platforms, our framework will propose this support.

2.5 Case study platform

As introduced in Section 1.5 the Perplexus platform is intended to scale with the problem complexity. A modular structure is required to allow this hardware scalability. This section presents the Ubiquitous Modules (Ubidules) that compose the Perplexus platform.

At the heart of the Ubidule, the Colibri module from Toradex is a powerful and low power computer based on the PXA270 XScale processor (implementation of the ARM V5 instruction set)[89]. Its reasonable power consumption (800 mW) and weights (12.2 g) make it particularly interesting for mobile embedded applications. The Colibri board is running "openembedded" Linux operating system that offers the required flexibility and low-level accessibility.

As suggested in Figure 2.13, the Perplexus platform is intrinsically ubiquitous because of its distributed structure made of numerous environment-embedded processing nodes. Communications among those nodes are also decentralized and take place in a peer-to-peer fashion using the Wifi technology.

Each node is then equipped with sensors and actuators that make it able to interact actively with its environment. This opens the way to an autonomous and wireless distributed platform.

2.5.1 UBIquitous CHIP: Ubichip

One of the goals of the Perplexus project is to develop a bio-inspired chip called the Ubichip [65, 66, 88]. An overview of the Ubichip that can be used with two different modes is provided in Figure 2.14. This chip has been developed at Haute école d'ingénierie et de gestion du canton de Vaud (HEIG-VD) (Switzerland) and universitat

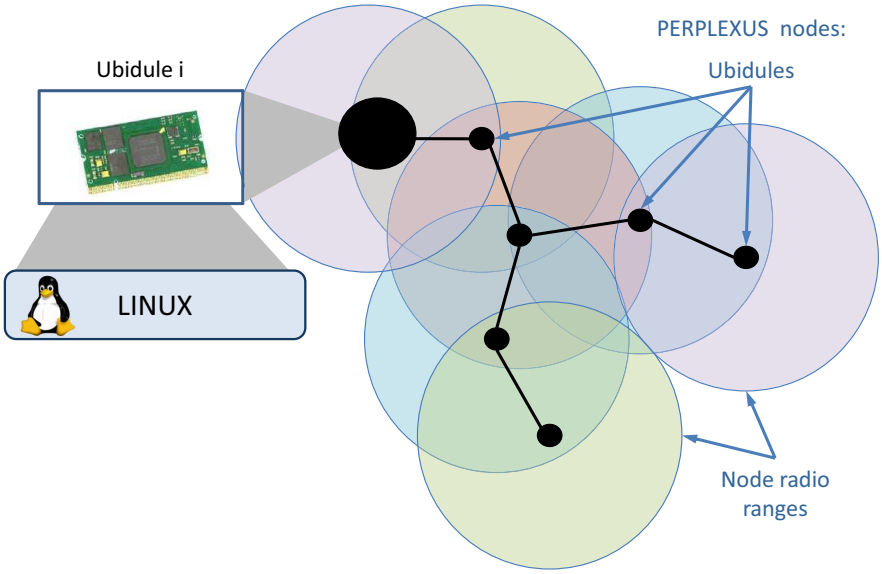

Figure 2.13: *Early Perplexus platform overview*

politècnica de catalunya (UPC) (spain). Validation, design and tests have been done at Technical University of Lodz (TUL) (Poland). At its lower-level, the Ubichip is made of a homogeneous array of elementary reconfigurable elements called Ubicells [65] and a system manager that includes Self Reconfiguration units (SR), Dynamic Routing units (DR) [88], a memory controller and an instruction sequencer.

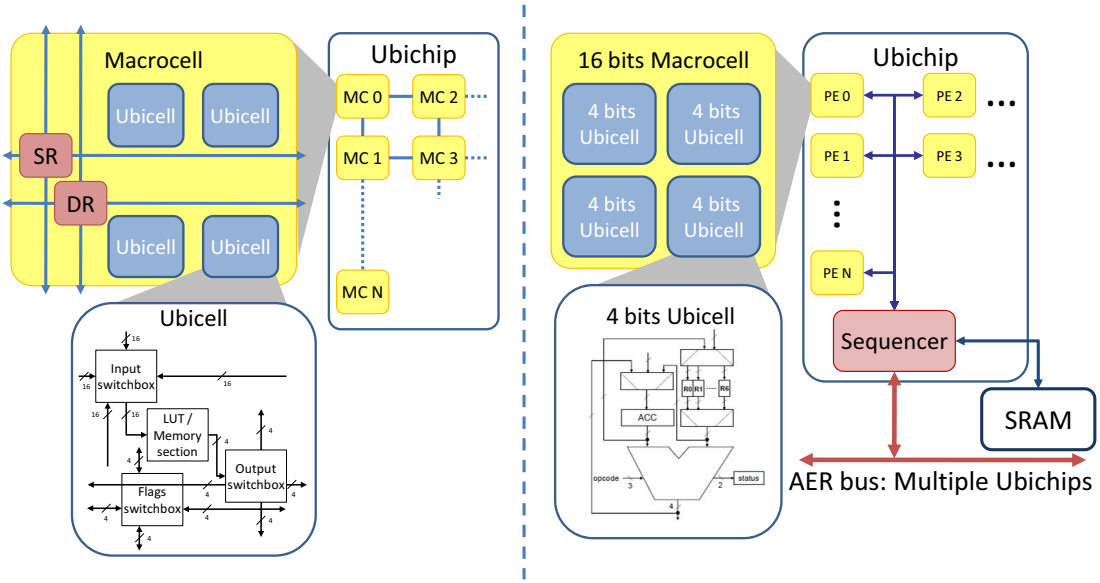

Figure 2.14: *Ubichip overview*

Each Ubicell can be configured in Lookup Table (LUT) mode (left side of Figure 2.14) or in Arithmetic and Logical Unit (ALU) mode (right side of Figure 2.14). In

Lookup Table (LUT) mode (also called Native mode), a Ubicell provides the functionality of four independent 4-input combinations functions, it allows to emulate several typical logic functions like wide decoder, counter, state machines or shift registers. Using several Ubicells, advanced functionalities such as neurons can be implemented then supporting Epigenetic processes. The Macrocell is defined as the gathering of 4 Ubicells, a SR and a DR. These units implement ontogenetic processes by allowing the chip to self replicate functionality in a free part of the Ubicell array.

When the Ubicell array is configured in ALU mode, a Macrocell becomes a basic datapath with an accumulator, seven general purpose registers and a 16-bit ALU (made with the four 4-bit Ubicell ALUs). An additional *shadowed* register bank allows increasing the number of registers to 14 and the number of accumulators to two. In this mode, each Ubicell is able to execute programs code with a reduced instruction set. Instructions like arithmetic and logical operations, move, load and store instructions, as well as swapping values between basic and shadowed register banks. The main feature of this architecture is that each Ubicell runs in parallel the same program loaded from an external program memory and controlled by a single dedicated sequencer. As a consequence, this operating mode is called the Single Instruction Multiple Data (SIMD) mode. The second contribution proposed in Chapter 4 on this document relies on this specific SIMD mode.

Finally at the routing layer, some communication resources allow connecting a Ubicell with eight of its neighbors. An automated routing algorithm has been included in the Ubichip to allow further automatic Ubicell interconnections.

Figure 2.15 presents the final Ubidules structure. Such modules are equipped with one or more Ubichips. The hardware of the Perplexus platform is then completed by adding application specific sensors and robotic actuators. Apendix A presents an overview of the way Ubidules are embedded onto robots called Tankbots.

As the Perplexus project progresses, a final Ubidule robotic skeleton based on the MarXbot has been proposed [58]. This robot embeds a large set of sensors, a "Treel" based motion system that takes advantages of tracks and wheels and a low level ASEBA controller [62]. Due to conception and fabrication delays, experiments on this skeleton were not possible at the time of this writing.

The resulting mobile and bio-inspired platform regroups all the described features that pervasive platforms are supposed to gather. As a consequence it offers an interesting substrate to test our pervasive platform programming framework and its capability to solve presented challenges.

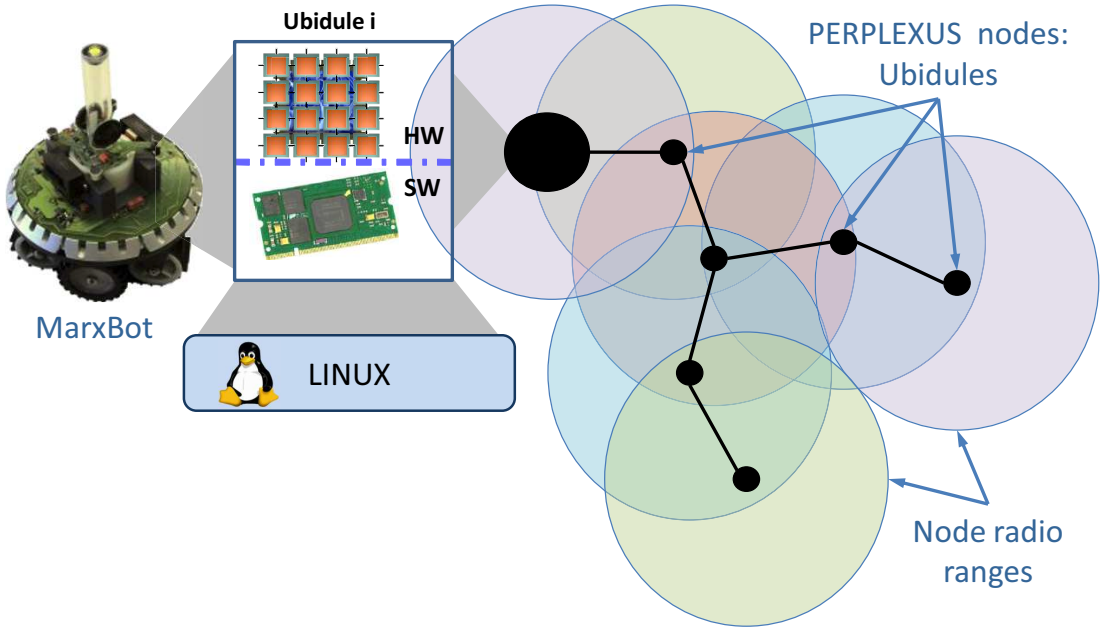

Figure 2.15: *Perplexus mobile bio-inspired overview*

2.6 Conclusion

This chapter provides information about the background knowledge this thesis relies on. The Ubiquitous Computing trend seems to be the future of computing systems and pervasive platforms are about to emerge. To pay close attention on the programming of such platforms is therefore required.

Based on the Sensor Network experience, key factor have been identified to allow the design of a reliable programming environment for pervasive platforms. This framework uses the AOP flexibility and scalability. Inter-agent communications allow distributing applications transparently over wide spread platforms. We believe that the added bio-inspired features improve the framework adaptability and obviously make its hosting platforms adaptive.

This bibliographic study helped to identify three the challenges that are adaptability, programmability and heterogeneity. The proposed contributions developed in the following chapter deals with these challenges to provide a programming framework for pervasive platforms such as the Perplexus platform. Chapter 3 mainly focuses on the programmability and adaptability issues. It presents the proposed bio-inspired and agent-based programming framework. Chapter 4 focuses on programmability and

heterogeneity issues. It presents a generic methodology to program hardware accelerators of such heterogeneous platforms. This is done with respect to the hardware independent policy of the framework proposed in Chapter 3 with a minimal cost at the programming level.

Chapter 3

Bio-inspired Agent Framework

Contents

3.1 A bio-inspired framework 57
3.1.1 Agent-Oriented Programming endowed with bio-inspiration 57
3.1.2 BAF overview 59
3.2 The BAF implementation 65
3.2.1 Bio-inspired agents 65
3.2.2 Platform and service agents 67
3.3 Applications 86
3.3.1 Evolving simple robot controllers 86
3.3.2 Online collaborative learning 88
3.3.3 Evolving a population of learning individuals 91
3.4 Conclusion on the BAF environment 95

CHAPTER 3 ABSTRACT

In this chapter a Bio-inspired Agent Framework (BAF) is proposed as a solution to the programming of Ubiquitous computing platforms. Technical choices are discussed to explain the philosophy of the BAF. This agent framework endowed with bio-inspired features relies on an ad-hoc wireless mobile network. Based on specific agents, the BAF has been experimented on the Perplexus platform through three pervasive robotic applications using bio-mimetic features.

Divide et Impera
"Divide and conquer"
ROMAN SENAT POLITIC .

As exposed in Chapters 1 and 2, ubiquitous computing is the future trend of computer systems. Invasiveness of such systems will greatly rely on their autonomy, reliability and transparent *programmability* as they are intended to keep a low profile [90]. In other words, nodes have to be autonomous enough not to warn user every time an error occurs. *Adaptability* is therefore a critical point to manage in pervasive computing environments. In addition the scalability of the platform has to be ensured as the pervasiveness of the platform requires numerous and ubiquitous nodes.

Pervasive platforms are generally based on decentralized networks in order to provide such scalability. Nodes participating in those networks could be heterogeneous and have active interactions with their surroundings. As a consequence, the programming of these numerous nodes is fastidious and far from the pervasive platform intended invisibility [90]. AOP has been one of our first choices to manage this programmability issue. Flexible and natively communication-oriented AOP is well suited to run applications distributed over a network. The recently published book [113] gives an idea of how this choice is been experienced by other researchers.

The originality of this contribution therefore relies in the way the platform adaptability is managed. To slove this issue, we propose to include the POE bio-inspired features in every nodes of the platform. This allows nodes to adapt themselves to their environment and simulation scenarios. Finally, using this Bio-inspired Agent Framework (BAF), simulated individuals of complex systems will take advantage of the flexibility, reliability and environmental integration of the platform. This chapter describes our proposed BAF.

3.1 A bio-inspired framework

This section presents the philosophy of the proposed framework. Two fundamental principles have been presented with AOP and bio-inspiration.

We designed the BAF with portability in mind; even if tests have been conducted using the Perplexus platform. This allows to use the BAF with other platforms and even on heterogeneous platforms. The Java technology is among the most commonly used portable language as it is based on virtual machines. Additionally it is the base of numerous multiple agent platforms. As a consequence this decision suits the AOP choice. Numerous multiple agent platforms based on Java exist; they are as many candidates for the base of our framework. The AOP platform selection is discussed in Section 3.1.2.1.

The advantage of using agents is their ability to natively communicate through an established network, nevertheless, generally there is no native support for mobile (i.e. topology changing) networks in multiple agent platforms. As this kind of ad-hoc network is a corner stone of ubiquitous platforms, such support is mandatory in the proposed programming environment. In the literature Mobile Ad-hoc Network (MANET) protocols are the most appropriate solutions to solve both issues at the same time. Using multiple hop routing algorithms, MANETs allow inter-nodes communications in ad-hoc networks with full connectivity. The MANET protocol selection is discussed in Section 3.2.2.1.

3.1.1 Agent-Oriented Programming endowed with bio-inspiration

As introduced in the previous section, Java is generally used for AOP platforms development. This technology fits our portability and flexibility needs. In addition, the Java object-oriented language is accessible and the class organization helps to define platform, service or application agents. Platforms like JADE [29] [30], JXTA [42], FIPA OS [39], JAX [84] or Multi-Agents Developpement Kit (MADKIT) [61] are examples of software platforms that implement the AOP concept. One important choice, not to develop another complete AOP platform, has been to select the appropriate agent platform. The portability guideline helped the decision as some platforms were ported to several and even heterogeneous targets [33], [56] and [67].

When dealing with the heterogeneity of a platform HW and SW aspects have to be taken into account. In Chapter 1 ubiquitous platforms were partially presented as interoperating devices. Such devices may come from different vendors, using different HW and different SW; a standard is therefore required to ensure devices interoperability. The Foundation for Intelligent and Physical Agents (FIPA) defines a standard to unify multiple agent platforms therefore providing interoperability between different platforms. This allows FIPA compliant Agent Platform (AP) to inter-operate. Figure 3.1 presents the classical IEEE standard of FIPA framework.

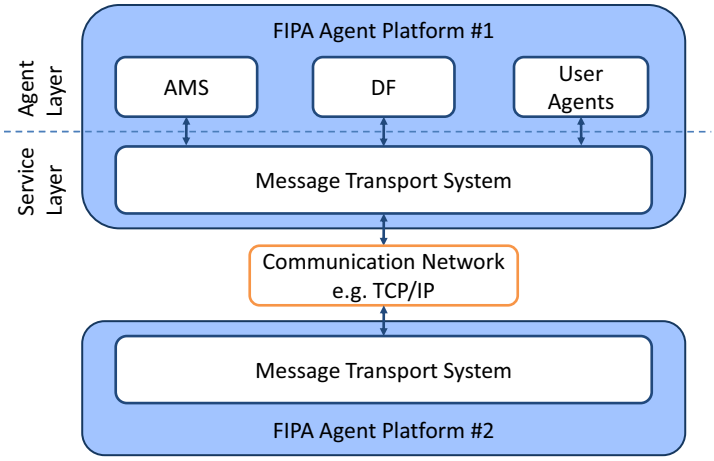

Figure 3.1: *FIPA classical framework*

This observation reduced the number of candidate and oriented our preliminary selection around 3 multiple agent platforms. Java Agent DEvelopment framework (JADE) is based on this FIPA standard. Juxtapose (JXTA) platform is frequently used and proposes interesting properties as exposed in Table 3.1. The Agent-Group-Role formalism is also frequently referred to in the literature [96, 107]. This formalism is built around the following principle: agents are members of groups and able to handle roles designed for these groups. This formalism is implemented in MADKIT. A qualitative evaluation of these platforms with respects to our requirements is presented in Table 3.1.

These qualitative tests in conjunction with our portability and interoperability guidelines oriented our choice toward the JADE platform. This platform is flexible and the high level description it offers help to describe complex agents. Based on this observation, it was then possible to think about integrating the bio-inspired POE features inside agents. Such agents would take care of managing the bio-inspired aspects inside each node of the platform. The following section presents this philosophy of our proposal in more details and gives an overview of the work done to design this BAF.

Criteria	JADE	JXTA	MADKIT
FIPA compliant	+	-	-
Manet management	-	+	-
GUI	+	-	+
User friendliness	+	-	+
Portable version	++	+	-
Comments	FIPA Standard. Modular. User friendly interface. J2ME version available.	Data exchanges based on XML. Low level management methods available (Peer, Service searches). Limited to a collection of functions.	Set of tools: programming and simulation interfaces. No J2ME compliant: not suitable for embedded devices. No formal standard

Table 3.1: *Qualitative comparison between selected platforms*

3.1.2 BAF overview

For the sake of clarity, the JADE platform has to be presented in order to show the offered possibilities and missing functionalities of this development framework. We then present the work done on this well known multiple agent platform in order to add the required functionalities of the BAF.

3.1.2.1 JADE

JADE is a platform for the development and run-time execution of peer-to-peer applications which are based on the agent oriented paradigm. It is a Java-based multi-agent development system that fulfills the FIPA specifications shown in Figure 3.1. A JADE AP as shown in Figure 3.2 contains at least one container named “Main-Container”. This “Main-Container” contains at least two mandatory agents, the Agent Management System (AMS) and the Directory Facilitator (DF) agent. The required inter-agent communications are managed thanks to the Agent Communication Channel (ACC).

- The AMS agent represents the authority in a JADE platform. It manages the life cycle of all agents of the platform; it can create, suspend or kill agents. It updates a list containing the name of all agents hosted on the platform, providing a "white pages" service. This list is made using the Agent IDentifier (AID) an object part of the JADE agent class that contains the agent address and its platform location. Both parameters are required to establish a communication with an agent. Finally, it controls the communication channels between agents.

- The DF agent records the services provided by all the agents of the platform, it acts as a "yellow page" service. At agent creation time or during runtime an agent can publish the services it is able to provide by registering the service in the DF database.

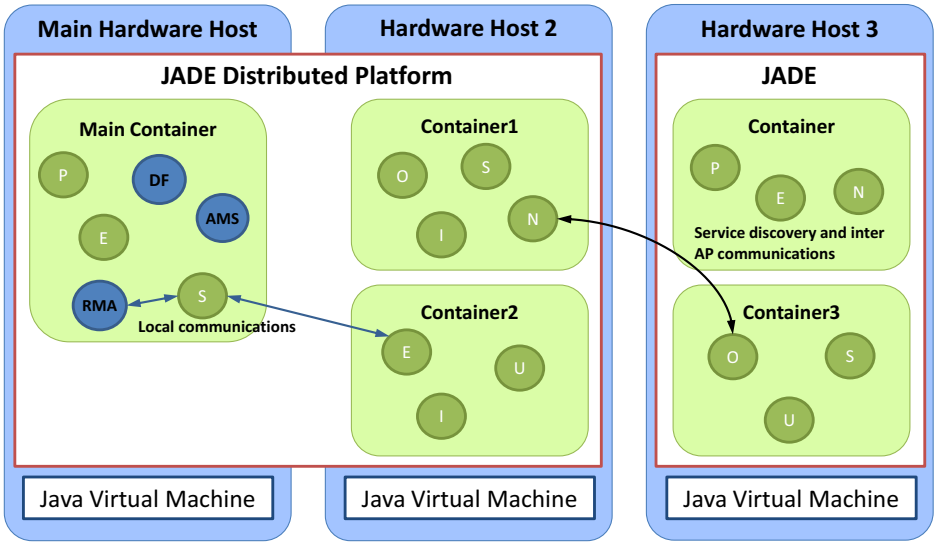

Figure 3.2: *JADE platform*

One optional agent may be started to provide a basic and platform centric GUI. The Remote Management Agent (RMA) allows controlling, launching or killing agents of its own AP. The possibility is also given to connect to other APs and get their agent list but there is no possibility to control them. Therefore this GUI loses its interest when several JADE platforms are running and cooperating as no unified control of the resulting platform is provided.

User agents are defined to execute the desired tasks, or to provide additional services. They are based on extending generic classes such as the *JADE Agent* as presented in Figure 3.3. Inside or outside of this Agent class one or more behaviors may

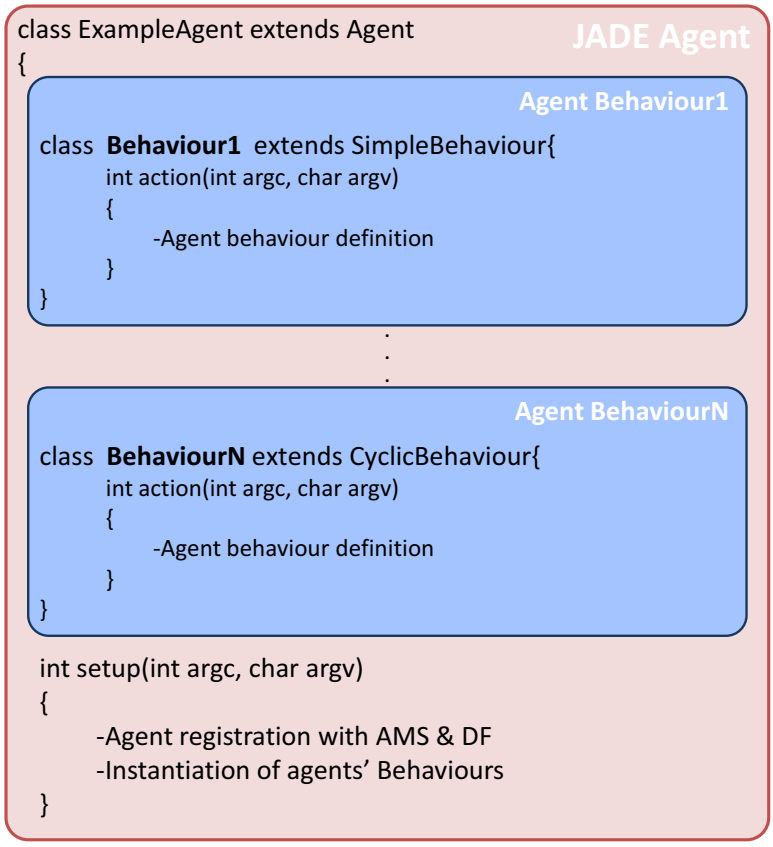

Figure 3.3: *Agent and Behaviours in JADE*

be defined extending generic behavior classes. The mandatory `action` function of these classes is used to define the actions to be performed when Behaviours are called. Finally, the `setup` function is used to define the lifecycle and registrations of the agent inside the platform. This function is run at the agent creation allowing it to be registered in the AMS, to register services to the DF and call the required Behaviours that define its functionalities. Obviously such functionalities may involve inter-agent communications.

Communications between JADE agents are made using the ACC service that is responsible of the intra-container, intra-platform and inter-platform message exchanges. In the case of extra-platform messages are exchanged using the Transmission Control Protocol (TCP) and thus require a reliable TCP/IP network support. This point is critical in pervasive platforms where nodes are potentially mobile.

3.1.2.2 JADE limitations

We found three restrictions within the classical JADE framework. The first limitation of JADE is the lack of adaptability at the network level. The framework actually relies on the system network routing therefore this routing must be adapted to the MANET routing scheme of pervasive platforms.

The second limitation is that JADE does not provide peer discover service as pointed out in Table 3.1. This limitation comes from the FIPA standard that does not have such service specification. Therefore this kind of service has to be implemented.

The third limitation relies at the GUI level. Even if the RMA offers a user friendly interface, the ability of this interface is clearly limited to the control of a single AP (i.e. a single node). A unified view of the nodes and centralized control of the platform are required to abstract the platform changing structure and to program the platform from a single device.

To solve these limitations as well as to provide the required bio-inspired features support in the framework, specific agents have been added to the JADE standard platform. Additionally some classes that inherit from JADE classes have been defined to cope with our ubiquitous and bio-inspired computing objectives. Such classes allow users to add their own application agents and therefore implement their BAF applications.

3.1.2.3 BAF: Agent organization

The proposed agents are presented in Figure 3.4 that gives a brief overview of the set of agents designed for the proposed framework hosted on a Perplexus Ubidule. These added agents can be classified in two categories, *Bio-inspired* agents and *Structure and services* agents.

1. *Bio-inspired* agents:
   - `Phylogenetic (P)` agent is in charge of phylogenetic processes. Applications may rely on the offered opportunity to simulate GAs in a distributed way. More details about the `P` agent are presented in Section 3.2.1.1.

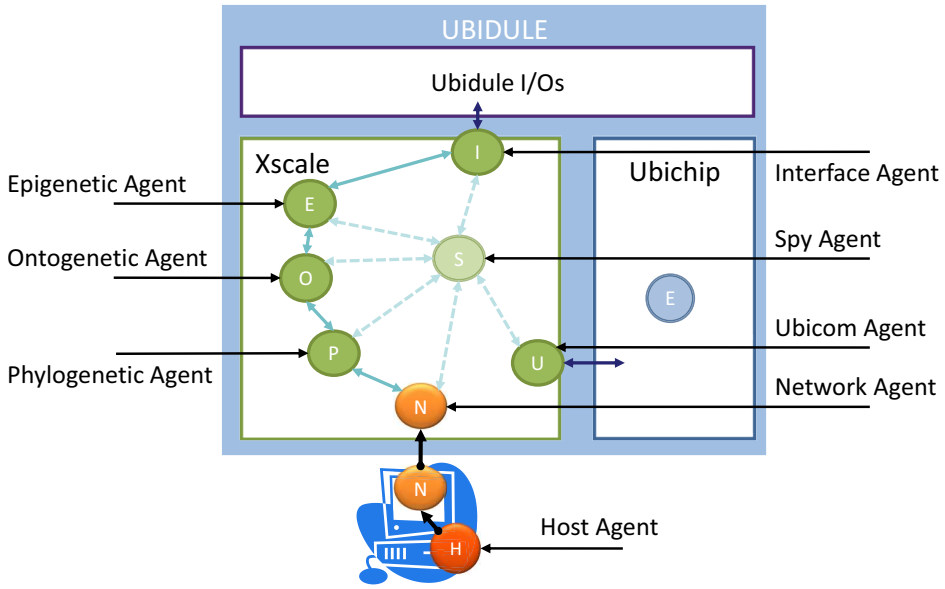

Figure 3.4: *Overview of the agents of the framework*

- `Ontogenetic (O)` agent implements the ontogenetic principles of biology. This agent provides ontogenetic services at a node level and may be used to instantiate agents that participate to the application. More details about the `O` agent are presented in Section 3.2.1.2.
- `Epigenetic (E)` agent offers the opportunity to add learning capabilities to the platform. Local `E` agents can be used to instantiate ANNs that endow applications with online learning capabilities. More details about the `E` agent are presented in Section 3.2.1.3. Figure 4 an `E` agent is represented in the Ubichip area, this reflects the opportunity presented in Chapter 4 to have parts of specific agents running on the Ubichip.

2. *Structure and services* agents:
   - `Network agent (N)` is the local responsible of the network link. It allows each node to interact with the rest of the platform. Information about this `N` and the underlying protocols are presented in Section 3.2.2.1.
   - `Host agent (H)` provides a graphical user interface that offers a complete control on the platform. The host agent is running on the user workstation and connects to the platform nodes through the Network agent. This agent is described in section 3.2.2.2.
   - `Spy agent (S)` allows users to gather nodes activity trace and therefore monitor and debug running applications. More details are provided in Section 3.2.2.3.

- `Interface agent (I)` is responsible of the HW/SW interfaces with sensors and actuators available on the node. For example TankBots are controlled thanks to this agent. More details are provided in section 3.2.2.4.
- `Ubicom agent (U)` is closely linked with the contribution presented in Chapter 4 as it is use to allow interactions between the SW side of the node and hardware accelerator such as the Perplexus Ubichip presented in Section 2.5.1. More details are provided in section 3.2.2.4.

As depicted in Figure 3.4 `P`, `O` and `E` agents are at the center of the Framework providing bio-inspired services for applications or bio-inspired applications by themselves as presented in Section 3.3. The common usage of the BAF is based on the following organization scheme.

The `P` agent receives or creates the genome simulated on the node, it is able to request the `O` to use the genome characteristics to implement agents with precise parameters. In the example of Figure 3.4, `O` may be used to create a given ANN by instantiating the `E` agent with parameters such as neuron model, learning model or network structure.

Service agents are more peripheral agents that are used as support for bio-inspired agents and application agents. The `N` agent provides a network interface to the node agents. As an example, it is used by the `P` agent during the inter-generation phase to exchanges fitness and genomes. The `H` agent is dedicated to interface the user with the platform and is able to control it through the `N` agent.

Additional agents may be instantiated to provide more services, as an example, the `S` agent may be called within every agent to report bugs that may occur on the node. In the case environmental interactions are required in the application the `I` agent provides an interface between the software and the sensors or actuators of the node. For example, the `E` agent may get its inputs from sensors and may provide commands for the actuators. Finally the `U` agent offer the opportunity to interface the software with an accelerator if present on the node (as for the Perplexus Ubidules). This later agent greatly relies on the contribution presented in Chapter 4.

3.2 The BAF implementation

In this section, implementations details about the BAF design are presented. Firstly, bio-inspired agents are presented as the corner stone of the BAF. Secondly, platform and service agents are presented with details concerning the provided services and functionalities.

3.2.1 Bio-inspired agents

As explained previously three bio-inspired agents have been designed to provide support to the three principles of life. This section focuses on these Phylogenetic (P), Ontogenetic (O) and Epigenetic (E) agents.

3.2.1.1 Phylogenetic agent

To represent the phylogenetic aspect presented in section 2.4.2, we designed the Phylogenetic agent (`P`). This agent is responsible of the execution of distributed Genetic Algorithms: it creates local genomes, calculates the local fitness of the individual (the local Ubidule) and synchronizes this information with all other `P` agents of the platform. Additionaly, it is responsible for triggering the death (end of a generation) and birth of the embodied individual hosted on the node.

For flexibility reasons, in the BAF this agent is provided as a software skeleton. Therefore, users have to redefine and adapt it to its own Genetic Algorithm. As it is part of the framework and uses the BAF classes this skeleton is fully compatible with the environment. For example `P` may be used in conjunction with `O` and `E` as follow:

- `P` gives information to `O` based on the local genome.
- `O` uses these information to trigger the creation of `E`.
- `E` runs the active application side and then gives feedback data to `P` that finally computes the fitness of the local genome.

Section 3.2.2.5 provides a more detailed description of this application example.

3.2.1.2 Ontogenetic agent

The ontogeny principle has also been included in the BAF within the O agent. O is responsible of ontogenetic activities in the framework at the node level. For example, this agent may be used to create new functionalities by instantiating agents. This improves the reliability of the platform as not available services can be instantiated on demand. If a service becomes unavailable, a request to the local O proposes to create the agent responsible of this service. The ontogenetic agent is then responsible to create the requested service as ontogenetic principles manage the individual growth and self healing capabilities.

Moreover, this simple agent is interesting to build composite bio-inspired applications. As an example the natural PO process that rules an individual growth by replication and differentiation of a cell that contains the individual genome may be implemented by coupling P and O agents. The Genetic algorithm may ask for multiple agent creation according to the genome of the simulated individual and then evaluate this agent configuration as the individual fitness.

Therefore the main characteristic of the ontogenetic agent is that it has the capability of creating other software agents on demand. The limitation of agent creation is associated to the existence of required class files on the O agent platform or on the host platform as they can be sent to the targeted node using the file transfer service.

3.2.1.3 Epigenetic agent

The third bio-inspired mechanism is the Epigenesis principle. The Epigenetic (E) agent has been designed to provide learning capabilities to the hosting platform. This kind of learning processes may be encountered in ANNs as presented in Section 2.4.2. Several Java ANN classes are provided to allow user to select the desired ANN type. Other ANN models may be added and instantiated in the E agent.

The inherent parallelism of ANNs makes it interesting to have our E agent to be able to be accelerated if a hardware parallel accelerator is available as presented in Figure 3.4. This is the case of Perplexus Ubidules with the SIMD mode of the Ubichip. We therefore provide the possibility to run this agent in both SW and hybrid HW/SW

execution modes that is presented in Section 3.2.2.2 and refers to the contribution presented in Chapter 4.

The goal with this agent as well as `P` and `O` agents is to provide the opportunity to run bio-inspired applications on the platform and improve the platform adaptability. This allows the platform to interact with and learn from its environment. Based on the FIPA standard description Figure 3.5 shows the BAF schematic. `P`, `O` and `E` agents are represented inside the user agents box as they may be modified to provide user specific functions such as desired GA or ANN types.

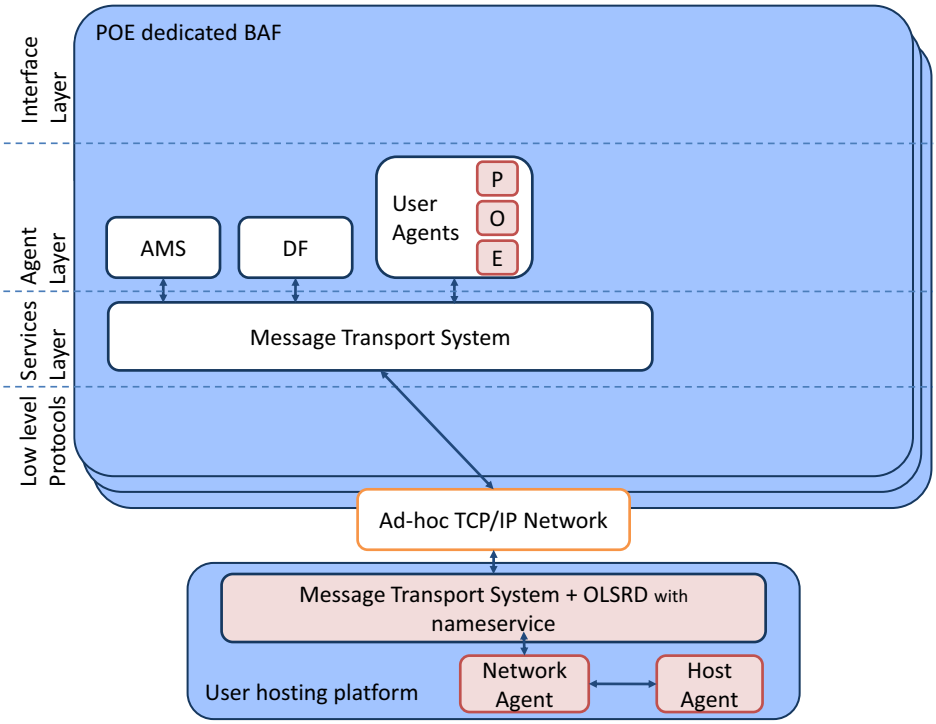

Figure 3.5: *Evolution on JADE framework for bio-inspiration support*

3.2.2 Platform and service agents

As presented before, bio-inspired agents are only one aspect of the proposed framework. The BAF does not rely on these agents to provide low level services to the platform. Services like network connection, peer search, service search or platform management and monitoring are required to make ubiquitous platform reliable and programmable. Dedicated agents have been designed to this purpose and are presented with details in this section.

3.2.2.1 Network support: Network agent

The network is a critical point of pervasive platforms. The platform greatly relies on the ability of numerous nodes to be able to communicate and to interact together. The mobile node context brings even more constraints and requires using dedicated routing algorithms that are adaptive enough to offer a complete connectivity in a topology changing network. The Network agent therefore relies on such protocol. This section presents the selected solution and its adaptation to the `N` agent specificities.

MANET protocols provide routing services comparable to regular wired networks without requiring any specific deployed infrastructure. These protocols abstract the physical network topology into a reliable logical network. Figure 3.6 depicts such a network where there are no direct connection (dashed arrow) between the Transmitter and the receiver which implies routing the messages through intermediate nodes (plain arrows).

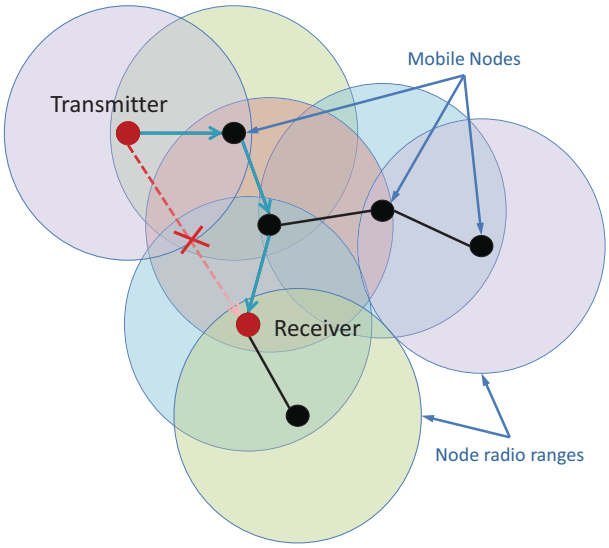

Figure 3.6: *Mobile Ad-hoc NETwork principle*

MANET implementations can mainly be grouped in two categories, reactive or proactive protocols.

Reactive Protocols. Reactive protocols are based on the on-demand routing principle meaning that a route is calculated upon transmission request only. This principle is known as the global flood search [108]. Protocols such as Ad-hoc On-demand Distance Vector (AODV) [108] [100] use this protocol and operate as follows in :

1. The source node broadcasts a Route Request (RREQ) message to find a destination node and waits for a response.
2. Each node receiving this RREQ message transfers it to all other in-range nodes excepted if it is the RREQ destination. Phase 2 is repeated until the destination is reached.
3. When a destination node receives several RREQ packets from a given source, it keeps information only from the first incoming packet. Others RREQ packets from the same source are discarded since they likely correspond to sub-optimal routes.
4. A Route Reply (RREP) is sent back to the source node using the backtracking route and specifying the route packets should take. Nodes routing tables are updated accordingly.

Reactive routing protocols present 2 main drawbacks: latency time because of the reactive, on-demand only scheme and traffic overhead because of the flood search. AODV and Dynamic Source Routing (DSR) [103] are two examples of reactive protocols. DSR is based on the source routing algorithm where relay nodes add their address to the route search packet to establish the route. Therefore only available routes can be backtracked and validated. But this principle limits the protocol scalability as message size depends on the number of hops and is constantly growing until the sink is found.

Proactive Protocols. Proactive protocols are table-driven; they actively and periodically update the routing tables by sending periodic messages. Thanks to this regular exchange of messages in the network, every node knows at any time the topology of the network. Therefore the route to reach a destination is always known for any application triggered communication.

Proactive protocols have two main drawbacks which are the constant communication overhead induced by the periodically exchanged messages and the workload induced by the route calculation. Consequently these drawbacks increase power consumption. Optimized Link State Routing (OLSR) [102], Fish Eye Routing (FSR)[105] or Landmark Routing protocol (LANMAR) [106] are some examples of proactive protocols.

OLSR and AODV protocols are among the most popular solutions in MANET routing algorithms, they both have open source and portable implementations. Therefore they have been chosen as candidates for the final selection.

OLSR and AODV comparison. Some comparative studies have already been conducted on proactive and reactive protocols. In [101] authors present performance figures for OLSR and AODV protocols. The results authors present have been obtained based on the Network Simulator (NS2) [99] that enables simulating with a reasonable accuracy the behavior of different ad-hoc routing protocols. Results reported from their work are presented in Annex B.

Table 3.2 summarizes the advantages and drawbacks of both AODV and OLSR protocols.

	AODV	OLSR
Advantages	No need of any known network topology. Standardized by IETF.	Low latency time in route establishment. Topology always known by all the nodes. Ported to many platforms (Windows, Linux, MACOS , $etc.$). Standardized by IETF.
Drawbacks	Important latency in route establishment. Poorly scalable: increasing communication control overhead.	Constant traffic overhead. Constant power consumption.

Table 3.2: *AODV OLSR comparative table*

For scalability reasons as well as low communication delay the OLSR protocol better suits our needs. Finally Optimized Link State Routing Daemon (OLSRD) [110] implementation of OLSR has been selected for the MANET management. This implementation proposes two interesting features that give OLSR a clearer advantage on AODV and other proactive protocols. The two following sections details these features.

OLSRD feature ETX link quality. OLSR periodically calculates routes using a shortest path algorithm. The OLSRD implementation provides an alternative metric for route calculation, named ETX link quality. This metric focuses on route reliability rather than delay; the daemon implementation actually takes both parameters into account which results in many possible traders.

The link quality is computed using the observed packet drop rate on the wireless links; hence, a connection whose packet drop rate is high will tend to be less used. Since in MANETs there are several approximately equivalent routes, this metric helps simplifying the logical network topology by selecting safest routes without any significant performance overhead.

Figure 3.7 illustrates both routing policies; shortest path is represented with dashed hops and ETX link quality with plain hops. Values on edges represent the packet drop rate of the associated link; the lower is the value the better is the link quality.

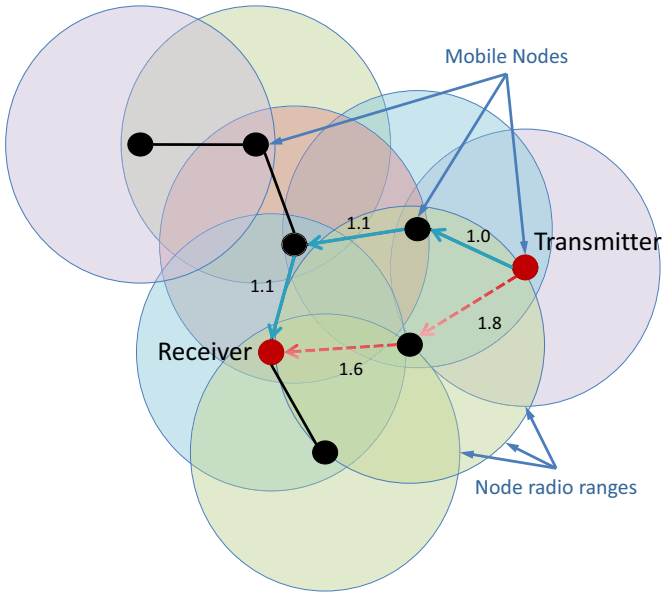

Figure 3.7: *OLSR routing alternatives*

OLSRD features: nameservice. In regular wired networks, Domain Name Service (DNS) is in charge of translating hostnames into network addresses. In the context of our framework, this service is important. As a matter of fact, JADE and therefore BAF agents exchange messages through TCP protocol using their host hostname.

In Ad-Hoc networks this type of service is not available by default. Some solutions exist like "Avahi" [98] or "Bonjour" [97]. These solutions are over-dimensioned for the purpose of the BAF; furthermore the chosen implementation of OLSR provides the "*nameservice*" plug-in which satisfies this basic DNS requirement.

The plug-in enables nodes to broadcast their hostname and IP address to other nodes of the network. Thanks to this, every node hosts a local DNS-like name service which offers a name to IP address translation service.

The BAF AP that compose the pervasive platform can then transparently use this information to enables communications between agents.

As a conclusion the OLSR protocol has been chosen to ensure the node connectivity due to its capability to route packets through the network in a reasonable time, it also has a scalable traffic overhead, supports well TCP protocol and offers valuable services like the *nameservice*. To validate this choice based on the bibliography and network simulations, additional tests in real conditions have been conducted. Protocols and results of these tests are presented in Annex B.

OLSR and Network Agent. During experiments with JADE and its different Message Transport Protocols (MTPs), we verified that inter-platform communications are based on hardware hostname while using the Hypertext Transfer Protocol (HTTP) MTP. The chosen OLSRD implementation offers an opportunity to transmit the hostnames with IP over the ad-hoc network thanks to the *nameservice* feature. Figure 3.8 explains how with this IP to hostname translation we can discover peer platforms on every connected node.

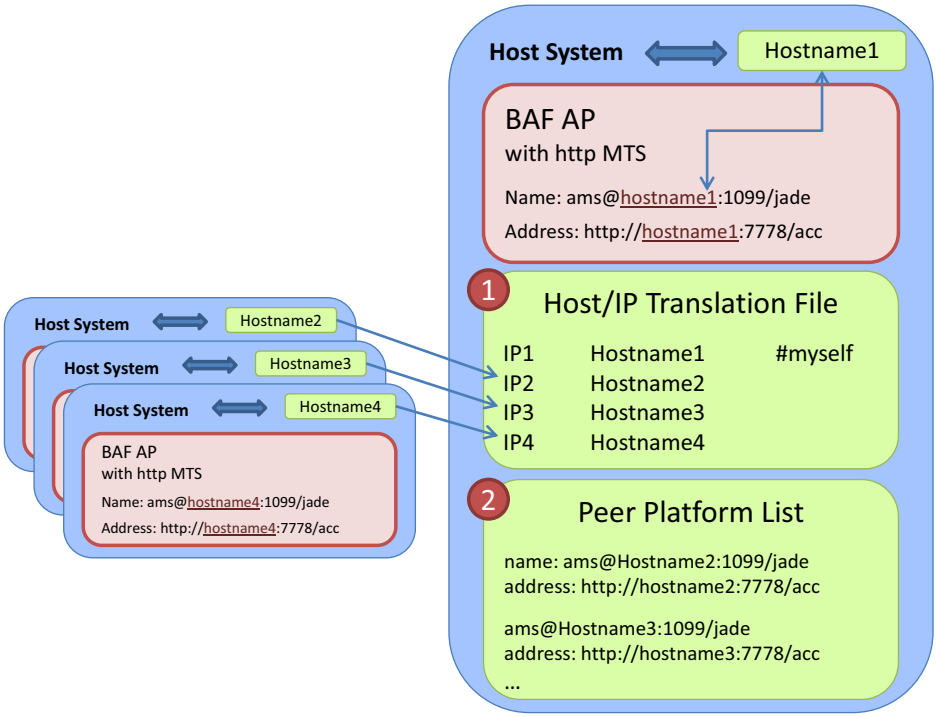

Figure 3.8: *Distributed hostname/IP translation*

In UNIX / Linux environments a local IP to hostname table is stored in `/etc/hosts` [1]. OLSRD *nameservice* manages this file with recovered hostnames/IP couples.

The BAF mandatory Network agent (named `N`) has been designed to create a bridge between OLSRD low level services and the classical JADE platform while offering higher level services such as *peer search* and *service search*. Therefore `N` agents are responsible of all network related aspects of the proposed framework.

Figure 3.8 presents the first service of the Network agent. It reads OLSRD provided hostnames and translate them into JADE AP HTTP addresses (label 1). This information is stored into a peer platform list (label 2). Peer AP list elements are composed with HTTP MTP mandatory elements, actually the HTTP MTP version of the `AMS` Agent IDentifier (AID). This AID is made with the HTTP address of the JADE AP and the corresponding `AMS` address as shown in the following example:

- AP address: `http://hostname:1099/acc`
- AMS address: `ams@hostname:7778/jade`

This list can be considered as a white page service that stores addresses of BAF peer APs. Using this list, it is possible to contact any agent hosted on any BAF peer AP. As a consequence, `N` may be considered as a permanent entry point on every peers of the platform. It is therefore possible to communicate with an agent hosted on a given peer through its local `N` agent. Figure 3.9 shows the JADE environment augmented with OLSRD and the `N` agent that provide the link between platform nodes.

On every node, the local `N` agent is responsible of maintaining this peer platform list. In addition we designed it to decode a set of node control messages. The goal is to provide a global control interface that eases application execution on the platform. As a consequence the `N` agent has been developed as an order relay and is therefore able to instantiate required agents (thanks to `O` services) and forward orders to instantiated agents. It is also able to send back agents feedback to the platform interface.

Finally the `N` agent provides two important services, to its AP local agents, *peer search* and *service search*. These service are accessible from user agents to abstract platform topology and allow inter-AP operations. These services are presented in the following paragraphs of this section.

[1] for Windows portability, the equivalent file is: *C:\WINDOWS\system32\drivers\etc\hosts*

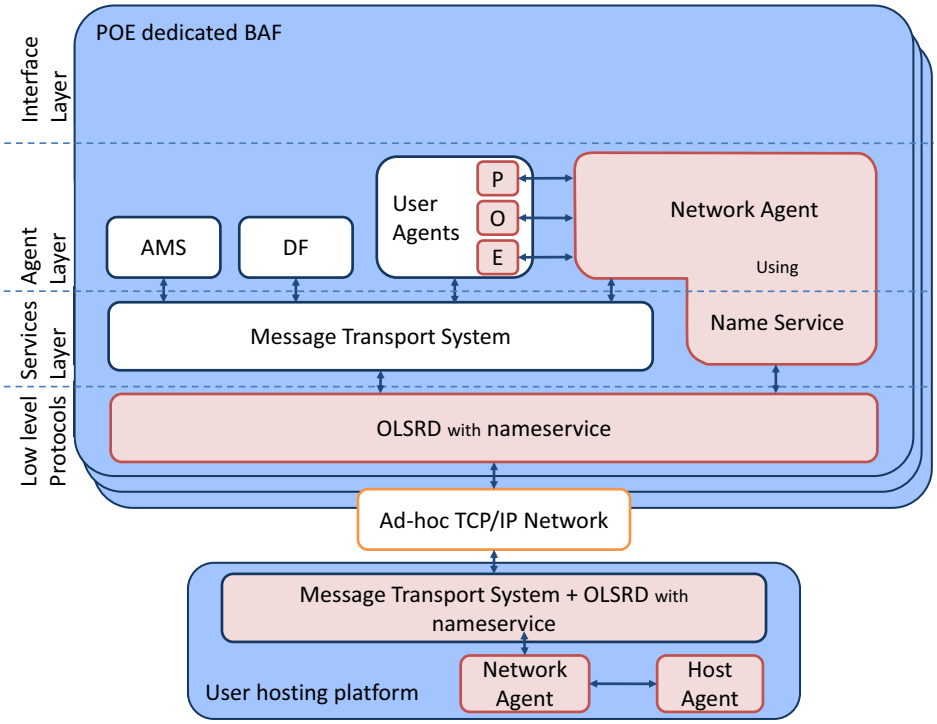

Figure 3.9: *Evolution on JADE framework for network support*

Peer Search. The *peer search* service allows retrieving the reachable peer list. As introduced in the previous section the peer search service uses the *nameservice* filled hosts file and translates retrieved hostnames into AP name and AMS addresses. The peer search service has already been illustrated in Figure 3.8 as it is closely linked to the *nameservice*. Figure 3.10 presents more details of this peer search.

Based on the peer platform list (label 1) a test is performed to check the peer agent platform availability (label 2). All peers that acknowledge their existence are published in the available peer platform list (label 3) that is used as the white page of APs.

Service Search. As shown in Figure 3.11, the *service search* service uses the previously described *peer search* service to retrieve the list of all JADE DF agents hosted on the platform (label 1). As DF is the yellow page JADE mandatory agent one DF is hosted on every accessible peer AP. Each one contains the list of all services registered on its AP. Using the DF list it is then possible for one `N` agent to ask each DF for one or more services (label 2). This allows knowing if the desired service is running in the platform. If so on which peer it is hosted and which agent provides it (label 3). Therefore it is then possible to connect to the agent that is offering the requested service.

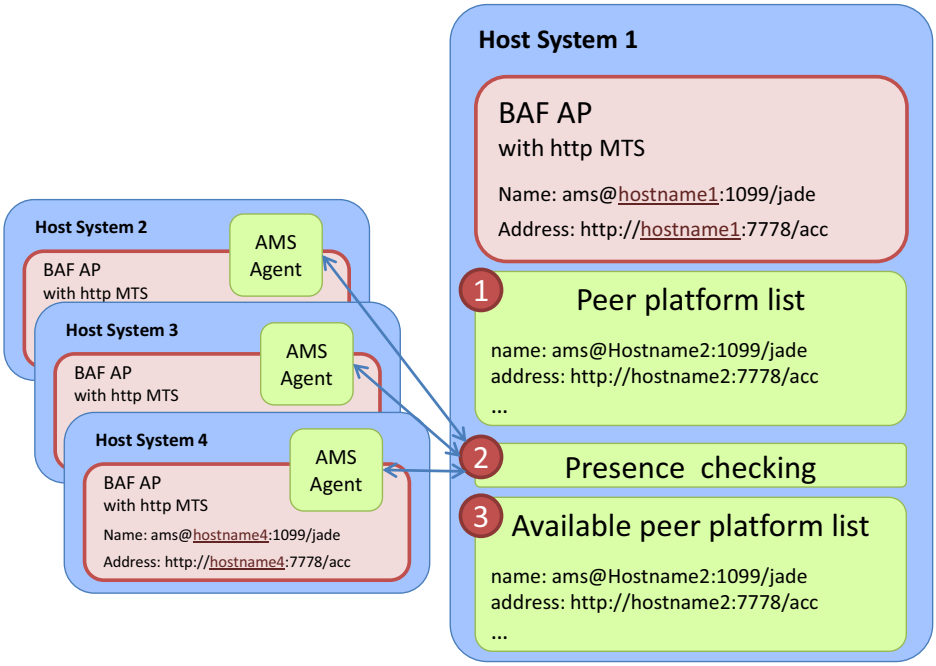

Figure 3.10: *Peer search service*

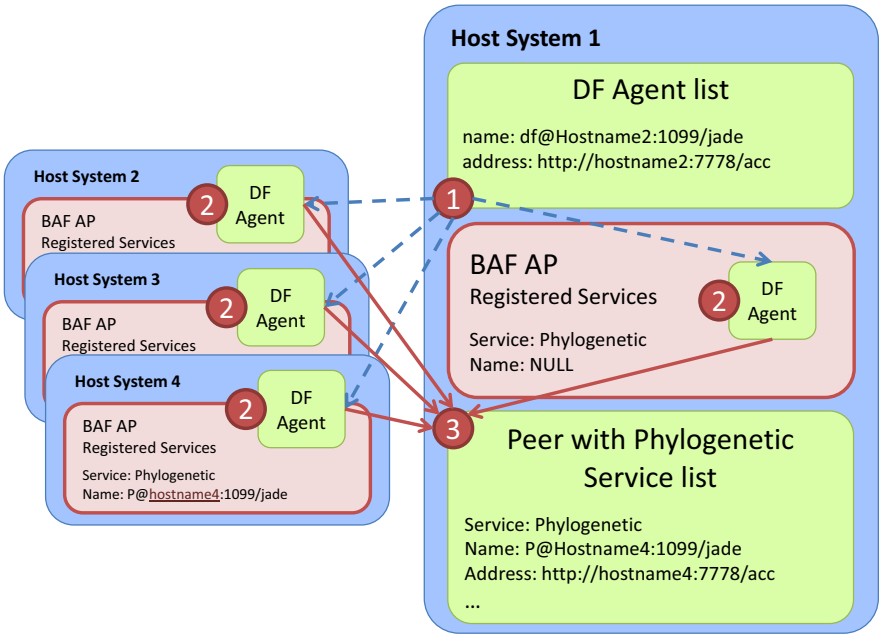

Figure 3.11: *Service search protocol*

This service is the base of an automated and adaptive platform as applications should be able to ask for a service and to connect to this service. The platform adaptability therefore requires to discover if a service is not available on the platform, and to take the decision to create it. Using the *service search* service and the `O` agent the required services may be instantiated improving the platform reliability. This can be considered as a kind of ontogenetic process applied to modular software.

The presented services are parts of the features we added to JADE. These services are mandatory for platform discovery, and to allow autonomous modular applications deployment. The integration of the OLSR routing algorithms and services as well as the Network agent services provides solutions to the two first JADE limitations.

The third limitation concerning the sparse platform control is then addressed by the `H` agent that is the subject of the following section. It abstracts services and the platform topology. Thanks to a unique GUI, `H` allows the user to launch applications on the ubiquitous platform.

3.2.2.2 User interface: Host agent

To provide an easy-to-use framework to the user, the actual platform structure has to be abstracted. This can be done by providing a unique interface that unifies the platform in the user point of view. This interface should broadcast commands as well as centralize application results and runtime information.

The host agent has been designed to provide a control interface to the platform made with a command launcher and a data collector as shown in Figure 3.12. The use of java Swing interface makes it a portable graphical interface that allows user controlling and using the BAF distributed platform. Based on windows pop up, the GUI provides guidelines that help user to deploy applications on pervasive hardware platforms.

As presented in Figure 3.12, using `N` services the GUI is able to provides the list of connected peers. This allows user to select a set of them as targets for its application deployment.

This deployment is done through multicast commands where agent classes, parameters and initialization files are sent on peer nodes. Figure 3.13 depicts the interactions



Figure 3.12: *GUI main window of the BAF*

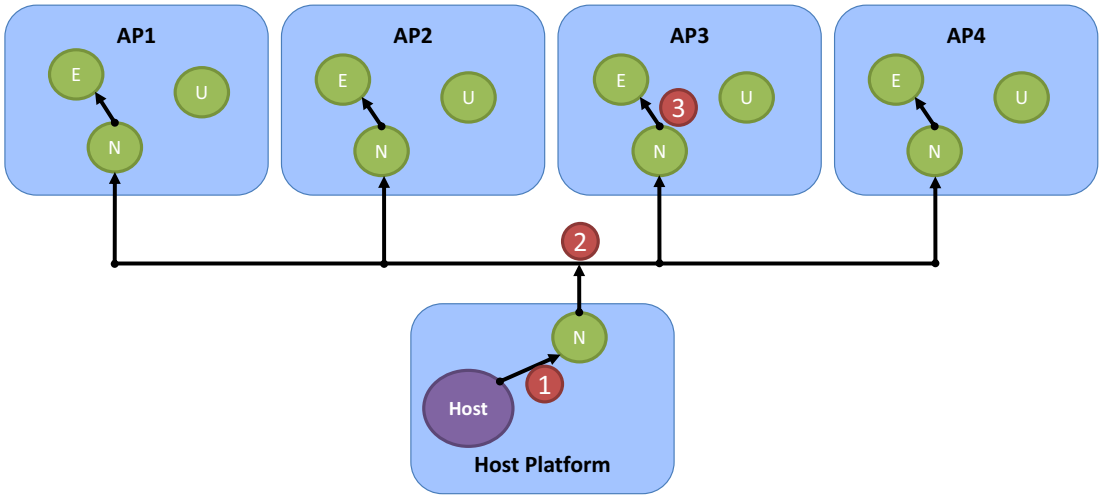

Figure 3.13: *Framework agents command interactions*

that take place in case a command is executed on a hardware platform made with four computing nodes and the host workstation.

Execution takes place as follow:

1. The `H` agent sends the command to the `N` agent. This command can either be a search service command, a peer service command, a file transmission, a result request or an application related command (start, stop).
2. This command is then multicast to the selected network agents.
3. It is then transmitted to the destination agent that will interpret it.

The following sections briefly list the GUI available platform controls and provided services. Applications require several of these services and controls to be deployed and executed on the platform. From the initial class file transfer to the application end the following services are use to have the application executed.

File transfer. The ability to transfer files in a distributed and even pervasive architecture is important for platform adaptability. Section 2.3.2 introduced the programming of SN architecture where applications or system updates are commonly deployed from an entry point; this is achieved using a *file transfer* service. The proposed `H` agent has therefore been developed to include two file transfer protocols:

1. Application files such as source codes or initialization files are transmitted from the host workstation to the targeted nodes. Classes are then stored on the node to be used at runtime.
2. Result files can be uploaded from nodes to the host platform at the end of the application.

Both protocols rely on the FIPA agent communication principles. Data files are sent as content of agent messages that are decoded and written when received by `H` or `N` agents.

Send Parameters. Depending on the desired application, some parameters may have to be sent to peers. This service allows sending different parameter sets to one or more peers on the platform. An example of application parameters to be sent is shown in Figure 3.14.

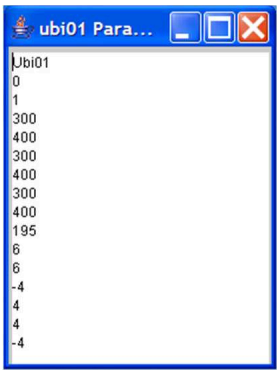

Figure 3.14: *Application parameters examples*

When these parameters are received the targeted node prints some messages that are presented in the green lower frame of Figure 3.15. The red upper frame in Figure 3.15 shows some platform related information such as addresses and names of the Host platform and other nodes.



Figure 3.15: *Node received its parameters*

Start application. The *start application* service is available from the user interface and is used to launch the application on the platform once it has been prepared and its parameters defined. This command proposes to select the nodes that will receive a given command to execute. The upper red frame in Figure 3.16 shows an application node messages printed at the application startup.

Stop application. The stop application service is also available from the user interface. It provides elementary services to stop the application and to gather results in a clean manner. The lower green frame on Figure 3.16 shows an application node messages printed at the application shutdown. The gathering of results takes place before agent shutdown phases. This result gathering is based on the above described file transfer service allowing user to rapidly get applications results on its host platform.

Execution mode. With specific hardware equipped architectures such as the Perplexus platform user is able to select an execution mode, two modes are provided, the SW mode and the hybrid HW/SW mode. When using the framework in a distributed platform that is not embedding specific hardware such as the Ubichip, user is only able to launch software applications based on agents.

When designed or used in SW mode, agents run on the software side of the node (or Ubidules is this precise case). Additionally, none of the hardware features of the Ubichip can be used with JADE-only agent specifications. The agents can freely establish communications and exchange messages with other agents running either on

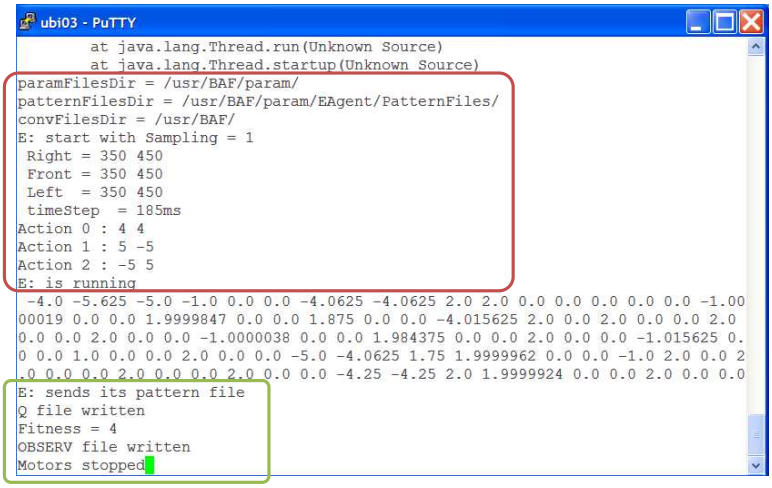

Figure 3.16: *Application start and stop commands*

the same hardware platform (i.e. Ubidule) or on distant platforms. They can furthermore undergo some specific bio-inspired processes such as agent or service creation, replication or destruction, etc.

On the contrary, the hybrid HW/SW execution mode which authorizes agent migration between software and hardware is currently proposed when the framework is being executed on the Perplexus platform. This approach is based on a restrictive specification of agent behaviors. Agents described with respect to these restrictions can have their kernel (i.e. computational part) compiled to the Ubichip in SIMD mode. This approach is more widely presented in Chapter 4 as the second contribution presented in this document that allows to take advantage of this parallel accelerator.

The mode selection is made at the application deployment but can be switched during its execution. A *mode switch* service has been developed to allow users to use this functionality.

Conclusion on the User interface. The third JADE limitation is the sparse control of the platform it offers. By abstracting the hardware platform the designed `H` agent provides a unified interface to control the set of agent platforms composing the pervasive system. Figure 3.17 shows the schematic of framework with its user interface that provides a unique entry point to control the hardware platform. Additionally, `H` gets results and traces thanks to a dedicated spy agent. This spy agent is presented in the following section.

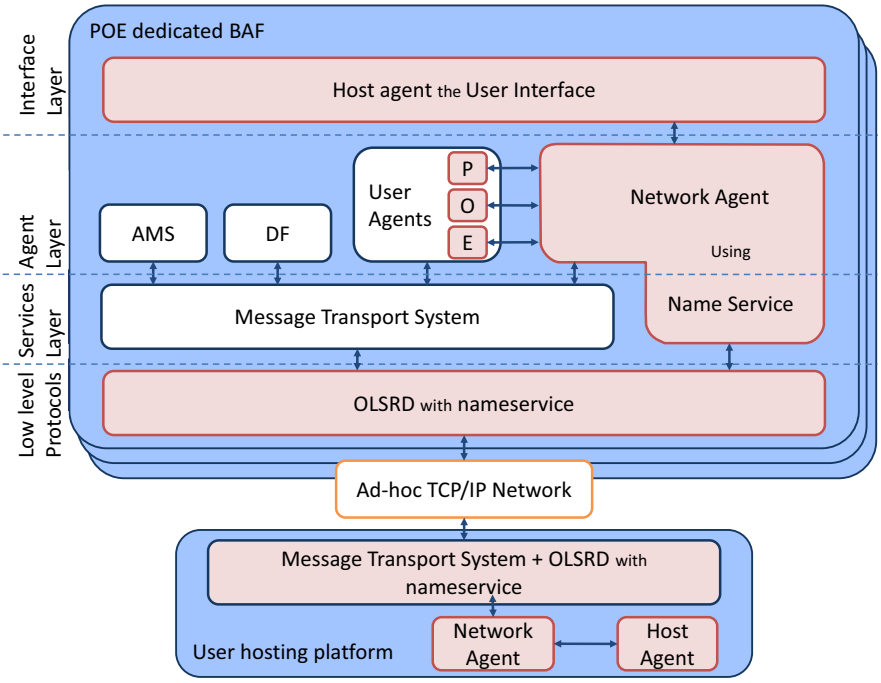

Figure 3.17: *Evolution on framework: Host agent*

3.2.2.3 Debugging and tracing activity

The spy agent also called `S` has been designed to provide debugging and tracing services inside the framework. `S` services can be used by user agents to report bugs or status as well as results. Two features are provided within this agent they are presented in the following.

The *SpyMessageBehaviour* can be added in user agents; in this case the agent regularly sends its status and results to the local S agent as periodic reports. Then `S` stores the activity in two different files:

- A file dedicated to the sender activity report. It gathers the activity of a given agent providing debug and activity information.

- A general file. In this file the entire AP activity of the platform is stored. Thanks to sequentially logged messages incoming from every agent in the platform events may be analyzed and give some information about the application runtime.

This allows the user to easily trace the activity of agents during an application debugging or result analysis phases.

The *bugBehaviour* described in the `S` agent branch is intended to collect error messages resulting from exceptions that may occur in local agents. As there may be no interface for each module of the platform, this service allows users to gather information in case an exception occurs. To use this service user agents have to call the *bugBehaviour* in every Java exception management blocks.

Both of these services are optional but are relevant and useful for debugging and testing purpose.

3.2.2.4 HW\SW communication

When the framework is used with ubiquitous computing platforms, agents may be intended to interact with the environment through sensors and actuators. The Interface agent (`I`) provides methods to access the sensors and the actuators of the node. As an example Perplexus Ubidules embedded onto Tankbots are able to control this artificial body through the `I` agent. This agent provides "on-demand" services that read sensor values and send command to the node actuators. It has been designed as a native library to deal with hardware from the Java Virtual machine that actually runs the BAF. Annex A presents the `I` agent and more precisely the Java Native Interface (JNI) library we designed to control the TankBots used in the applications presented in section 3.3.

Due to our specific Perplexus case study an additional agent has been designed to be devoted to the communications between the processor and the Ubichip (presented in section 2.5.1). Consequently this agent is used in conjunction with the hybrid HW/SW execution mode. This Ubicom `U` agent should be only used when the framework targets the Perplexus hardware. The `U` Agent contains methods that are dedicated to the management of the Ubichip. Methods like writing Ubicell configurations, reading or writing Ubichip memory, enabling or disabling Ubichip operating mode are provided and allow to use the Ubichip in both native or SIMD modes, from the BAF.

It has been designed with the same JNI methodology than for the `I` agent as it allows to directly interacting with hardware. As a conclusion, the `U` is responsible of

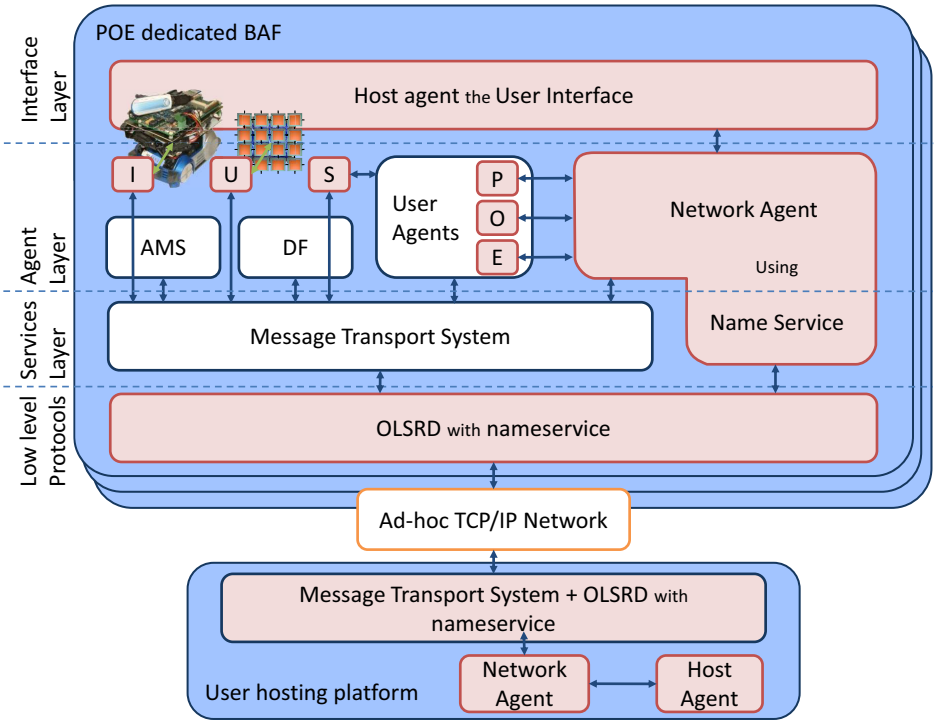

Figure 3.18: *JADE enhancement for HW/SW interfaces and activity tracing*

the bidirectional communication between the BAF and the Ubichip. Chapter 4 presents this functionality of the BAF with more details.

Figure 3.18 presents the BAF with interfaces and debugging enhancement over JADE.

Thanks to the presented BAF agents we believe that pervasive platforms may be programmed in a reliable manner. The proposed bio-inspired enhancement has been designed to improve the adaptability of the targeted platforms. The following section introduces the three applications we designed as proof of concept to test the BAF with an example of application scheduling.

3.2.2.5 BAF in POE applications

Figure 3.19 presents an example of a POE application based on the interactions between three computing nodes and a host platform. Low level services as well as tracing messages are not illustrated for clarity reasons. However most of communications between agents take place in the computing nodes, and only mandatory messages are

exchanged between these nodes. For example, the `Host` agent is quiet after the initialization phase and the sole `Phylogenetic` agent have inter-node communications during specific phases (i.e. at the end of generations).

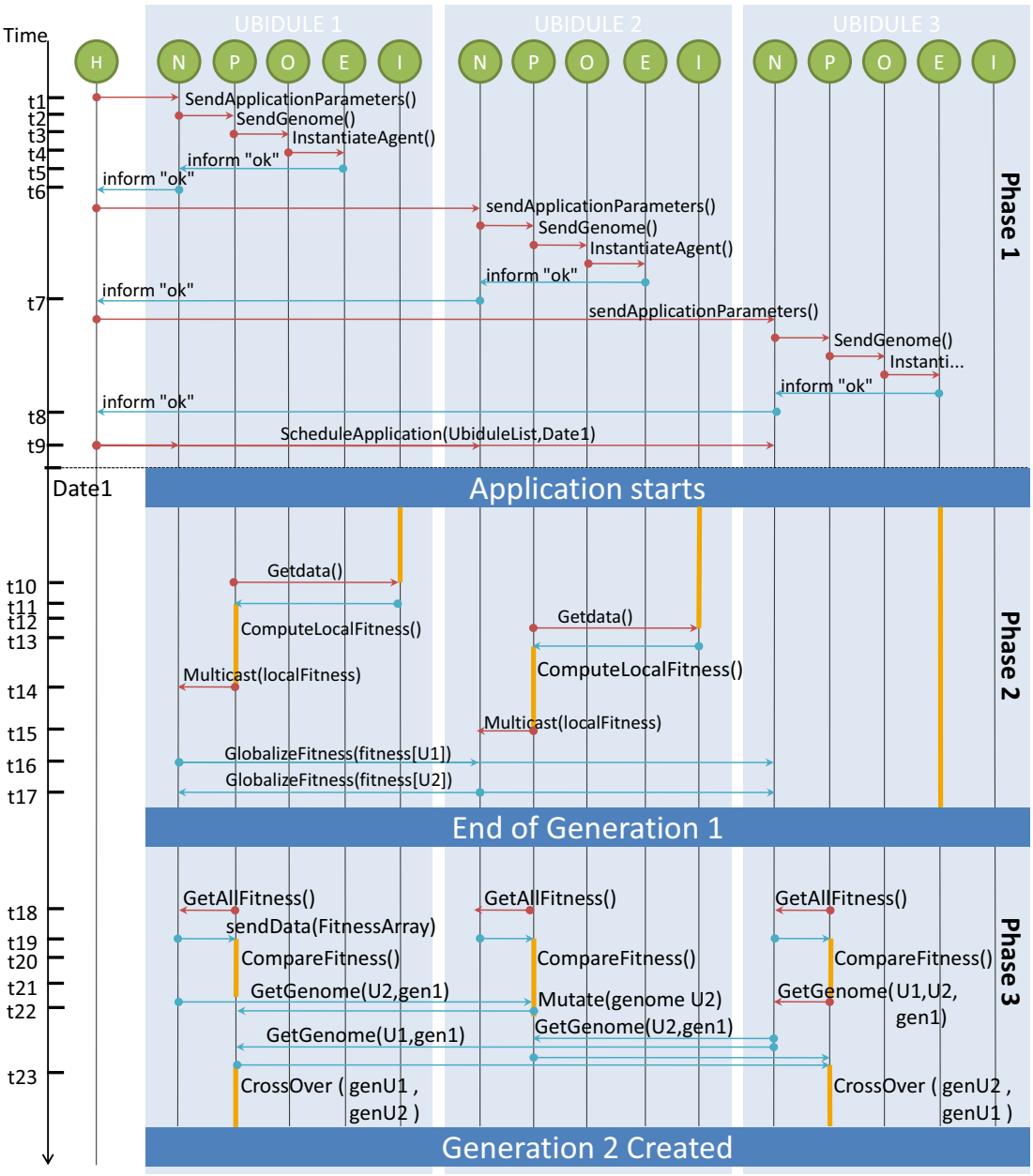

Figure 3.19: *Example of a POE application scheduling with three nodes*

The presented POE application is mainly composed of three blocks:

1. One *initialization* of the platform is required to prepare the application execution.

2. As the application is based on phylogeny several *generations* will be run until the end of the application. During this phase each node executes its task this parameter extracted from its genome.
3. Between two consecutive *generations* an *Inter-generations* phase occurs. During these phase nodes globalize their data to allow the next generation process to take place at the platform level.

In Figure 3.19 time steps from `t1` to `t9` corresponds to the initialization of the first generation. Each node is configured in the same way, parameters are sent to the local `N` agent (label `t1`). These parameters are then addressed to the `P` agent (label `t2`) that generate the first generation genome. This genome is transmitted to the `O` agent (label `t3`). Based on the received genome the `O` agent is able to create the appropriate agents like `E` in this case (label `t4`). Once this agent is running, either in SW or hybrid HW/SW execution modes, a confirmation message is addressed to the host (labels `t6`, `t7` and `t8`), through the local `N` agent (label `t5`). When all nodes have been configured the application execution is scheduled (label `t9`).

Application starts at the scheduled date and lasts different time depending on the node genome and its fitness. When the individual simulation ends, data are transmitted to the `P` agent this allow the local fitness to be computed (labels `10` to `13`). This fitness is broadcast to other nodes (labels `14` to `17`).

At the end of the generation lifetime all local fitness are compared with the fitness of the other individual (labels `18` to `20`). Decision is then taken to cross and mutate best genomes to obtain the following generation individuals to simulate (label `21` to `23`). The next generation is then ready to be simulated. *Generations* and *inter-generations* phases are then alternatively run until a satisfying solution is found. Applications based on phylogenetic processes presented in the following section are based on a similar scheduling.

The conducted applications we present in the following section only make use of the software execution mode. This is due to the unavailability of the Ubichip at the time of this writing. We expect that running epigenetic processes in hardware mode will allow to design learning applications that exhibit more interesting and complex emerging behaviors.

3.3 Applications

Three case studies are presented in this section for illustrating bio-inspired applications based respectively on evolution (P), learning (E) and a PE features combination. These applications are powered by the proposed framework and are proof of concept for the BAF. Presented applications only make use of software mode[2], they are executed on Ubidule prototypes that are embodied into TankBot skeletons. Figure 3.20 schematically depicts the used robots, their sensors and actuators as well as the framework agents presented previously.

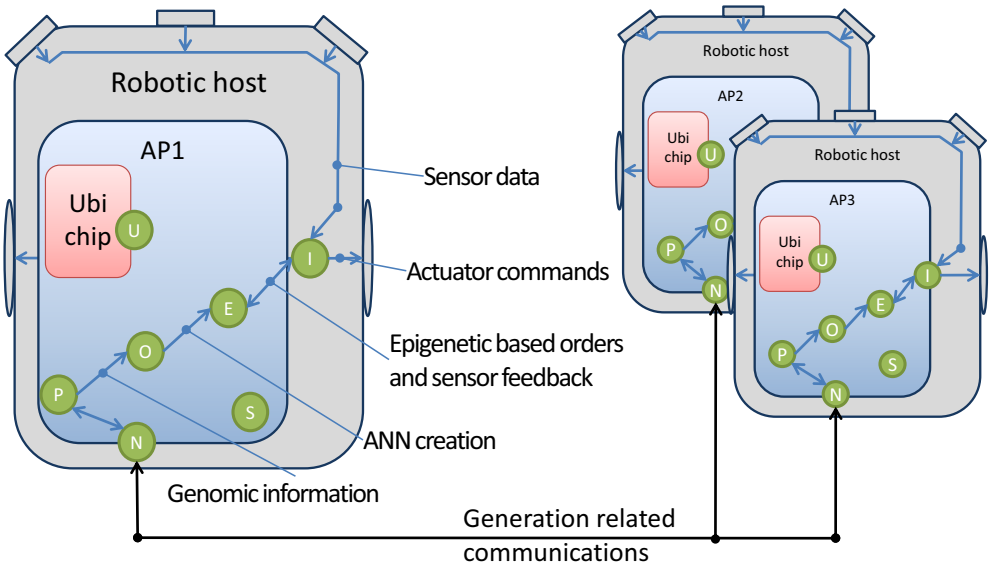

Figure 3.20: *General application environment*

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 thanks to the `I` agent. The `P` agent is responsible of the robot controller evolution. It computes the robot fitness and runs the genetic algorithms with the `P` agents of other robots using `N` services. The `O` agent instantiates the `E` agent based on the genome provided by `P`.

3.3.1 Evolving simple robot controllers

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;

[2]The `U` agent is unused here as the chip is still under fabrication.

the robot controller (E agent) being here a simple LUT. It 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 involved robots is to run one lap.

Figure 3.21 shows the principle of this genetic race the lap time gives the fitness of a given robot controller and hence its genome which is the LUT content (i.e. actions associated to sensors stimuli). Therefore there is no learning in this application, changes in the robot behavior being driven by evolution only.

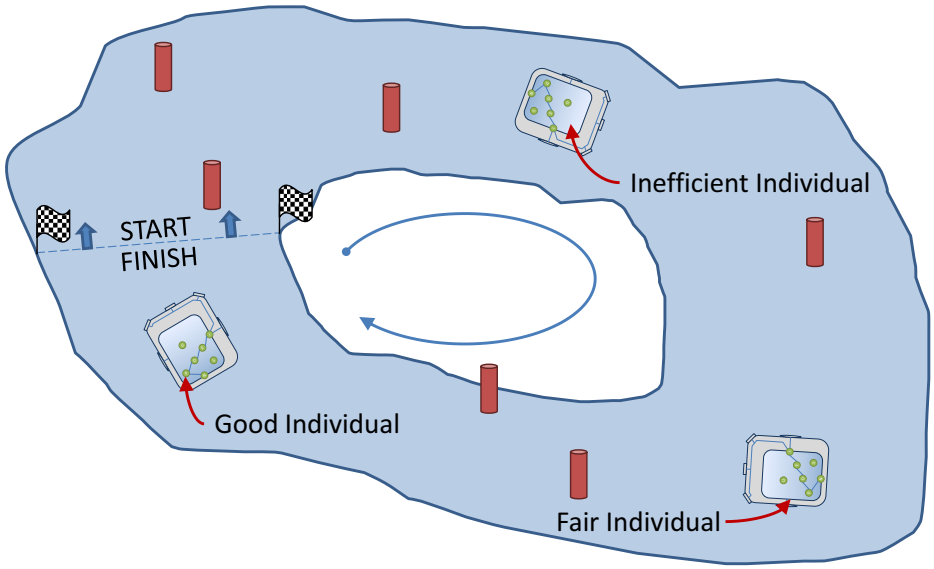

Figure 3.21: *Evolution-driven application overview*

The individual fitness is computed thanks to the Equation 3.1. Where a time limit T_L is set to reduce the generation duration. C reflects the number of touched obstacles, this is a collision penalty. Finally T_I is the time spent for the individual to achieve one lap of the track.

$$Fitness = \frac{T_L - T_I}{C + 1} \tag{3.1}$$

This fitness therefore selects individuals that are able to finished one tack lap in a reasonable time while minimizing the number of obstacles they collide into. These individuals are crossed and/or mutated by the P agent to create the next generation therefore replacing individuals with inadequate behaviors. Once a new individual genome is ready, the P agent forwards it to the O agent that instantiates the corresponding E agent as represented in Figure 3.22. Generation after generations robots exhibit better

behaviors. This tends to prove the reliability of the software as well as the possibility to handle POE problems thanks to the platform.

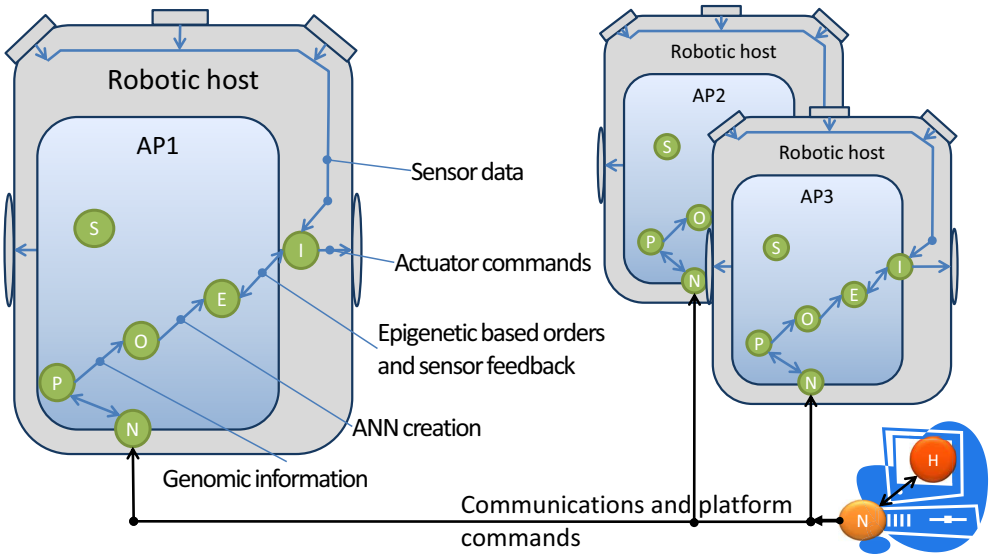

Figure 3.22: *Application involved agents*

A demonstration video that illustrates this application is available online at: `http://www.lirmm.fr/~brousse/Ubibots/Ubibots_demo_1.avi`.

3.3.2 Online collaborative learning

Robots which participate use online learning (Epigenesis) to improve their performance. Figure 3.23 shows the robots that are enclosed in an arena scattered with obstacles; collision avoidance is here the main objective. As this application only targets learning, the `P` and `O` agents are not used.

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 3.23 an ANN is in charge of controlling the robot. Figure 3.24 depicts the principle of this application: the `E` agent is a multi-layer Perceptron ANN that uses a standard back-propagation learning algorithm.

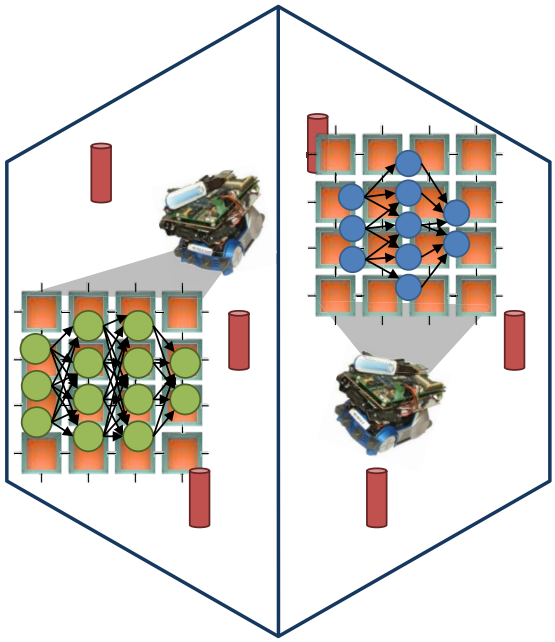

Figure 3.23: *Application overview*

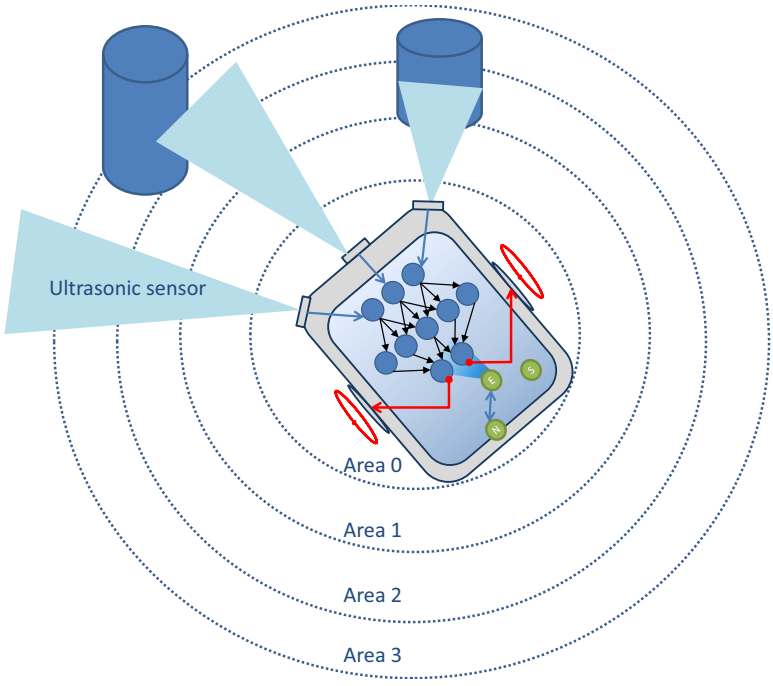

Figure 3.24: *Collaborative learning application overview*

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 3.3.

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

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

Table 3.3: *Ultrasonic sensors areas*

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 move pattern involved in the collision 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. This speeds up the convergence toward a satisfying solution.

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

To summarize the above Figure 3.25 presents agents involved in this collaborative learning application.

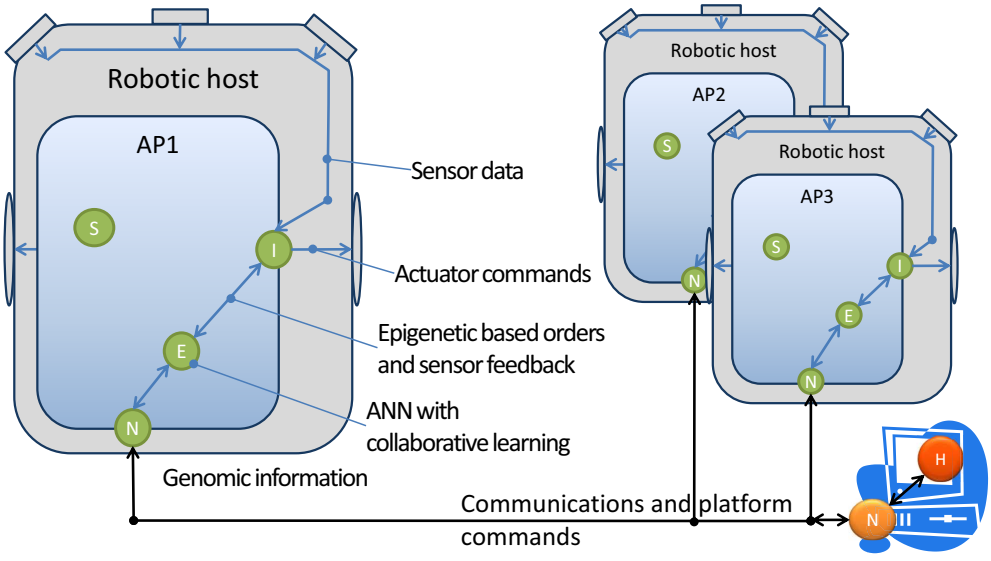

Figure 3.25: *Application involved agents*

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

A demonstration video that illustrates this application is available online at: `http://www.lirmm.fr/~brousse/Ubibots/Ubibots_demo_2.wmv`.

3.3.3 Evolving a population of learning individuals

Based on the same idea to prove that bio-inspired features are useful for distributed and pervasive systems adaptability, the third application relies on the association of `P` and `E` features as shown in Figure 3.26. We designed it as a mixed evolution and learning robotic demonstrator.

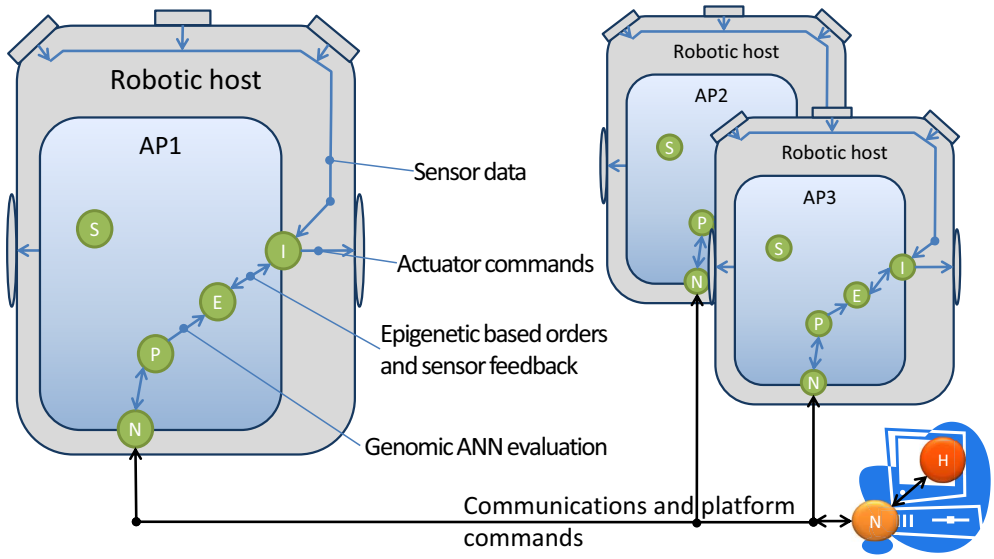

Figure 3.26: *Application involved agents*

In this application the robot behavior is set using a dynamically changing quality function created following the individual genome. This function is based on state/action couples and is modified using reinforcement learning [104]. As a consequence, robots are following move rules that depend on the present sensors state. Actions are

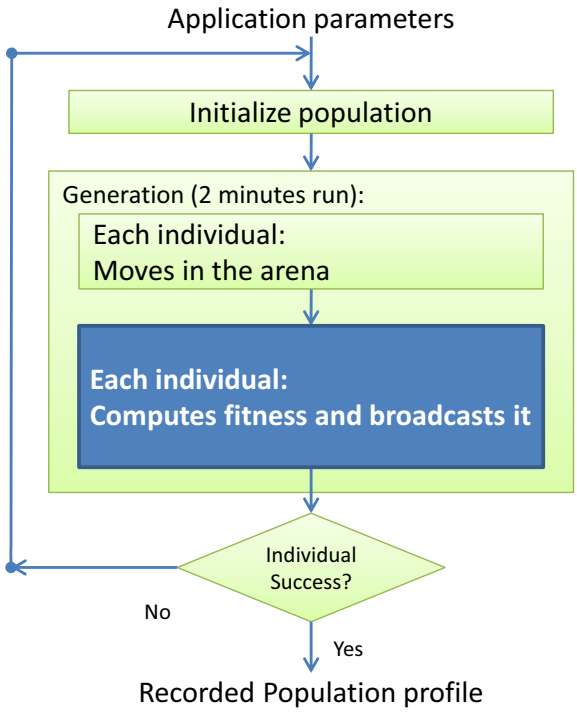

Figure 3.27: *Second generation application steps*

rewarded if they induce a successful move or punished if a collision occurred. The application runtime is described in Figure 3.27 and can be explained with the following steps:

- Robots move in the obstacle scattered arena of Figure 3.23 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 a robot gets stuck to the wall 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.

- 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.

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

- 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 3.27 blue part represents step of the application where individuals interact with each other 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.

- 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.
- 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 for a given time allowing them to acquire essential knowledge.

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 evolves with the species.

The individual genome is represented in Figure 3.28 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.

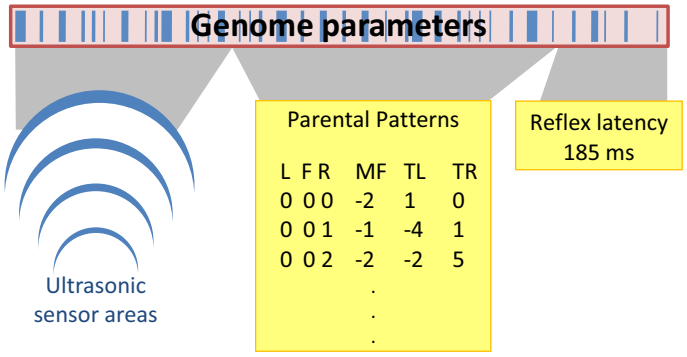

Figure 3.28: *Robot genome example*

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 3.4. Presented values are obtained after the first generation run.

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

Table 3.4: *Quality function example*

The fitness of an individual is defined in Equation 3.2 where α is the reflex normalization coefficient, S_{mean} the mean value of motion speed and B_i the collision penalty of the ith move.

$$Fitness = \alpha \times \left(\sum_{i=1}^{n} S_{mean} \times (1 - B_i) \right) \tag{3.2}$$

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 3.4 shows the resulting quality functions of individuals. In the presented case 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 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 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 3.3.1.

A video showing the runs of the above presented individuals is available online at: `http://www.lirmm.fr/~brousse/Ubibots/Ubibots_demo_3.wmv`.

3.4 Conclusion on the BAF environment

The BAF presented in this chapter has been developed with the ultimate objective of combining bio-inspiration and AOP in a framework for pervasive platforms programming. This framework has been designed with scalability, portability and adaptability in mind to allow a reliable programming of pervasive platforms.

The presented agents and services have been implemented and validated on an intermediate prototype of the Perplexus platform made of three Ubidules based on the TankBot. This setup allowed to validate the concept of the approach on the software

side; the whole JADE infrastructure as well as the presented agents are implemented on these mobile Ubidules. We are able to program this pseudo-pervasive platform and observe the behavior of the mobile nodes from a workstation running the BAF environment as well. All three bio-inspired aspects are implemented and validated in the presented applications. As examples, collaborative online learning thanks to communicating ANNs or a PE application where evolution process get improved by self individual experience compared to classical P-based applications. Finally, the Ontogenetic features reveal to be an advantage for the platform flexibility and low level adaptability as it allows agents to bear (i.e. to be created) into any node of the platform when they are required.

The interest of presented applications is limited to the proof of concept level due to setup limitations. GAs always make use of large number of individuals per simulated generations. The limited number of available robots and their limited sensing capability and limited autonomy prevent to build complex POE applications. Therefore we limited the size of the solution domain, used for the phylogenetic process examples, to obverse the evolution on a small number of generations with few individuals per generations. This point will be addressed in a near future as the MarXbot fleet will be operational by the end of 2009. This fleet is composed of up to 70 robots evolving in a closed arena. Additionally a complex battery charger is installed in this arena. Therefore empty batteries can be hot swapped by full batteries. Thanks to this collaborative robotic platform, the opportunity is offered to have collaborative robotic applications running for days. In this pervasive context, we believe that the BAF will provide a relevant way of programming reliable ubiquitous applications; and that such applications will provide interesting results in the complex system field.

Chapter 4

Jubi toolchain

Contents

4.1 Jubi methodology and language 100
4.1.1 Unified HW/SW description file 101
4.1.2 The Jubi Language 102
4.2 Perplexus accelerator: Ubichip 105
4.2.1 Kernel loading 105
4.2.2 SIMD operating mode 107
4.2.3 Ubichip modifications 113
4.3 Programming tools and validation 115
4.3.1 JubiSplitter 115
4.3.2 JubiCompiler 116
4.3.3 Code Analyzer 118
4.3.4 UbiAssembler 118
4.3.5 Functional tests 119
4.4 Experimental results 120
4.4.1 SNN simulator algorithm 120
4.4.2 GP-GPU solution: nVidia architecture 122
4.4.3 Compiling applications for GP-GPU 123
4.4.4 Experimental results 125
4.4.5 Results interpretation 125
4.5 Conclusion on the Jubi Methodology 133

CHAPTER 4 ABSTRACT

In this chapter we present the methodology and associated tools we propose to compile hardware accelerated kernels. This work allows taking advantage of the Ubichip by compiling functions from software source code therefore reducing the time of the application development and implementation complexity. The generated code is then compared to classical solutions for a Spiking Neural Network application.

L'amour est une accélération de la vie
"Love is an acceleration of Life"
Jacques CHARDONNE ,
L'Epithalame.

In the previous chapter we introduced the BAF programming environment that provides adaptability to pervasive platforms. This adaptability greatly relies on the previously presented bio-inspired features. Features such as GAs and ANNs are known to be computing power demanding. Both of these examples are involving numerous elements (individuals, neurons, etc) with consequent amount of data and processes to execute. Therefore embedded devices used as modules in pervasive platforms may not offer the requested computing capabilities. This greatly affects applications execution time either at module and platform levels where nodes maybe waiting for other nodes to complete their part of the application.

As presented in Chapter 2, current SNs may be considered as early ubiquitous platforms. Such platforms are based on embedded devices. The designed BAF may be use to gather these embedded nodes into a scalable and user-friendly platform. The counterpart of this improved scalability is the limited hardware capabilities of a single node. This limits the platform application field. The use of hardware accelerators, as in the Sniper localization example presented in Section 2.3.1.3 is an efficient solution to improve performance. Some accelerators are flexible and even reconfigurable, others are application specific. This general solution may be applied to the nodes of pervasive platforms. As an example, in the Perplexus project the Ubichip presented in section 4.2 and in [82] has been designed to accelerate the execution of bio-inspired features at the node (i.e. Ubidule) level. If each node of the platform is endowed with one or more accelerators, the overall platform performance is improved; therefore this opens interesting opportunities to run more demanding applications. Accelerators may be classified into to two major categories *generic* and *embedded*. *Generic* accelerators are able to deliver high level performance as they are not constrained with power consumption problems unlike *embedded* accelerators. Obviously, Ubichip as other embedded accelerators are less powerful than their generic counterparts. For example

the Ubichip consumes approximately 3 Watts compared to the 250 Watts of a GPU board; they propose some tradeoff between performance and power consumption. A direct consequence, as we are using embedded accelerators, obtainable performance improvements will be lower than the one we would have expect using an accelerator designed for the *generic* use.

The main drawback of using hardware accelerators to improve performance is that they have to be programmed or configured. The languages and tools used for this purpose may be different than the one used to program the application: VHSIC Hardware Description Language (VHDL)/C, etc. versus Java in our case. In parallel architectures, this issue increases the difficulty of programming applications at the node level. This issue is critical in the context of pervasive platforms. As a matter of fact, users already have to take into account the application programming at platform level in addition to this node level. Moreover the multiplication of these programming levels is error prone and should be avoided as much as possible.

The objective of the thesis is to provide a global framework that eases the programming of pervasive platforms. To this purpose an agent-oriented framework has been proposed in Chapter 3. This framework eases the programming of pervasive platforms by allowing to program applications independently of the platform topology (i.e. distribution of nodes in the space) and hardware heterogeneity. Using hardware accelerators hardly suits the desired hardware independence of the proposed framework. Reconfigurable accelerators such as Complex Programmable Logic Devicess (CPLDs) [114, 128, 123], FPGAs [127, 115, 124], or Ubichip in native mode clearly require complex code translations to map the desired functionality implemented on the target. This kind of accelerators:

- relies on specific architecture.
- with heavy processes to adapt the configuration from a device to another (i.e. one bit-stream is required for each kind of reconfigurable device involved in the platform).

As a consequence, they may be roughly compliant with the proposed hardware independence and user-friendly policies of the platform programming.

To improve performance while keeping the platform programming user-friendly is a challenge this chapter intends to tackle. The proposed solution is focused on

the exploitation of the explicit parallelism of applications directly extracted from user descriptions of agents. The proposed approach is introduced, validated and tested with Ubichip SIMD mode in this chapter. Nevertheless the associated methodology is generic enough to be adapted to other accelerators.

This chapter presents this contribution starting with an introduction of the proposed methodology to program HW/SW application. Then in a second section of this chapter, the Ubichip, used as a case study platform, is presented with details on the SIMD mode. The third section of the chapter then focuses on tools we designed to adapt the generic proposal to the Ubichip as well as their validation thanks to VHDL simulations. Finally, experiments and obtained results are presented to lead the discussion toward conclusions about the proposed approach.

4.1 Jubi methodology and language

The approach introduced in this chapter has been proposed to solve the possible issues of programming parallel accelerators as the SIMD mode of the Ubichip. If we consider a program as an alternate sequence of sequential sections and parallelizable sections as depicted in figure 4.1, the inherent parallelism of these latest sections may be used to accelerate it. In order to allow such acceleration, parallel parts must be extracted from the original application to be compiled and executed on the SIMD parallel accelerator.

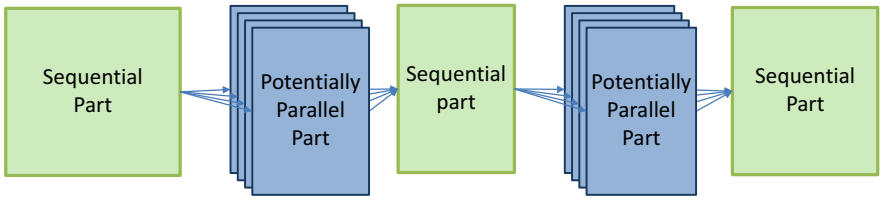

Figure 4.1: *Sequential and parallel mixed parts of a program*

In the case of the Perplexus platform, the Perplexus Ubichip - that falls into this accelerator category - may be used to increase performance of parallelizable applications. Within Perplexus a biologically plausible neural network simulator has been used as test application. This Spiking Neural Network (SNN) uses the model presented in [117] and is massively parallel as hundreds of neurons owning hundreds of connections have to be simulated. This simulator is therefore a typical example of application that benefits from the use of parallel hardware accelerators.

4.1.1 Unified HW/SW description file

To really take advantage of this accelerator, while using the BAF that is Java-based, we chose Java as source programming language. This allows user to easily port its BAF applications to the Ubichip. The goal of the proposed approach is therefore to compile HW/SW hybrid agents directly from a user source code as proposed in ATI Stream Computing or nVidia CUDA. Figure 4.2 introduces the Java for Ubiquitous (JUBI) methodology and associated tools, the current section develops the proposed methodology.

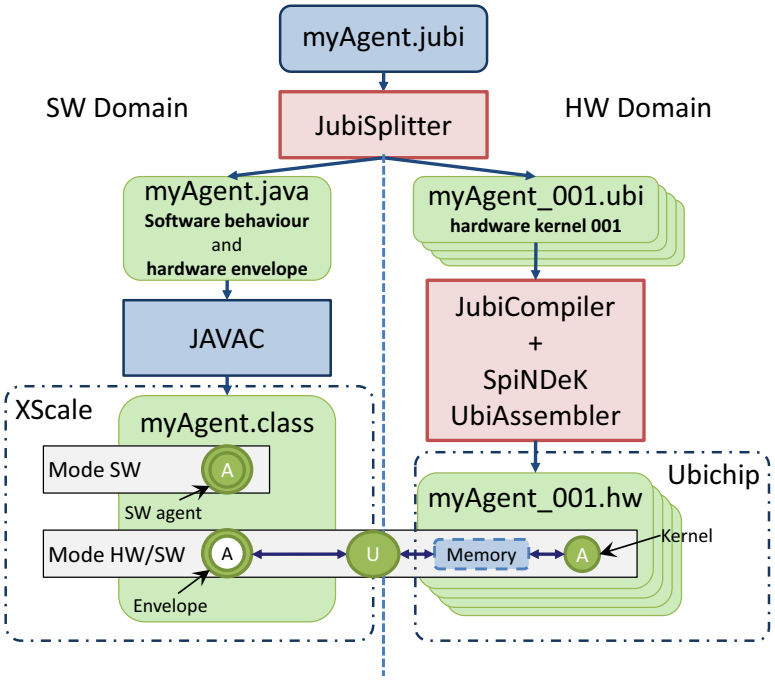

Figure 4.2: *Jubi compilation flow, from source code to HW/SW hybrid execution*

The JUBI approach takes one agent description and provides all required binaries and bytecode to run this agent either in pure software mode or in HW/SW hybrid mode. The agent code must be written in a ".jubi" file. This agent is in essence a classical JADE agent with some sections of code flagged with dedicated directives. This file is preprocessed by a code separator, called JubiSplitter, which produces two files, a ".java" file and a ".ubi" file:

- The ".java" file embeds two JADE *behaviours*:
  - A software *behavior* that represents the monolithic agent instance.

  - A hardware *behavior* that implements the non-parallel sections of the code, called the *envelope*, along with the necessary function calls for communicating with the hardware.

- The ".ubi" file represents the parallel code extracted from the JUBI file. These codes are called *kernels*.

The left part of figure 4.2, shows that the Java file (myAgent.java) is fed into the Java compiler (javac), which produces a JADE agent class. In case of a pure software execution, the *softwareBehaviour* is selected. This behavior embeds functionally equivalent descriptions of the JUBI parallel parts and corresponds to the monolithic agent implementation. In case of a hybrid SW/HW description, the *hardwareBehaviour* of the agent is selected. This behavior and the rest of message-passing interface of the agent constitute the envelope. Communication between this envelope and the targeted hardware accelerator are handled through a dedicated BAF agent called Ubicom as briefly introduced in Section 3.2.2.4 and illustrated in Figure 4.2.

The right part of figure 4.2 is dedicated to show the processing of the hardware side of the application. The hardware file (myAgent.ubi) is read by the JubiCompiler, which produces an assembly description for the targeted parallel accelerator. This program, also called kernel, is assembly and loaded in the accelerator program memory thanks to the Ubicom agent.

To summarize, the proposed approach is based on the separation of software sequential descriptions and hardware parallelizable sections present in an agent oriented application. The JUBI language has been designed to provide this code separation; the following section gives details on this language.

4.1.2 The Jubi Language

The JUBI Language has been introduced to cope with the hardware and software description separation. The main guideline during the JUBI specification was to keep the maximum compatibility with Java allowing the JADE agent formalism to be preserved (Agent description, behavior description, mandatory methods...). The major differences with a Java description rely on specific directives that help flagging code sections for hardware execution.

The following code presented in Figure 4.3 gives a simple example of a JUBI agent. The relevant code in the JUBI file is red-colored and marked with a brace. The processing of this JUBI file leads the creation of the presented Ubi and Java files where white (and respectively red-colored) codes correspond to the relevant code of the JUBI file.

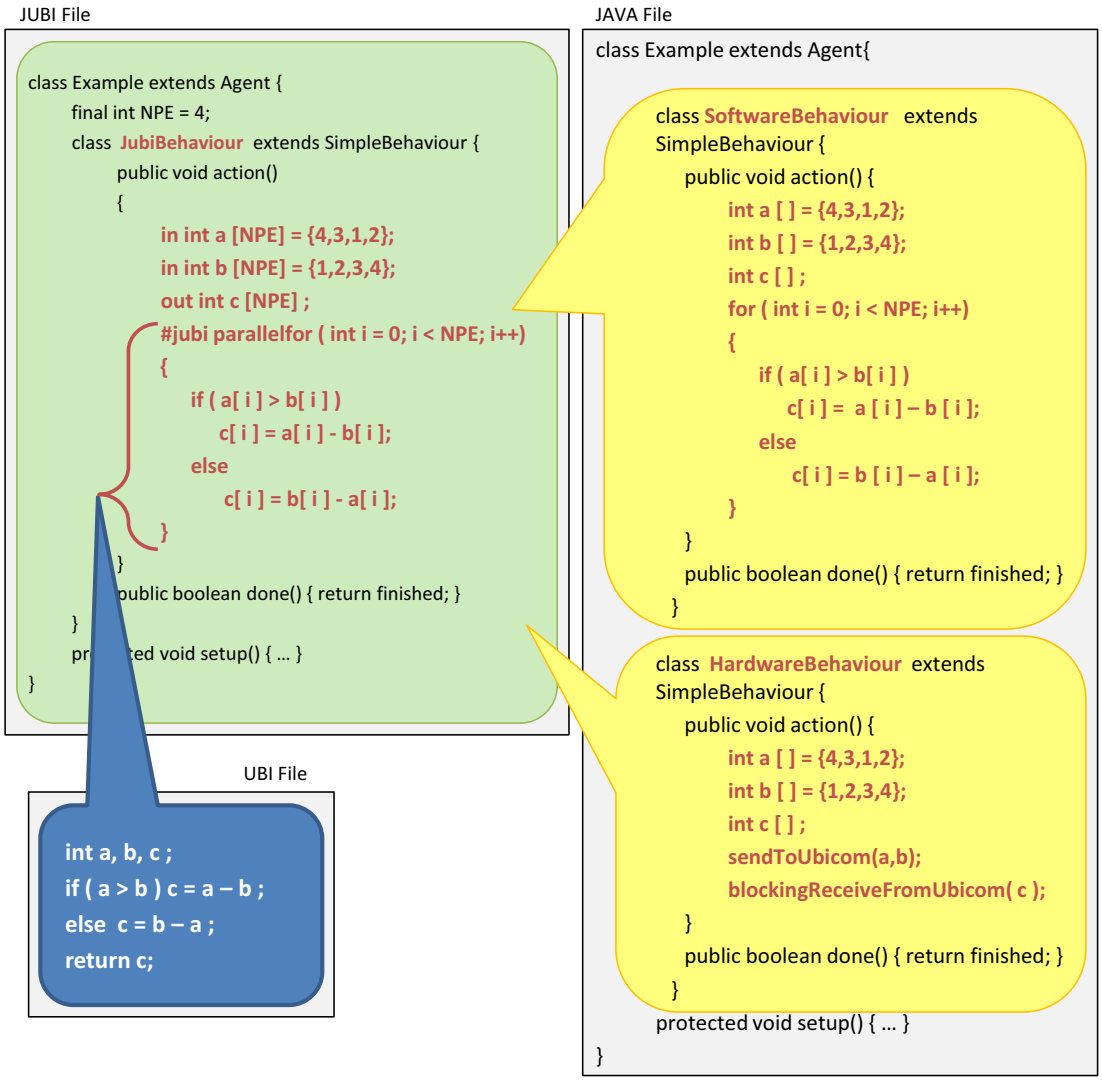

Figure 4.3: *Jubi splitting scheme*

In order to specify the portions of the code that will be executed on the Ubichip, a component view has been used. Thus, an accelerated section can be described with its inputs, outputs and internal behavior.

Adding this approach to Java requires to set firstly `in` and `out` keywords that will be considered respectively as input and output flags for inputs and outputs pins of the

component. In order to take profit of the potential parallelism of the Ubichip, a `NPE` keyword allows the user to specify the number of Processing Elements (PE) that will be used for this application. This `NPE` number must be defined as a constant. The following code gives an example of the transformations:

```
1  final int NPE = 5;
2  int a[NPE], b[NPE];
3  int c[NPE];
```

becomes:

```
1  final int NPE = 5;
2  in int a[NPE], b[NPE];
3  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. The following code gives an example:

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

Where `a`, `b` and `c` are arrays of scalars that will be distributed over the PEs is spanned over the PEs of the Ubichip architecture as scalar variables (in HW). For clarity reasons we name this kind of variable multi-scalars to differentiate them from vectors.

Finally to enable the parallelization of software sequential loops in the hardware accelerated mode we introduce the `parallelfor` keyword. This keyword allows to generate both software and hardware implementations from the same unified description.

```
1  #jubi
2  parallelfor(int i=0; i<NPE; i++)
3  {
4    c[i]= a[i] + b[i];
5  }
```

In this current version of the tool only declarations, assignments and integer arithmetic or logic (Boolean) expressions are supported inside the `#jubi` blocks. This avoids the use of classes and functions inside any hardware blocks.

To validate the proposed approach, JUBI tools - JubiSplitter and JubiCompiler - have been designed to adapt the JUBI methodology to the Perplexus Ubichip as it is used as case study platform. The following section gives more details about the SIMD mode of this accelerator that has been briefly presented in Section 2.5.1.

4.2 Perplexus accelerator: Ubichip

Any application description that complies with the previously presented workflow, may be run as HW/SW applications on the Perplexus platform thanks to the Ubichip accelerator and adapted tools. In the first part of this section, the runtime mechanism that allows *hybrid enabled* software agents to be accelerated is described. In a second part, the early Ubichip is presented in details. In the last part we present the modifications we performed to improve the Ubichip flexibility that led to the final description of this accelerator.

4.2.1 Kernel loading

During hybrid HW/SW simulations the Ubicom agent is used as a linker between hardware and software parts of the application. Figure 4.4 shows this link principle and the difference between purely software and hybrid executions. The left part of the figure represents the purely software execution where the complete agent is running on the GPP. This software agent implements its *SoftwareBehaviour* behavior that contains its entire functionality.

On the right side of Figure 4.4 the hybrid execution mode is used. The principle of this hybrid execution is to accelerate the application by using the Ubichip in SIMD mode to execute the kernel of the agent. This is achieved by running the *HardwareBehaviour* of the agent that therefore acts as an envelope in the software side of the Ubidule. This envelope provides an agent-to-agent interface that allows inter-agent communications even if the agent is partially executed on the accelerator. Therefore

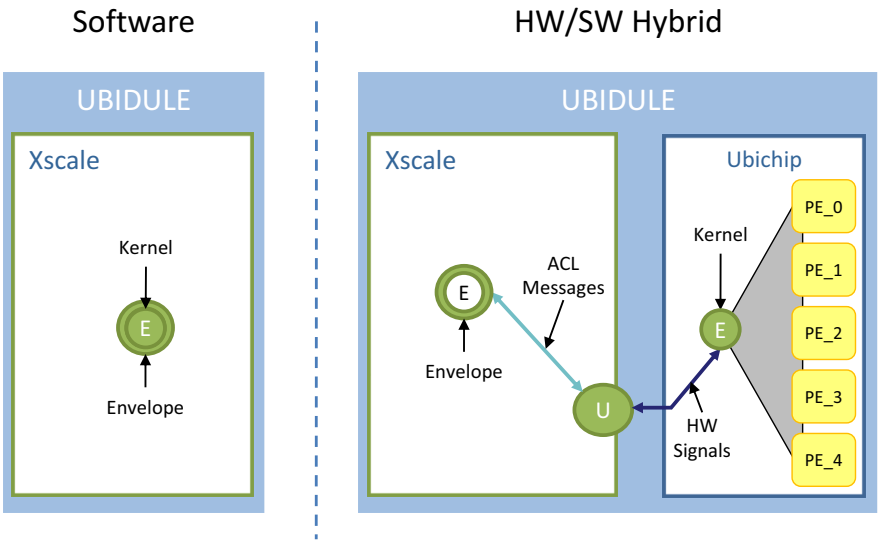

Figure 4.4: *Software and hybrid execution modes*

a dedicated communication link is established using the Ubicom agent as a wrapper between the envelope and kernels. This makes the Ubicom agent responsible of controlling the Ubichip from the BAF (c.f. Section 3.2.2.4).

The internal behavior of the Ubicom agent can be described as follow:

- Wait for a hardware execution request.
- Request processing:
  - Search for the corresponding kernel.
  - Kernel data update if necessary.
  - Kernel loading into Ubichip SRAM.
  - Set Ubichip to enable state.
- Wait for Ubichip end of program (thanks to an interrupt).
- Read results in Ubichip SRAM.
- Send results to the agent envelope.

As the Ubichip SIMD mode is single task, this behavior is blocking and following demands are treated in arrival order thanks to a task FIFO manager that interrupts the request processing of incoming tasks just before the data update phase. When the

previous request has been executed the processing is resumed and the Ubicom agent continues to operate with this policy until the end of the application.

Several hardware dependent characteristics have to be taken into account to adapt the proposed approach that is generic to particular hardware targets such as the Ubichip. For example:

- The single task aspect of the architecture.
- The SIMD scheme of the Ubichip.
- The Ubichip specific instruction set.

As a consequence, the following section provides more details on Ubichip SIMD characteristics.

4.2.2 SIMD operating mode

Within the Perplexus project, the development of a hardware accelerator able to deal with bio-inspired features was one of the main objectives. Therefore the so called Ubichip has been developed to support and accelerate the execution of heavy applications at the Ubidule level. The Spiking Neural Network (SNN) application proposed as a case study in the project has been used as a base for the Ubichip development [122]. Research led at UPC on this point concluded that the SIMD architecture would offer the desired capabilities to accelerate the SNN application. Actually, this kind of architecture allows to simulate a maximum number of neurons in parallel under the constraints of academic funding possibilities (coarse technology and reduced size). As generic SIMD processors the Ubichip is build around an array of PEs controlled by a unique Reduced Instruction Set Computer (RISC) instruction sequencer [126]. Figure 4.5 shows the major features that compose this SIMD architecture embedded in the Ubichip.

As previously presented in Section 2.5.1 Ubicells include a 4 bits ALU and four 4-inputs LUTs. Four of these elementary cells are grouped into Macrocells that therefore provide 16 bits PEs. The current version of the Ubichip is made with a 10x10 matrix of PEs. Inside these PEs no specific register nor stack is available. The architecture

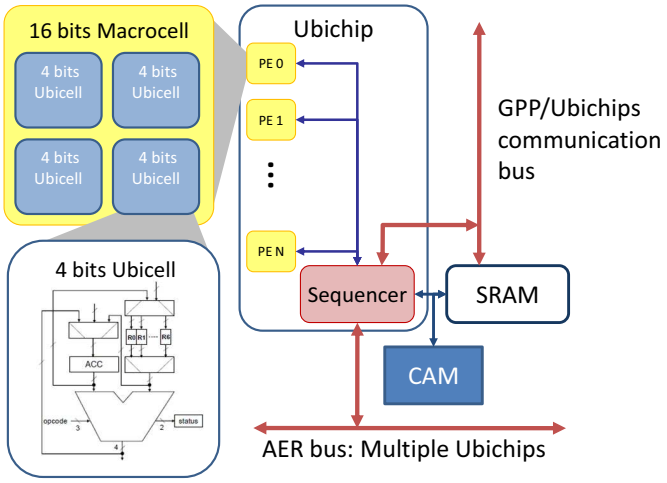

Figure 4.5: *Ubichip configured in SIMD processor*

is therefore register-based. The sequencer proposed in [126] is the unique controller of this PE array (i.e. this actually makes the architecture an SIMD processor). The sequencer also integrates a 64 bits Linear feedback shift register (LFSR), that is used to efficiently generate random numbers, and interfaces the Ubichip with external elements such as:

- The Ubidule GPP.
- The program and data memory.
- The Content-Addressable Memory (CAM) controller connected with an Address Event Representation (AER) bus for inter-Ubichip communications [125, 116].

As presented briefly in Section 4.2.1 the Ubichip is connected to the GPP that controls it. When used in SIMD mode this simple control is based on a bidirectional communication, an *enable* signal activate the Ubichip execution. When the execution ends an *interrupt* is sent to the GPP notifying it that results are available.

The memory access is shared between the GPP and the Ubichip allowing data transfers between these two devices. The Ubichip memory controller manages transactions between the Ubichip and the external SRAM in a Von Neumann fashion, without any internal cache memory. Four 8 bits instructions or two 16 bits data are read from the 32 bits memory at once. As an example, 50 transactions are required to load or store a multi-scalar value on the 100 PEs of the SIMD.

The SRAM memory is continuously accessed during and after the program execution. As a matter of fact the SNN application runs step-by-step simulation and requires setting up networks of important size and inter-neuron connectivity. For example targeted network involve 10000 *neurons* with 300 *synaptic connections* per neuron. Such networks exceed the capability of a single Ubichip; they are therefore split onto several chips. To connect neurons hosted onto different chips a CAM is used to update neuron input signals between two simulations time steps. Figures 4.6 represents this multiple Ubichip communication based on the association of CAM and AER. The AER is used to transmit data with a minimal delay between all connected chips. A token ring - with time multiplexing - communication allows each Ubichip to send addresses of activated neurons and receive other Ubichips' results.

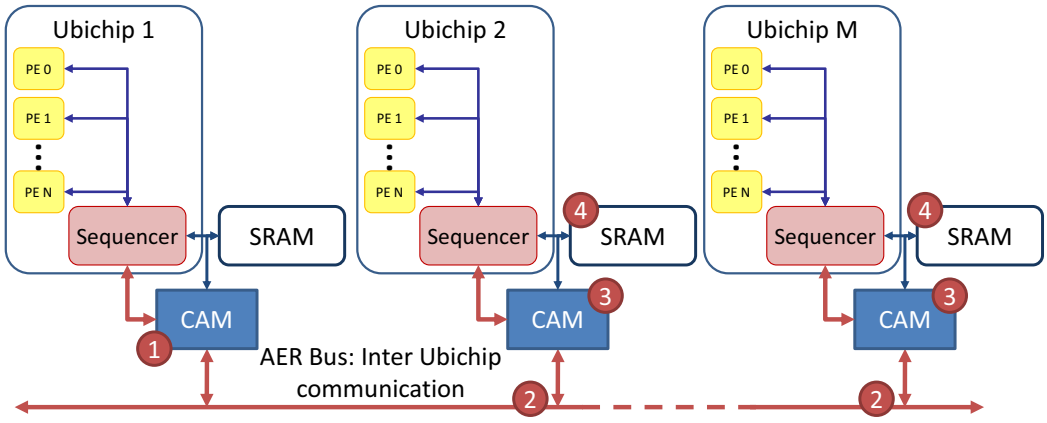

Figure 4.6: *CAM updates neurons' input signals depending on data received via the AER bus*

Figure 4.6 shows an example of behavior that occurs when the AER communication is triggered. The CAM controller of the Ubichip 1 gathers the addresses of activated (or spiking) neurons (step 1) and sends them on the AER bus (step 2). CAM controllers of other Ubichips receive these addresses and decode them (step 3). If local neurons are connected to an external spiking one their input signal is updated inside the SRAM (step 4). This principle is repeated for every Ubichip allowing a full connectivity of the neural network over several Ubichips. As CAM and AER are included in the final Ubidule board we were not able to test this feature at the time of writing this document.

Several instructions are used to load, store data or activate the CAM communication, these control instructions that operate inside the Ubichip sequencer are described in the following section.

4.2.2.1 Control instructions

The Ubichip sequencer has been designed to provide direct access to data contained in the SRAM memory.

- "`LDALL D, reg`", loads a given data `D` in the register `reg` of all PEs. This instruction is used to load constant data in all PEs.

- "`LOAD M, reg`", loads a given multi-scalar value `M` in the register `reg` of PEs. This instruction is used to load PE dependent values in their respective PE.

- "`STC M, reg`", "`STNC M, reg`", "`STZ M, reg`" and "`STNZ M, reg`" are used to store the content of registers `reg` in the multi-scalar `M` with different conditions. Conditions are PE dependent and based on the ALU Zero and Carry flags. `C` and `NC` represent conditions based on the Carry flag (respectively 1 or 0). In the same way `Z` and `NZ` represent conditions on the Zero flag (respectively 1 or 0).

- "`SETZ`", "`SETC`", "`CLRZ`" and "`CLRC`" instructions allow overwriting the current Zero and Carry flags of all PEs with a fixed value: 1 for `SET` instructions and 0 for `CLR` instructions. These instructions are useful to provide unconditional stores that are not implemented.

- "`LOOP n`" and "`ENDL`" instructions are used to create loops with a fixed size of `n` in a program. Macro instructions presented in the following section use these instructions.

- "`GOTO L`" instruction provides an unconditional jump. When executed the sequencer reads the program address corresponding to the label `L`.

- "`HALT`" and "`RET`" instructions are used at the end of programs to respectively trigger the CAM mode or the GPP interruption.

- "`RANDINI s`", "`RANDON`" and "`RANDOFF`" instructions are used to trigger the Random Number Generator (RNG) included in the sequencer. Between `RANDON` and `RANDOFF` instructions every call to `LOAD` and `LDALL` instructions are using the number generated using a 64 bits LFSR - used as RNG - instead of reading memory content. The RNG is initialized thanks to the `RANDINI` instruction that loads the seed `s` in the LFSR.

The SIMD policy imposes a systematic exploration of all the branches of an algorithm. In other words, when solving a conditional statement, each created branch must be executed by the concerned PEs and only by those concerned. To allow operations on a set of PEs the chosen approach is based on a write protection of registers. All PEs continue to execute the same instructions in the same time but only concerned PEs have the possibility to change any value of their registers. Other PEs are considered to be *frozen*. To trigger this particular behavior the `FREEZE` instruction family has been proposed.

- "`FREEZEC`", "`FREEZENC`", "`FREEZEZ`" and "`FREEZENZ`" instructions are used to *freeze* PEs that are respectively in Carry, No Carry, Zero or No Zero states.
- "`UNFREEZE`" instruction is used to resume the *freeze* state of PEs. 256 *freeze* levels are available in the final version of the Ubichip sequencer allowing the resolution of complex conditions, nested tests or nested loops for example.

Such conditions may be used to resolve conditional tests and therefore allow the required systematic exploration of the entire algorithm. Thanks to these instructions the Ubichip sequencer is able to control data and the PE array. Additional instructions are executed inside PEs. The following instructions complete the instruction set and provide operative mechanisms. As a consequence of the SIMD when such instructions are used the sequencer triggers all PEs ALU to execute them.

4.2.2.2 ALU instructions

In this section elementary ALU instructions and macro instructions are briefly presented.

- "`MOVA reg`" and "`MOVR reg`" instructions are respectively used to write a value from the register `reg` to the accumulator and vice versa.
- "`ADD reg`" and "`SUB reg`" instructions are used to ad and respectively subtract the value content in the accumulator with one in register `reg`

- "`SHR`" and "`SHL`" instructions are used to shift to the right and respectively to the left the value contained in the accumulator.

- "`AND reg`", "`OR reg`" and "`XOR reg`" instructions are used to apply associated bitwise logical operations between the accumulator and the register `reg`. Then the accumulator receives the result of this logical operation.

- "`INV reg`" instruction is used to load the accumulator with the complement of the value contained in the register `reg`.

- "`MOVTS reg`", "`MOVFS reg`" and "`SWAP reg`" instructions provide inter-register bank communications. `MOVTS` writes the value contained in the register `reg` in the associated shadowed register. Respectively `MOVFS` writes in the register `reg` the content of its shadowed register. `SWAP` switches the contents of registers `reg` and its shadowed register.

4.2.2.3 Architecture limitations

During the JUBI specification and JubiCompiler development restrictions have been encountered with the Ubichip hardware. The first SIMD sequencer was not able to deal with some required features listed below.

1. *Indexed addressing* avoiding the opportunity to use data vectors and arrays.
2. *General conditional jumps* preventing the description of programs with iterative statements as `while` and `for` structures.
3. *Conditional Memory writes* causing unwished memory writings.

To solve these limitations, consists in making the Ubichip Turing complete; which allows it to be able to execute any general purpose program in its SIMD paradigm. Therefore several modifications have been proposed to solve the above mentioned limitations. These modifications are exposed in the following section.

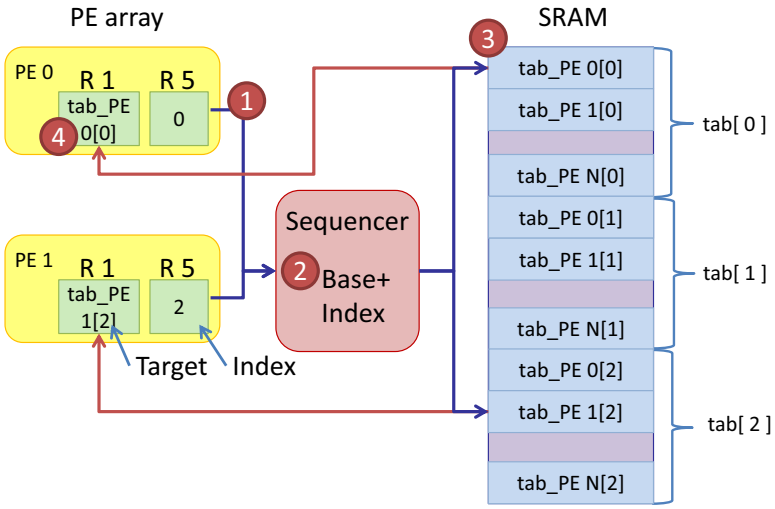

Figure 4.7: *Ubichip SIMD indexed addressing*

4.2.3 Ubichip modifications

Some Ubichip modifications have been proposed at the sequencer level to overcome the presented restrictions. A memory indexed addressing instruction has been added and operates as presented in Figure 4.7.

For each PE the sequencer:

1. Reads the index value contained in the specified index register.
2. Sets memory pointer to $base + index$.
3. Reads memory content at the final address.
4. Stores read value in the target register.

Thanks to this mechanism every PE is able to operat on data vectors at independent indexes. Therefore PEs can operate on independent data with different indexes that suit their own execution context. This improves the flexibility of the SIMD architecture as index value may change from one PE to another.

To deal with the iterative statements issue, we proposed to use the PE freeze state as the ending condition. Therefore, a *global freeze* signal is raised when all PEs are frozen. The `GOTOF` instruction has been included to the instruction set, and thanks to the *global freeze* signal, it provides a *globally conditional* jump function. Such global conditional jump is required to enable iterative statements processing in the SIMD paradigm. The `GOTOF` instruction mechanism is illustrated in Figure 4.8 when used to end a `for` loop statement.

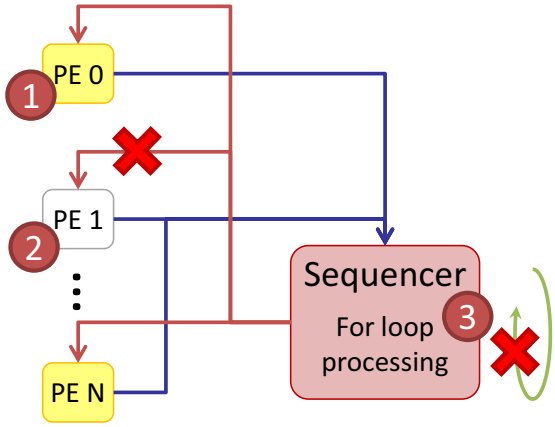

Figure 4.8: *Ubichip conditional memory writing*

1. The sequencer feeds statement instructions in the PE array and gets the different PE freeze state.
2. A PE switches to freeze when its own loop is processed. At every loop test the `GOTOF` instruction is processed with a false condition.
3. When all PEs are in freeze state, the `GOTOF` instruction condition is turned to true and the jump is finally processed ending the loop iterations.

The last limitation we found with the initial Ubichip sequencer was data inconsistency that came from the unprotected memory writing policy. For example when a conditional test occurs in the program both *true* and *false* branches are processed (cf section 4.2.2.1). In the case data have to be written in the memory inside one particular branch, only the concerned PEs must write their data contrarily to frozen PEs that must not. As a matter of fact, data values are not coherent with the context of the PE. In the first version of the sequencer the freeze state was only protecting frozen PE registers from external writings. But these PEs registers were read when a `STORE` instruction occurs. This policy results in the storage of inconsistent data in the memory. Figure 4.9 shows the write protection we proposed to avoid such data inconsistency.

When the write protection on freeze state is activated and a `STORE` instruction occurs two cases can be encountered. In the first case (1), the PE is active and value is stored as usual. In the second case (2), the PE is frozen and its data must not be stored in the memory the sequencer therefore skips the memory writing for this PE.

Such modifications were required for the Ubichip to offer interesting parallel execution capabilities. Presented modifications have been validated thanks to test programs presented in Section 4.3.5 and included in the final version of the Ubichip SIMD sequencer.

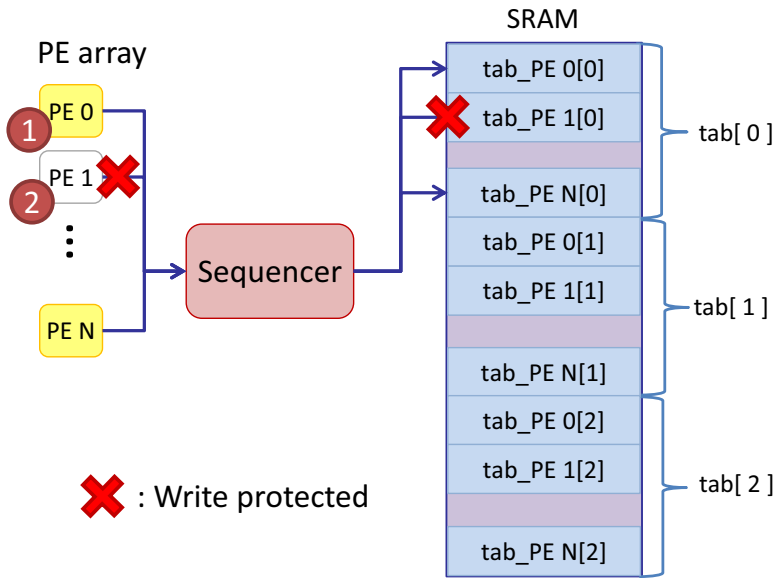

Figure 4.9: *Ubichip conditional memory writing*

4.3 Programming tools and validation

With these issues solved the JUBI language was flexible enough to offer the interesting opportunity to describe application either for SW or hybrid executions. As shown in Figure 4.2 this is achieved thanks to dedicated tool the following sections describe.

4.3.1 JubiSplitter

The entry point of the JubiTool compiling environment is the JubiSplitter that splits the JUBI description into a Java description for the software parts and several Ubi descriptions for the hardware accelerated parts. In order to make Input file fully Java compliant a cleaning feature of the JubiSplitter tool removes JUBI specific keywords.

The JubiSplitter generates two Java classes named `softwareBehaviour` and `hardwareBehaviour`. These JADE behavior classes represent respectively the whole agent functionality in SW mode simulations 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 4.3 gives an example of this code splitting step that is the first stage of application compiling process.

4.3.2 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:

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

is then compiled into:

```
1   .data
2       V01="00030004","00020001"
3       V02="00020001","00040003"
4       V03="00000000","00000000"
5   .code
6       load  r1,V01
7       load  r2,V02
8       mova  r1
9       add   r2
10      movr  r3
11      store r3,V03
```

JubiCompiler is based on a flex [119] and bison [118] description of the Ubi grammar and is a SIMD oriented compiler [120]. `NPE` indicates that every Processing Element of the SIMD processor will receive a different data set. 1-D arrays therefore become PE-local scalar variables. In this way a 1-D variable of NPE elements (in SW) is spanned over the PEs of the Ubichip architecture as scalar variables (in HW), for clarity reasons we name them multi-scalar to differentiate them from vectors. As an example the code provided in the figure 4.3 becomes:

```
1   .data
2     C0="00000000"
3     CM1="FFFFFFFF"
4     C1="00010001"
5     V01="00030004","00020001"
6     V02="00020001","00040003"
7     V03="00000000","00000000"
```

```
8           .code
9             load   r1,V01   ;       data loading
10            load   r2,V02   ;
11            mova   r2       ;
12            ldall  r2,CM1   ;       test condition
13            sub    r1       ;
14            freezec         ;
15            ldall  r1,C0    ;
16            unfreeze        ;
17            mova   r1       ;
18            freezez         ;       if true
19            load   r2,V01   ;       data loading
20            load   r3,V02   ;
21            mova   r2       ;
22            sub    r3       ;       subtraction
23            movr   r2       ;
24            setz            ;
25            stz    r2,V03   ;       data storage
26            unfreeze        ;
27            mova   r1       ;
28            freezenz        ;       else
29            load   r2,V02   ;       data loading
30            load   r3,V01   ;
31            mova   r2       ;
32            sub    r3       ;       subtraction
33            movr   r2       ;
34            setz            ;
35            stz    r2,V03   ;       data storage
36            unfreeze        ;
```

Variables a,b and c are here multi-scalar variables respectively called V01, V02 and V03. Constants 0, 1 and -1 are defined in every programs as they are used for tests and increments for example true is coded -1 (FFFF in 16 bits hexadecimal) and false 0 (0000 in 16 bits hexadecimal).

As exposed previously the *freeze* state and associated instructions are used to allow conditional statements execution (see lines 14, 16, 18, 26, 28 and 36).

This example also shows that the current register management policy is not optimal as variables V01 and V02 are loaded several time (see lines 19, 20 and 29, 30).

4.3.3 Code Analyzer

In order to gather data on the compiled program a basic code analyzer has been integrated within the JubiCompiler. Thanks to the SIMD architecture of the Ubichip and its clock cycle accurate memory transactions (no cache memory), the analyzer is able to provide a clock cycle accurate timing of the compiled program.

Analyzer results are stored in a Comma-Separated Values (CSV) file that can be read by spreadsheet utilities. User is then asked to modify unknown parameters such as parameterized loop iterations to be able to know the number of clock cycles required for the program to end. This step may also be used to extrapolate application timings with different parameters at a glance.

Therefore it provides an interesting opportunity to investigate the application parameter space. During the JubiCompiler development and testing as well as for Ubichip model testing we used this functionality to predefine the amount of necessary simulation time steps. This can be used to see the parameter impact on the application acceleration without running numerous and time costing VHDL simulations.

4.3.4 UbiAssembler

The UbiAssembler is the final tool used in the flow. It is in charge of translating the JubiCompiler generated assembly code into the Ubichip SIMD hardware language. This tool is part of the Spiking Neuron Development Kit (SpiNDeK) environment [121] made at the Department of Electronic Engineering of the Universitat Politècnica de Catalunya and was developed to program the Ubichip using assembly.

As a consequence JubiSplitter and JubiCompiler have been designed with UbiAssembler compatibility in mind. Two main features are provided in this tool in addition to the code translation: the memory layout setup that involves variable symbolic naming translation to physical address, and jump linking provided by label to physical address translation.

The SpiNDeK environment is able to create a VHDL memory file filled with the assembly code. This file is used as program memory in VHDL RTL simulations (using Mentor graphics Modelsim). These simulations therefore provide an environment to

check Ubichip behavior and generated code reliability. Therefore these simulations were used to check that generated codes provide functional Ubichip programs.

During this Perplexus project the Ubichip has been developed and fabricated but at the time of writing no experiments have been possible on the final Ubidule equipped with at least one Ubichip. Therefore functional tests based on VHDL simulations have been done to check the compilation flow functionality.

4.3.5 Functional tests

During the joined development of the Ubichip and the JubiTool workflow seven test groups were used to check the functionality of the generated code on the Ubichip. This section presents the functionality coverage of the proposed test groups. Principal tests are presented in Annex C.

1. Test group 1 corresponds to arithmetical expression tests.
2. Test group 2 corresponds to logical expression tests.
3. Test group 3 corresponds to simple iterative expression tests.
4. Test group 4 corresponds to advanced Iterative and logical expression tests.
5. Test group 5 corresponds to simple vector manipulation tests (i.e. indexed addressing tests).
6. Test group 6 corresponds to advanced vector manipulation tests.
7. Test group 7 corresponds to random number generation tests.

The main test of every group has been extracted from one of the Perplexus applications: the biologically plausible neural network simulator using the model presented in [117]. The compiled code has been assembled and simulated using UbiAssembler and Modelsim VHDL simulations. Results of these simulations were then compared with intended results to validate the code functionality.

Simulations were targeting a 2x2 matrix of Ubichip Processing Elements and last from seconds to tenth of minutes depending on the simulated code length. Simulations on the complete Ubichip were too long to be useful for functionality checking. Therefore to know the actual number of clock cycles on the 10x10 PE matrix of the

Ubidule analyzer extrapolation was used to provide information about the duration of execution of codes on the complete Ubichip.

After functional validation of the generated code the SNN simulator application has been translated to JUBI and compiled as described previously. This application has been chosen to bench the proposed solution and compare it with the pure embedded software solution as well as a generic CPU solution and a CPU+GP-GPU solution. The following section therefore presents this SNN implementation compared to these three other architectures.

4.4 Experimental results

The SNN simulator application has been used as a base for the Ubichip conception; it is therefore obvious to use this application to bench the proposed solution. The following section focus on the experimental protocol we used to conduct these benchmarks. Section 4.4.2 and 4.4.3 present both hardware and software sides of the GP-GPU solution we chose to include in this comparison.

4.4.1 SNN simulator algorithm

The model of this SNN is introduced in [117], Figure 4.10 summarizes the part of this simulator we are focusing on. The dark blue part represents the kernel function of this algorithm; the agent envelope is then represented in the light green parts.

Sequential executions with 100 neurons spend 95% of the computation time in the parallelizable section with neurons possessing 300 synaptic connections. Four hardware targets grouped according to their respective computing areas have been used for these experiments.

For the Ubiquitous computing area:

1 The Ubidule XScale processor that gives a single core processor reference using the software execution mode.

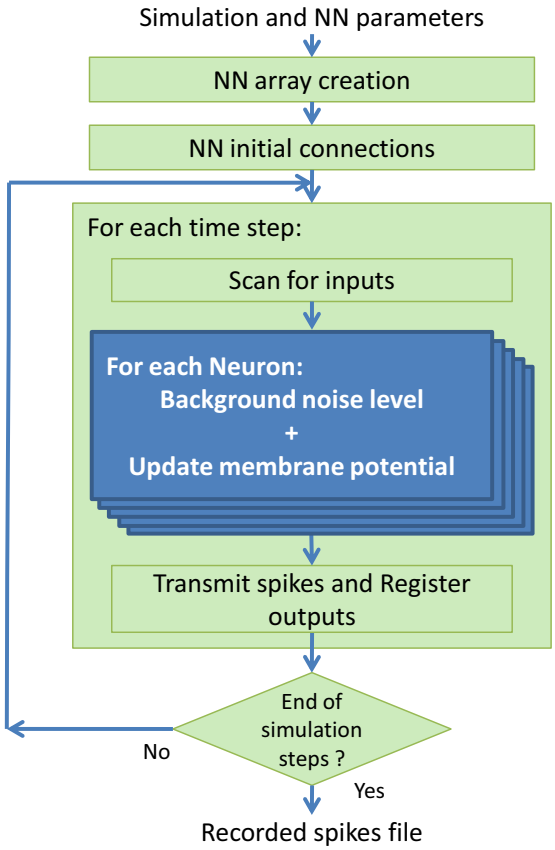

Figure 4.10: *SNN simplified algorithm*

2 The Ubichip SIMD code simulation. This target provides an idea of the improvement on application acceleration. To give an industrial aspect to the Ubichip we consider its working frequency to be equal to the XScale frequency that is 520MHz.

For the Personal computing area:

3 An Intel Xeon single core processor working at 3GHz. This sequential processor is used to figure out what can be the application performance on classical PC.

4 A Intel Xeon single core processor working at 3GHz endowed with an nVidia GTX280 board presented in section 4.4.2.

To introduce this benchmark presentation, the following sections presents the used GPU architecture. The GP-GPU solution has been chosen because it is an increasingly popular parallel computing solution with a programming philosophy close to our proposal.

4.4.2 GP-GPU solution: nVidia architecture

Our experiments were done on a workstation made with a 3GHz Xeon processor and two different nVidia graphical boards. Benched GPUs are different in terms of number of PEs. In Figures 4.11 and 4.12, nVidia G105M and GTX280 architectures are briefly introduced. Figure 4.11 also focus on the Texture Processing Cluster (TPC) design.

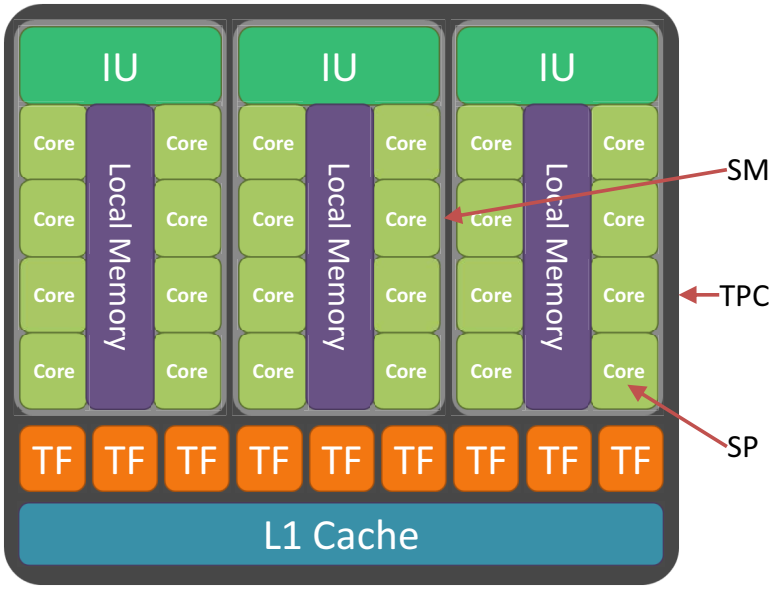

Figure 4.11: *nVidia Texture Processing Cluster architecture*

The TPC processes texture when the GPU is used in 3D scene rendering mode. When used in GP-GPU it is a cluster of 24 computing cores or Streaming Processors (SP) sharing one cache memory and 8 specific units. These 24 computing cores are divided in 3 groups of 8 cores that share a local memory. These blocs are called Stream Multiprocessors (SM). The G105M GPU is built around 1 TPC whereas the GTX280 is composed with 10 TPC achieving a total of 240 computing cores, as presented in Figure 4.12.

A physical scheduler called *Thread Scheduler* is located on the top of this architecture to distribute application threads on the computing resources. This is done without any noticeable computing load.

The SM is based on a Single Instruction Multiple Thread (SIMT) architecture. The SIMT principle is an evolution of the SIMD architecture where the vectorization is not made at data level but at the context level. Based on a pipelined architecture with an

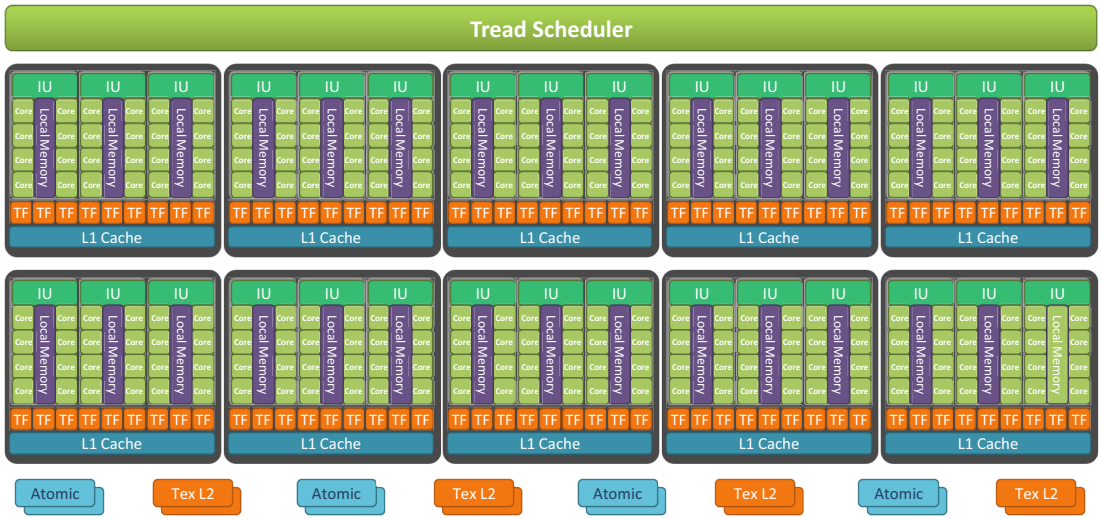

Figure 4.12: *nVidia GTX280 architecture overview*

instant thread switching mechanism, a single SP is able to run 128 threads simultaneously without any noticeable computation overhead thanks to an instant thread context switching. At the SM level 1024 threads can be processed at a time corresponding to a total of 32720 threads for the whole GTX280 chip. The main reason of this thread multiplication is that it offers more possibility to span an application over the architecture in a regular manner.

An easy observation is that such a massively parallel architecture is almost impossible to program using assembly code and therefore requires a programming environment. The same holds true for other equivalent solutions like the Ubichip, and the same apply to reconfigurable devices descriptions. The following section discuses the problem of implementing an application on a system using a hardware accelerator.

4.4.3 Compiling applications for GP-GPU

For the conducted experiments we were using the CUDA programming environment developed by nVidia. This tool is intended to ease the porting of parallelizable applications to GPU massively parallel architectures as well as multiple core processors. CUDA is available for Windows, Linux and MacOS operating systems. The NVCC compiler is the base of CUDA compilation flow as presented in Figure 4.13. This tool splits the parallel kernels from sequential part of the application to provide two types of output:

1. The CPU code that is composed of the sequential parts and instructions to interact with the graphical board system driver such as copy from main memory to board memory and code execution commands. This Code is compiled with an integrated GCC compiler to be executed on the CPU.
2. The application Parallel Thread eXecution (PTX) byte code that is used to create the low level code executed by the GPU.

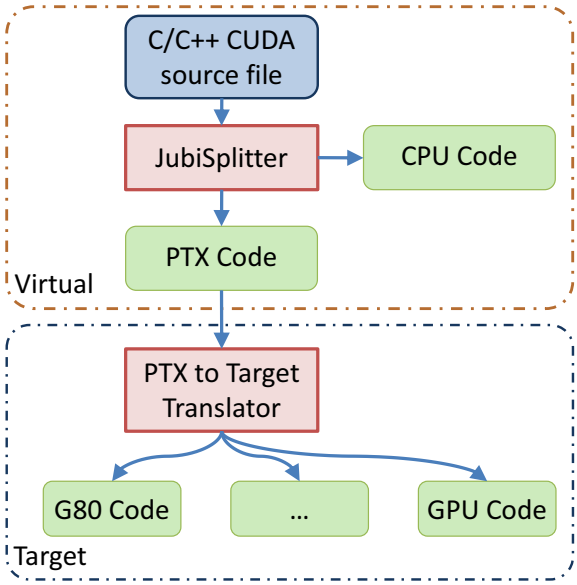

Figure 4.13: *nVidia CUDA Compilation flow overview*

The PTX to Target translator integrated in the nVidia system driver is then able to map the PTX virtual code on the system graphical board. Therefore it is then possible to target the different generations of nVidia boards whatever the model is with one PTX code.

The experiments on CUDA show that CUDA programming environment is accessible, provides an efficient methodology to program parallel applications and offers interesting opportunities to accelerate application executions as shown in Section 4.4. This may explain the GP-GPU current usage growth.

We therefore integrated the presented GP-GPU solutions into the proposed benchmark as parallel accelerators for personal computing solution. The following sections present these benchmarks protocol and results.

4.4.4 Experimental results

In order to compare the different software and hardware accelerated executions, the execution time of the parallel part on architectures 1, 3 and 4 have been measured, for the Ubichip part a cycle accurate simulation has been used. No communications between GPPs and accelerators have been taken into account due to the unavailability of the Ubichip. The spiking neural network code is compiled with the Java compiler for the XScale, with our JubiCompiler for the Ubichip finally CUDA has been used to compile the code for both the Xeon processor implementations with and without the GPU acceleration.

The "per neuron per time step" computation time has been extracted from original data to provide a relevant metric. The interest of this metric lies in its capability of assessing with the different tested platform and that it allows knowing the amount of neuron that can be computed with real time constraints.

Table 1 gives for each hardware target the resulting computation time per simulated neuron for simulation time step of 1 ms. For this test, the frequency of the Ubichip has been set to the frequency of the XScale, this allows to see the impact of the use of the SNN parallelism.

#	Architecture	Time per neuron	RT neurons
1	XScale (1 core @520MHz)	540μs	$<$2
2	Ubichip (100 datapath SIMD @520MHz)	87μs	$\approx$11
3	Xeon (1 core @3GHz)	64μs	$\approx$15
4	GTX280 (240 SIMT cores @602MHz)	0.4μs	$\approx$2500

Table 4.1: *Per neuron computation time comparison*

4.4.5 Results interpretation

Based on the results of the Table 4.1 we can interpret and draw some conclusions about the Spiking Neural Network implementations and the proposed approach.

4.4.5.1 GPU driven acceleration

In the case of Xeon and Xeon + GPU implementations the obtained results clearly demonstrate the interest of the GP-GPU solutions for applications like this SNN simulator. Further investigations showed that the observed speedup between CPU and GPU implementations for this particular application ranges from 37 to 164 as shown in Figure 4.14.

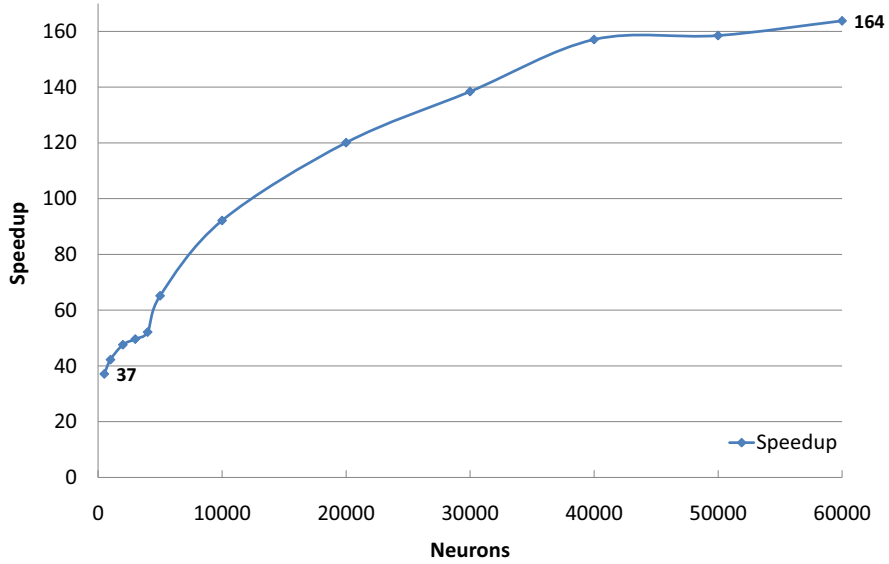

Figure 4.14: *GPU versus CPU implementations speedup*

These experiments also showed that the CPU execution time is increasing as a piecewise linear function. As shown in Figure 4.15 this evolution is constant and fast. We interpret this by the increasing amount of cache miss that occurs when the number of neuron increases.

In the GPU execution time case the increase is linear and evolves slowly for all simulations. There is no cache miss problem with this architecture therefore it provides linear performance for this particular SNN application.

4.4.5.2 Architectures comparison

Figure 4.17 presents some direct comparisons between all benched architectures depending on the number of used computing elements. XScale and Xeon are efficient architectures with only one computing element therefore the mean computing clock

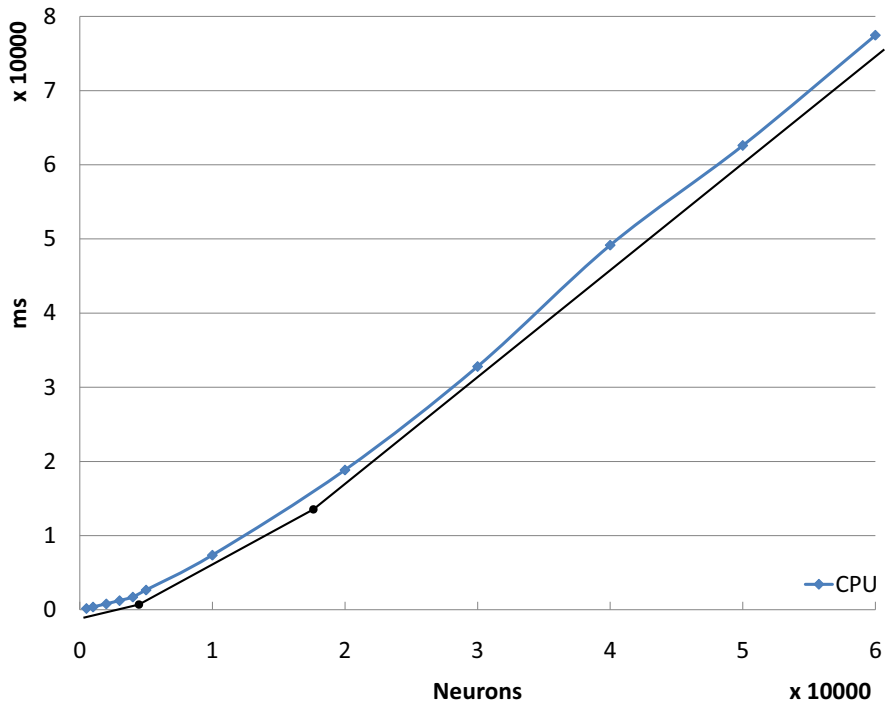

Figure 4.15: *CPU execution time*

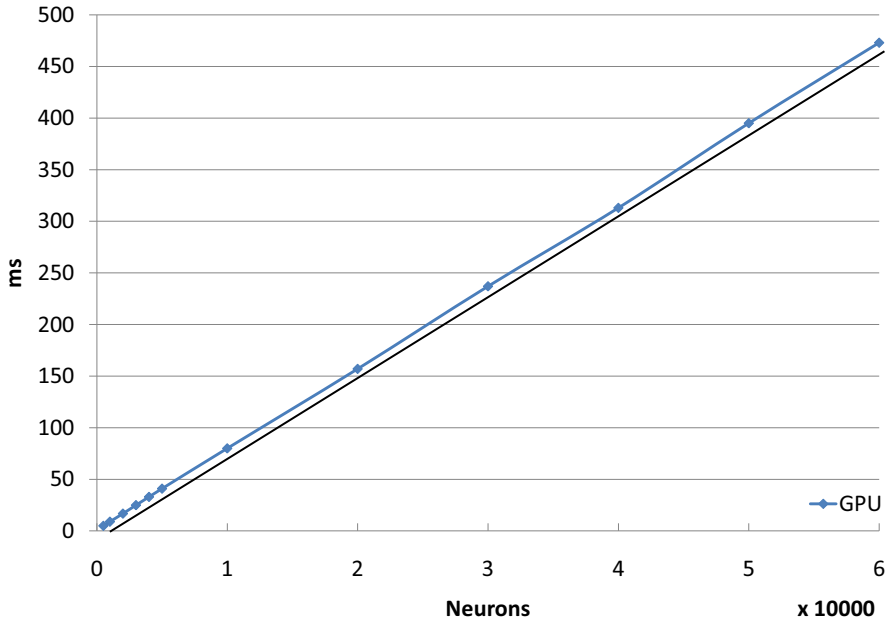

Figure 4.16: *GPU execution time*

cycles for one neuron is quite high compared to GPUs with 24 and 240 smaller processing elements. The Ubichip particular architecture proves to be an efficient solution if 10 or more processing elements are used.

Nevertheless, Ubichip performance are obviously significantly lower than GPUs'.

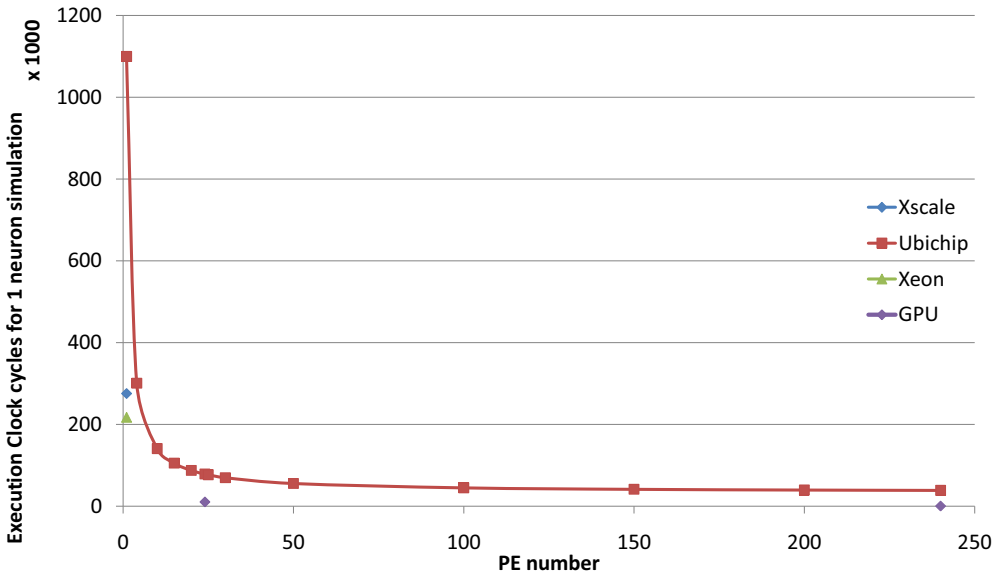

Figure 4.17: *Compared clock cycles to compute 1 neuron*

Technological and mean reasons can explain this difference in performance. The Ubichip has been build for embedded devices with an old CMOS 180nm process on a 50mm^2 silicon die. Concerning the GTX 280 GPU the 55μm process has been used on a 576mm^{2}silicon die. Additionally these result have to be carefully used as they do not take into account any communication between a GPP and the associated accelerator. For this reason we now give some preliminary conclusions that focus on the impact of using the Ubichip in the presented context. These conclusions as the one presented above should be reconsidered after complete tests including communication costs.

4.4.5.3 Ubichip impact

In the case of the XScale and Ubichip implementations results the acceleration is not as obvious as in the previous case. Nevertheless logical explanation can be found to interpret this result. The Ubichip architecture built around two operating modes does not offer the intended performance because of the inherent complexity of having both FPGA and SIMD in the very same logical substrate. Routing may be involved as well as the lack of hardwired multipliers inside SIMD PEs. The memory interface and management also impacts performance as a limited bandwidth does not allow offering rapidly data required by 100 PEs to achieve the computation.

The JubiCompiler is also involved in the low performance of the Ubichip SIMD mode. The actual register management is not optimized enough and induces more load and store instruction than necessary. The lake of an optimized register management coupled with the memory bandwidth problem causes a consequent reduction of performance in data intensive applications, such as the presented SNN simulator.

With the identified issues it is therefore possible thanks to the code analyzer to provide some prospective results on the possible improvement on the SIMD mode of the Ubichip and its compiler.

4.4.5.4 Performance improvements

We have discus three possible improvements for a possible future version of the Ubichip. The first proposal is to integrate hardwired multipliers that are widely used for applications like the SNN simulator. The second proposal is to improve the JubiCompiler register management with a more efficient register management. The third and last optimization is to improve the memory interface in order to increase the bandwidth and therefore reduce the data access latency.

Based on the SNN application these possible improvements have been investigated in order to see their possible impact on the Ubichip performance. In the current section we present the prospective results concerning five possible improved Ubichip architectures and code compiling solutions:

1. An Ubichip endowed with hardware multipliers.
2. An Ubichip executing a hand optimized register management.
3. An Ubichip that supports a memory with a doubled bandwidth (requires modifications at the memory controller level).
4. An Ubichip endowed with multipliers and that executes the optimized code. This solution allows observing the combined impact of using multipliers with a reduced number of memory accesses.
5. Finally an Ubichip endowed with multipliers, the enhanced bandwidth that executes the optimized code. This final solution takes advantage of the three basic optimizations.

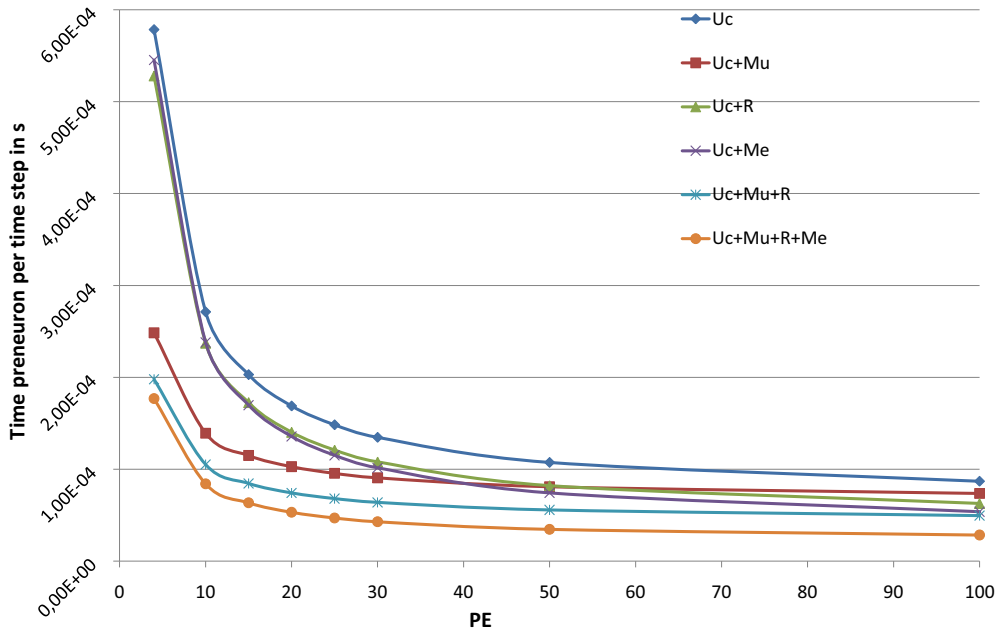

Figure 4.18: *Ubichip possible evolutions performance*

These solutions are compared to the current solution on the SNN simulator and use from 4 to 100 PEs as presented in Figure 4.18; for this test neurons are simulated with 300 synaptic connections each.

The graph of Figure 4.18 exhibits results linked with this particular application. In this case the multiplier impact is really noticeable for small amount of PEs reducing execution time with a factor around 2.3 for a 4 PEs example and only around 1.2 in a 100 PEs example. On the contrary the impact of both memory optimizations is more and more noticeable as the number of PE increases. The code optimization therefore offers a speedup around 1.4 in the 100 PEs case when a ratio of 1.1 is only possible when 4 PEs are used. The same happens with the memory bandwidth enhancement with ratio of 1.6 and 1.1 respectively.

When combining the multiplier endowed architecture and the code optimization, the speedup ratio increases to 2.9 and 1.8 respectively with 4 and 100 PEs. The graph also demonstrates that combining all optimizations results in improving the system performance by a factor of 3.1 approximately.

Figure 4.19 shows a performance comparison between the proposed Ubichip architectures and code optimizations when the number of synaptic connections is increased. These figures clearly show that the proposed improvements may benefit to the Ubichip and should be investigated in the future.

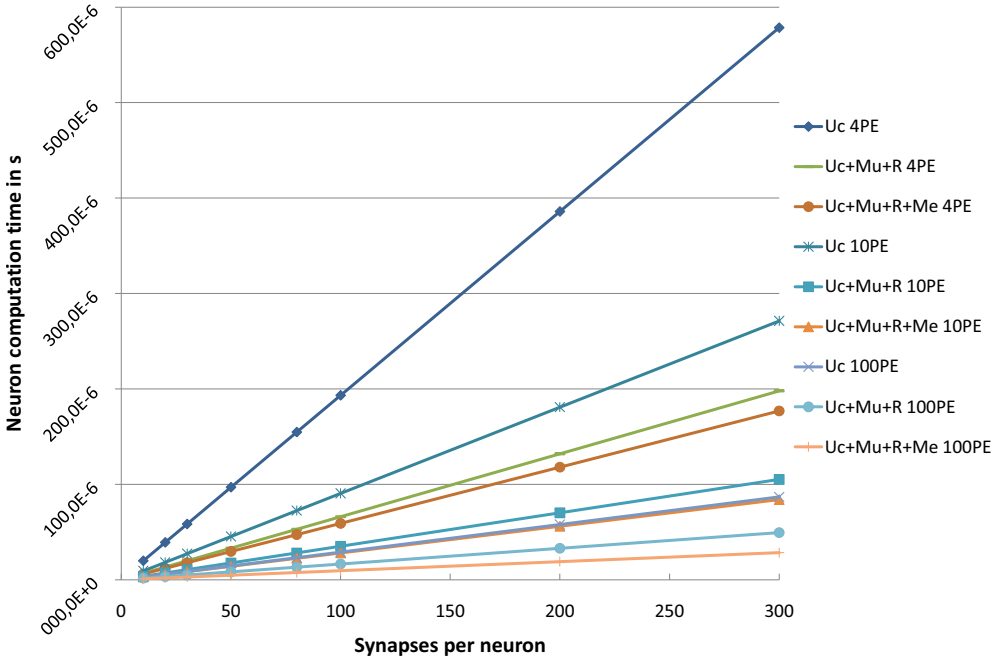

Figure 4.19: *100 PEs Ubichip architectures compared performance*

As presented in the above, speedup from 1.2 to 3.1 can be obtained in this case. Such improvements clearly increase the interest of the Ubichip SIMD mode as the 87 ms per neuron per simulation time step result may be reduced to 28 ms. Compared to the XScale performance of 540 ms an optimized Ubichip may provide a speedup around 19X if both architecture are running at 520MHz.

Figure 4.20 presents the comparison between benchmarked architecture and the proposed Ubichip evolution in term of clock cycles required to compute one time step of one neuron. The 3x performance improvement is noticeable on this figure if compared with Figure 4.17. The most important point to notice is that the improved Ubichip architecture using 100 PEs is close to the 24 PE GPU (G105m) in term of computing clock cycles.

To explain the remaining performance difference between GPUs and the Ubichip, several architectural considerations may be used. The following points are only examples of theses architectural differences.

- Dual Data Rate 3 (DDR3) memories clocked at 700MHz (G105m) at least are much faster than the proposed Static Random Access Memory (SRAM) clocked at 100MHz associated to the Ubichip. Memory buses are also different, 512 bits for the GTX280, 64 bits for the G105m and 32 bits for the Ubichip.

- An efficient memory hierarchy is used inside GPUs allowing them to reduce and mask memory interactions. This feature is not included in the Ubichip for area and cost reasons.

- ALUs of these different devices are not of the same complexity. The even the proposed evolution of the Ubichip is built around simple ALUs whereas GPUs are based on complex computing units able to compute evolved instructions such as *multiply-add* in floating point precision.

- Finally a sequencer hierarchy is proposed in the SIMT architecture of GPUs. Based on this hierarchy clusters of PEs are more flexible than the Ubichip SIMD allowing different clusters to work in an asynchronous way.

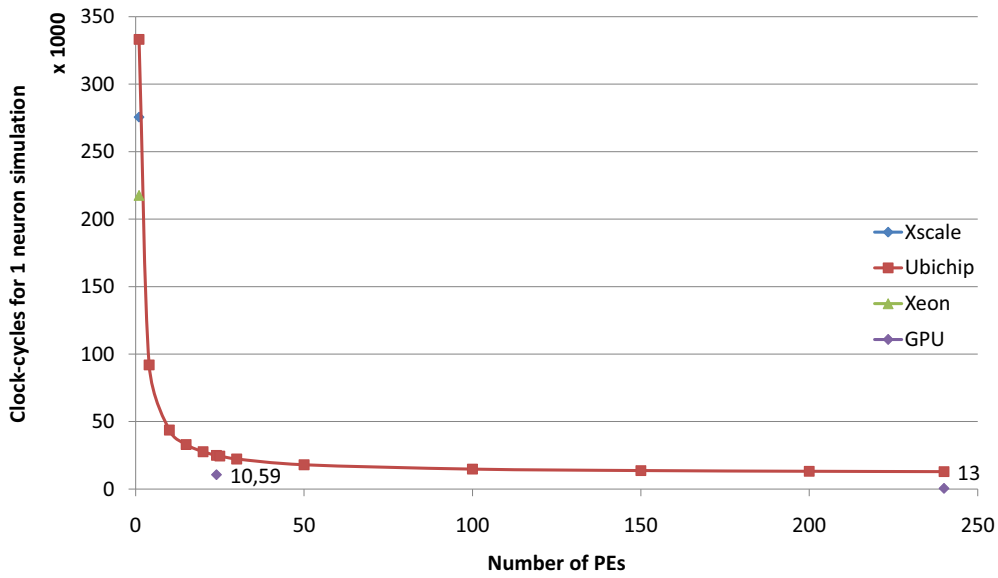

Figure 4.20: *Compared clock cycles to compute 1 neuron*

As a conclusion the proposed improvements greatly impact the Ubichip performance and should be considered for an Ubichip next version. Several other improvements maybe useful but require more silicon area and smaller technologies to be included in the Ubichip. In the academic context of the Ubichip development such features are difficult to integrate taking into account that the SIMD mode of the Ubichip is not the sole functional mode of this chip; the *native* mode can also be used to simulate bio-inspired features inside the chip.

4.5 Conclusion on the Jubi Methodology

In this chapter we present a contribution that aims at allowing performance increase by the use of hardware accelerators with a minimal impact concerning the application programming. The proposed generic approach allows compiling hardware accelerator code from BAF user agent descriptions. It has been tested with the Ubichip architecture. This eases the development of hardware accelerated applications compared to a complete description of the accelerator code written in assembly. Therefore, the proposed solution allows the user to use the Ubichip without knowing its specific SIMD assembly.

The code splitting in Kernels and envelope also offers the opportunity to use this acceleration transparently from the BAF environment. Therefore hardware accelerated agent can be used as purely software agents and communicate with any other agent on any local or remote platform.

Some improvement can be done in future versions of the JubiCompiler as the register management clearly reduces the obtained performance. Investigations show that with an optimized register management Ubichip performance may be increased with a 1.4 ratio (i.e. around 40%).

During the development of the JubiTool toolchain, we noticed some limitations in the Ubichip instruction set and therefore were able to correct the architecture. In our case, the development of a compiler helped to check the Turing compliance of the Ubichip. Some over modifications have been proposed for a future version of the Ubichip to improve performance with a 3.1 ratio.

Due to delays in the Ubichip fabrication we were not able to implement the Spiking Neural Network simulator on several final Ubidules. Therefore the presented results are only representative of the use of one Ubidule. Additional tests in real conditions are therefore required to observe the impact of using such accelerators in a pervasive context. In the following chapter, the general conclusion therefore draws some perspectives about the use of pervasive platforms.

Chapter 5

Conclusion

As exposed in this thesis, computer sciences evolve following the multiplication of computing devices per user; this is true from the early mainframes to the current mobile era. Therefore a still growing number of devices are already available for the upcoming Ubiquitous computing era. As explained in Chapter 2 programming methodologies and frameworks have evolved to better exploit capabilities of their computing paradigm; from hardware connection establishment (c.f ENIAC) to distributed computing (c.f. cloud computing), always proposing better hardware abstraction. The Ubiquitous computing trend appears to be the next computing era, environmental constraints which have a great influence on these platform compel us to improve the platform adaptability. Therefore adapted programming frameworks have to be proposed to take the full advantage of this architectural evolution while allowing them to adapt themselves to their surroundings. Current Sensor Networks (SNs) can be considered as a step toward Ubiquitous Computing platforms therefore experience on their programming methodologies can be used as a base to program pervasive systems.

Programming of such pervasive systems may be eased thanks to adapted solutions. In this thesis we focused our studies on the Agent-Oriented Programming (AOP) that offers interesting task distribution and communications possibilities. These low level properties of AOP are essentials in the Ubiquitous Computing context where numerous nodes have to cooperate inside a platform to achieve a global task. A second focus has been put on bio-inspired features, especially on Phylogeny, Ontogeny and Epigenesis, that have proven their adaptation capability in nature and improve the ability of pervasive platforms to adapt themselves to the environment. Therefore AOP appears as a

natural and appropriate solution for application description in the context of pervasive platform when used in conjunction with bio-inspired features that bring the required adaptability.

As described in Chapter 3, the proposed Bio-inspired Agent Framework (BAF) combines advantages of AOP and bio-inspiration in a programming environment that provides adaptability to the targeted platforms. Designed with portability and flexibility in mind the framework can be used with a wide range of hardware targets from HPC to SN. It includes services that ensure pervasive platform coherency, and allow to control the platform from a unique point. This allows to abstract the potentially changing topology of the platform in the user point of view. Even if the number of involved nodes is reduced, the proposed validation applications, made in the collaborative robotics field, clearly place targeted platforms in the Ubiquitous Computing paradigm. This application field provides interesting opportunities to experiment the proposed pervasive framework in real environmental conditions and test both bio-inspired and AOP aspects of the approach. Presented results are promising concerning the adaptability of the platform and framework services but this contribution requires more tests with more computing nodes to draw clear conclusions on the scalability aspect of the programming framework. In facts, with numerous collaborating robots, complex systems may be studied in a way that will ease the observation of emerging behaviors as proposed in the Perplexus project.

Presented in Chapter 4, performance improvement at the node level is at the base of the second contribution of this thesis. The antagonism in using low computing power devices to simulate complex systems can be solved by multiplying the number of devices as well as enhancing computing nodes with general purpose accelerators. The proposed solution eases the programming of these accelerators in the context of pervasive platforms. As a consequence, the programming complexity aspect has been taken into account and resulted the JUBI methodology and tools. This programming and compilation flow allows code compilation for two execution modes of targeted platforms: fully SW for debug and test without accelerators and hybrid HW/SW for improved performance on hardware accelerated platforms. This methodology, based on compiling a language extended from Java, allows the specification of code sections that will be accelerated. Thanks to the developed tools, this code selection can be done while describing BAF agents. Similar solutions are used for GP-GPU programming in environment such as CUDA, Stream Computing, OpenCL and DirectCompute for examples. These solutions proved their user friendliness and efficiency to accelerate

applications by tagging parallel sections of codes. Therefore, the JUBI methodology associated with the BAF allows to easily generate applications that take advantage of target accelerators. The example of the Ubichips has been used to test the proposed approach; therefore results are closely linked to this specific architecture. Presented results are promising but they do not take into account the communication cost between SW and HW sides. Additional tests are therefore required to conclude precisely on the current version of this JUBI proposal.

As a consequence of the two previous conclusions, short-term perspectives for these two contributions are mainly focused on gathering more quantitative results to give more general conclusions. Different complex systems can be simulated on the final Perplexus platform. Applications such as the cultural dissemination proposed by the UNIL partner are about to be ported to the Perplexus platform. Others applications are also used outside of the Perplexus project and are also relevant in the context of simulating complex systems, as an example, the Coast European project validated its contributions using a biomedical application, related to treatment of coronary artery diseases [130, 131]. Such applications may be used to open the Perplexus application fields. Additionally the support of the MarXbot robotic fleets - proposed and hosted at the École Polytechnique Fédéale de Lausanne (EPFL) - may be used as a pervasive hardware substrate for all these applications as well as to develop more complex collaborative applications. As a matter of fact, the number of available robots - about 70 units - their sensing capabilities, their autonomy and the technical support of the arena will greatly increase the potential of possible applications. Therefore, future work concerning the BAF should be based on this platform. This will give interesting results on the BAF and Ubichip performance.

A short to mid-term perspective of the work related in this document is to extend the Jubi approach to other targets such as other SIMD, MPSoC or MIMD platforms. This point is mandatory to improve the portability and interoperability of the proposed framework with various pervasive platforms. Techniques such as hardware virtualization [129] or code to description [132] - using the native mode of the Ubichip or others reconfigurable devices - may be used for this purpose. Additionally, several other solutions can be used. Methods such as direct porting will involve strong and low level changes in the code generation part of the JubiCompiler. This solution is highly time consuming and required for each additional target, but should to deliver the maximum performance of the target. The development of an Instruction Set Simulator (ISS) (i.e. an efficient a minimal virtual machine) that can be easily ported on different targets

may be a smart alternative as it would allow to rapidly use the proposed approach on a large amount of targets at the cost of interpreting the Ubichip language. This solution would therefore offer significantly less performance than the first proposed solution. The third possibility may be to use the approach GPU founders retained, it involves a system driver that acts as a wrapper between a virtual code (the PTX in the case of nVidia) and the hardware target. Main application code is executed on the system processor that triggers the accelerator driver functionalities. The parallel section virtual code is then read by this driver and mapped onto the targeted accelerator. Using this solution, the first communication between applications envelope and kernels is relatively long but the code is executed without interpreter and does not require any recompiling to be executed onto different accelerators.

Once the proposed solutions will be fully validated on a sufficient amount of pervasive applications, the following step may be to promote it by proposing it as a FIPA standard extension or even as a pervasive computing standard. This would allow its wide distribution and therefore to offer the possibility to industrials to integrate it into a large amount of devices - in 10 to 15 years. This may increase the number of home automation (domotics) aware devices, generalize the pervasive computing to the consumer market and therefore to actually step into the Ubiquitous Computing era.

Appendix A

Ubibot:Moving TankBot with a Ubidule

A.1 Hardware Implementation

The controller board of the TankBot (robotic part of the Ubibot) is equipped with two processors, master and slave processors. They communicate together through a serial synchronous link. This link is unidirectional from the master to the slave and it is made of two signals: a CLK signal and a DATA signal. These two signals are generated by software in the master processor and are using two digital outputs. The communication protocol has been defined by Wany Robotics, the master sends five bytes to the slave (see the "Communication Maitre-esclave" document by Wany Robotics), the slave is in charge of commanding of the two motors. For our project, the simplest modification is to replace the master processor of the TankBot by the Colibri and then generates the clock and data signals using the XScale. As shown on Figure A.1 we propose to disconnect the two signals DATA and CLK of the master processor (unsolder the pins 105 and 106) and then connect them to two GPIO signals of the Colibri evaluation board. We have chosen GPIO_12 for the DATA signal and GPIO_17 for the CLK signal. The signals will be generated by a software driver running on the Colibri.

A.2 Software driver implementation

The program which sends the motor commands to the slave processor of the TankBot is written in C language. It is called a native method and it will be called by our java program. To do this, we use the Java Native Interface (JNI):

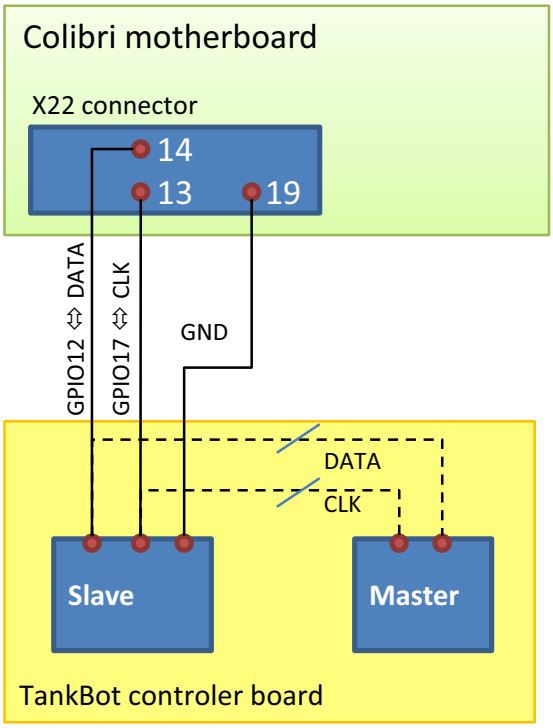

Figure A.1: *Connexion between the Colibri Board and the TankBot Board*

1. Java Source File: The Java Source file contains a native method:
   TankBot.java
2. Compile the Java source file.
3. Run javah on the resulting class file, javah reads the java class file and for each native method declaration, it generates a function prototype in a C header file (here TankBot.h)
4. Open the file TankBot.h and replace:
   Java_TankBot_move
   with:
   Java_Agents_Perplexus_TankBot_move
5. Now we have to create a C source file (move.c) that includes the javah generated header file and implements the native method. The code is then compiled and linked into a file called libTankBot.so
6. Relocate the library libTankBot.so in the /usr/lib folder of the COLIBRI
7. Launch the program on the COLIBRI.

The `move( )` method receives two speeds for the two motors and a duration in seconds. When called this way it sends the received values to the slave processor on

the TankBot board using the DATA serial signal and triggers the CLK to synchronise the communication. If the provided duration is negative the speed values order will be maintained until a new called to the `move` function overwrites it.

Input Parameters:

- `SpeedMotorRight` : Speed of the right motor between -7 and +7
- `SpeedMotorLeft` : Speed of the right motor between -7 and +7
- `Duration` : Duration of the movement in milliseconds

Table A.1 presents the UML description of the TankBot class

TankBot

`+ move( int SpeedMotorRight ,`
`int SpeedMotorLeft ,`
`int Duration )`

Table A.1: *class TankBot in UML representation*

Appendix B

MANET protocol of the BAF

B.1 OLSR and AODV detailed selection

As shown in Figure B.1 which depicts the average packet delivery delay as a function of node mobility (expressed in m/s) in a network composed of 50 nodes deployed on a 1000m^2 area, OLSR performs better latency versus nodes mobility than AODV achieving an overall lower latency. This underlines the benefits of proactive protocols which maintain routing tables up-to-date in case the network is made of mobile nodes as pervasive computing systems such as the PERPLEXUS platform.

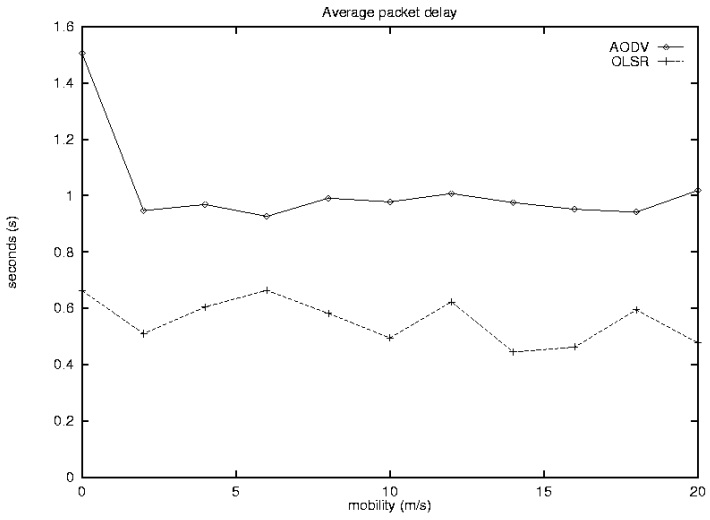

Figure B.1: *OLSR and AODV compared latency*

Figure B.2 depicts the difference between both routing algorithms in term of communication overhead, expressed in bytes per second. As expected, OLSR exhibits a constant communication overhead which only depends of the chosen proactive messages intervals. AODV exhibits a communication overhead which increases with the network usage, this due to the flood-search algorithm used to establish routes.

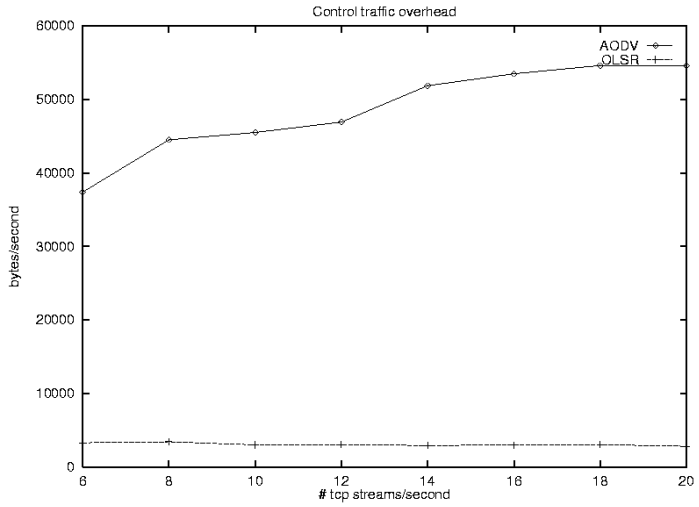

Figure B.2: *OLSR and AODV compared TCP Control Overhead in B/s*

On average, AODV control traffic overhead is around 20 to 30 times higher than that of OLSR for this setup which suggests that this protocol does not scale well with the number of nodes.

Figure B.3 depicts the average number of TCP messages required for one TCP message delivered. These results suggest that the OLSR implementation involve less retransmissions and results in a better communication reliability. Conducted experiments suggest that since the average message delivery delay in AODV is higher it accommodates less efficiently to topology changes; therefore several tries are often required in order to obtain a route.

Although we have not conducted any study on power consumption, which is one of the drawbacks of proactive protocols due to the constant control traffic, we estimate that this does not constitutes a major drawback. Furthermore, this traffic overhead can be lowered by adjusting the period of topology information messages.

For scalability reasons as well as low communication delay the OLSR protocol better suits our needs. As a conclusion OLSR protocol has been selected for its performance. It proves well suited to TCP streams which are wildely used by JADE (c.f. Section3.1.2.1).

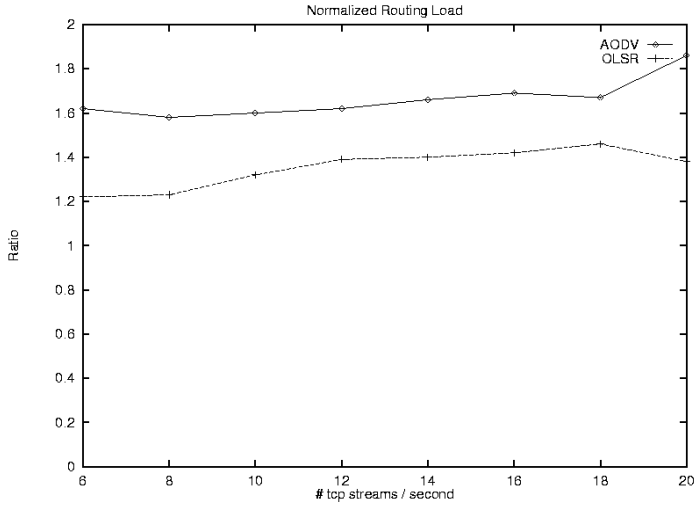

Figure B.3: *OLSR and AODV compared normalized routing load*

In order to validate the choice of OLSR tests in real conditions have been conducted, the following section presents tests protocols and results.

B.2 Experimental protocols and obtained results

Two experimental series of OLSR tests, one dynamic and one static, have been conducted to quantify data bit rate, network connectivity robustness and reactivity with ETX link quality disabled. In both tests, three Ubidules and a host station are used. The Wifi connection bandwidth is set to 11Mb/s as the Wifi device is connected to a USB 1.1 controller. The USB 1.1 protocol bandwidth is hardware limited to 12Mb/s.

The dynamic test uses one mobile node and three fixed one. The moving Ubidule follows the trajectory depicted in figure B.4 receiving packets flows incoming from the three static nodes at a theoretically fixed rate of $3 \times 58kb/s = 174kb/s$.

In this indoor test walls and a moving receiver are disturbing wireless communications inducing changes in the network topology. The presented results are based on one run that is representative of the overall experiment runs.

This test allows validating the use of the OLSR protocol in a real environment where node topology is changing. Figure B.5 shows the connectivity of the four peer used in

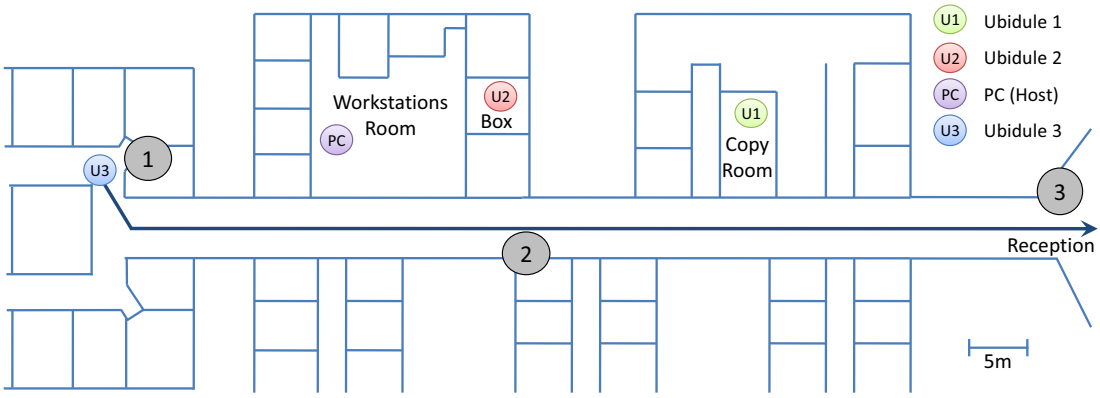

Figure B.4: *Experimental topology and connectivity test protocol*

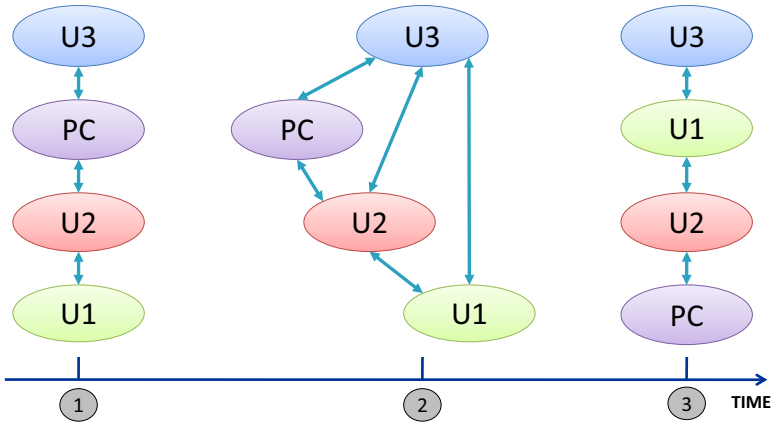

Figure B.5: *MANET changing topology*

the experiment. The time stamps 1, 2 and 3 of Figure B.5 correspond to the marked positions in Figure B.4.

During this experiment we collected statistics concerning the data bit rate of packet flows received by the mobile node from the PC, Ubidule1 and Ubidule2. Figure B.6 presents time dependent and consequently topology dependent bit rates of these static nodes. Graphs exhibit quite constant message flows with some positive and negative peaks.

A negative bit rate peak (i.e. a loss in bit rate) appears when a node becomes unreachable. In the case represented in Figure B.7 of a direct transmitter to receiver connection break this induces a loss of data. This case is indicated by red arrows in Figure B.6. Here packets are not received anymore and this situation may last up to 5 seconds using the selected parameters for OLSR. Temporary connection losses are not critical as communications are reestablished as soon as a new route has been calculated allowing TCP protocol to operate normally.

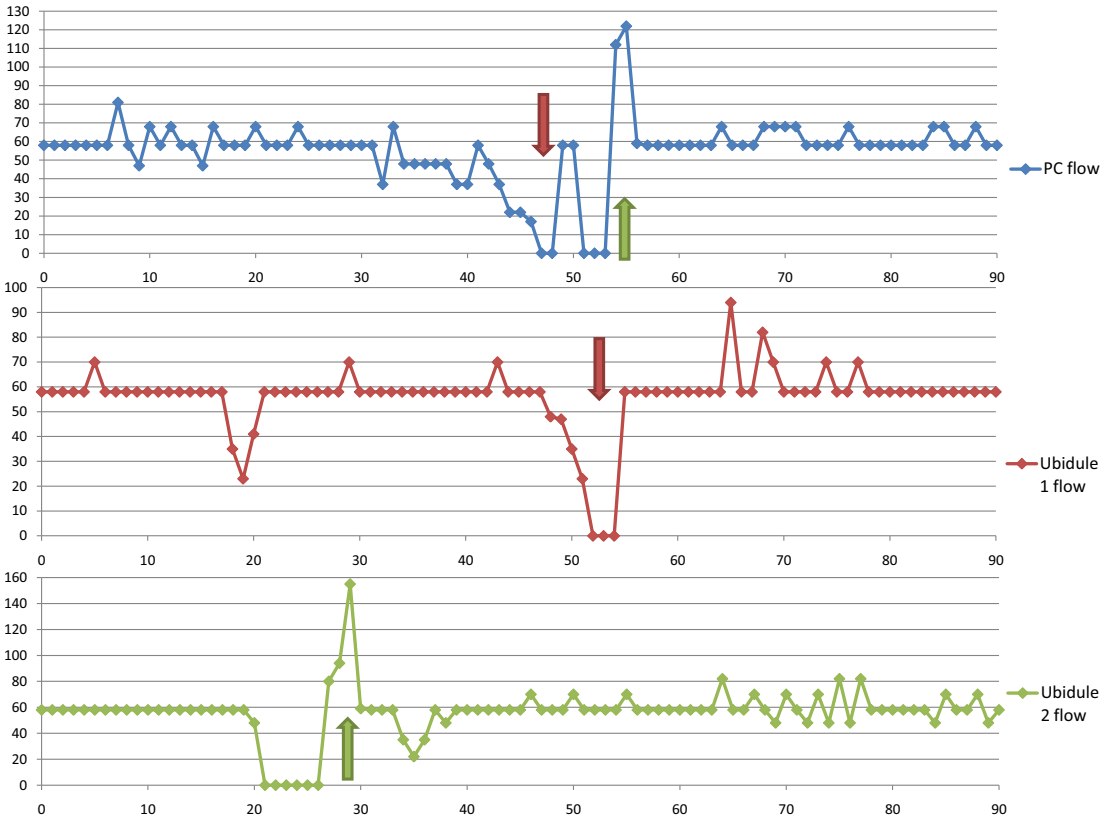

Figure B.6: *Data exchange rates while network topology changes*

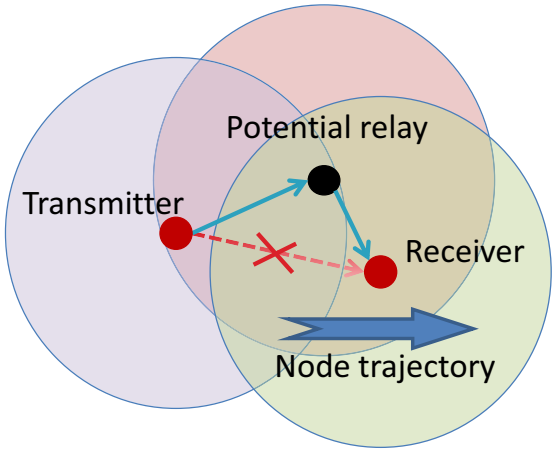

Figure B.7: *Direct connection break*

An indirect connection break may also happen has represented in Figure B.8. In this case packets cannot be transmitted form a relay to the next one or to the destination. These packets are temporarily stored in the relay node and send when the connection has been reestablished. In Figure B.6 two buffered transmissions are highlighted with light green arrows; they correspond to the sudden bite rate increase that occurs when stored packets are transmitted.

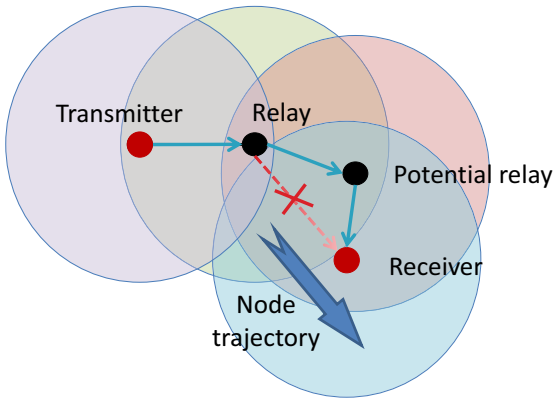

Figure B.8: *Relayed connection break*

Table B.1 presents TCP and UDP bit rates obtained during this test as well as the link quality (i.e. packet drop ratio for UDP and routing load for TCP).

	UDP	TCP
Average bit rate (in kb/s)	168	157
Packet drop ratio (in %)	5.93	-
Routing load (sent packets / received packets	-	1.13

Table B.1: *Mobile test results*

They corresponsd to the NS2 simulation results with 6% of UDP packet drop and arround 13% of TCP packet retransmissions. UDP and TCP compared errors have a factor of 2 that cope with possible TCP acknowlegment loss in addition to message loss.

The static test consists in sending ping data through a network composed of four nodes to gather delays and bit rates as a function of the number of hops that separate transmitter from receiver. In this experiment, the network is placed in a topology similar to the one of position three in the previous test (c.f. Figure B.5). Two types of messages where exchanged between the transmitter and the receiver. When 10000

messages have been transmitted, the target is replaced by the next one from the nearest to the furthest. Therefore increasing the number of hops required to reach the target. Figure B.9 shows the impact of the number of hops on the ping delay.

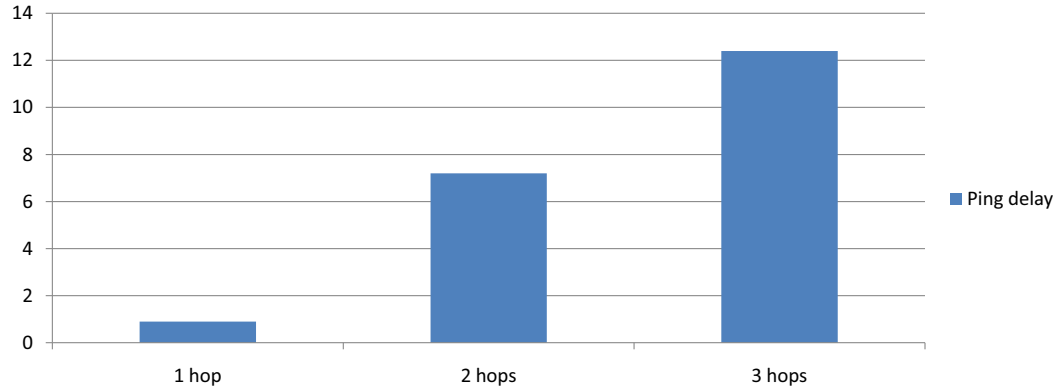

Figure B.9: *Observed ping delay vs Num. of transmitter to receiver hops*

The delay is affected by the number of hops, relay have to make the packet transit through their network stack to reroute the incoming packet to the next node as shown in Figure B.9. Therefore, the transmission delay is increased by 6ms per hops approximately using the Ubidule hardware. This also induces a decrease of the communication bit rate as shown in Figure B.10.

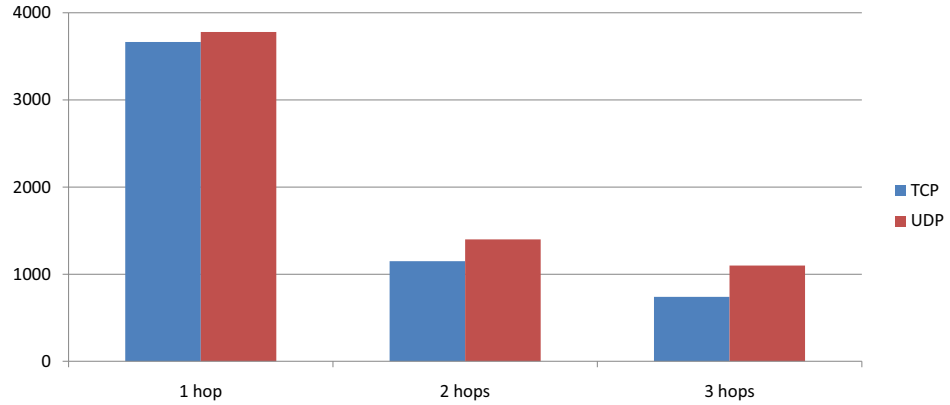

Figure B.10: *Observed bit rate vs Num. of transmitter to receiver hops*

If delay increase is proportional to the number of hop, bit rate is impacted differently. We noticed that low Wifi 11 Mb/s bandwidth was limiting data transfer rate of relays. This is due to the bandwidth sharing between reception and transmission of the relayed flows. This hardware limitation induces a severe drop of the final communication bit rate. When adding one or more hops (i.e. for connection using two or more relays) the bit rate decrease is less important and mainly comes of the increase in

transmission delay. Therefore communication bit rate is reasonnably scalable. Moreover, OLSR ETX link quality routing improves the network reliability as this algorithm provides slightly longer and consequently more buffered routes if the link quality decrease.

Finally, OLSR was functionally validated with a streaming video application; a mobile Ubidule prototype was equipped with a webcam and the video was streamed back on a host PC. The Ubidule was running the webcam server application that compresses video streams to to regularly updated JPEG images. the image flow can be visualised from a remote destination browsing a web page hosted on the robot. Using the image flow the robot could be remotely controlled in an area which corresponds to the maximum coverage obtained with OLSR for the mobile indoor tests with 4 nodes as depicted in Figure B.4. A demonstration video of this experiment is available online at:
`http://www.lirmm.fr/~brousse/Ubibots/video_stream.wmv`.

B.3 Conclusion

A first bibliographic approach led us to select two major MANET protocols one reactive AODV and one proactive OLSR. Based on network simulations and offered possibilities we finally selected OLSR that offers us the required services and performance level. This point have been validated in a first time through dynamic and static tests using message flows. In a second time a live video streaming application have been developed to practically see what is the influence of topology changes in a network based application runtime.

Appendix C

Validation test suite

In this annex meaningful functionality tests are presented. These tests were used during the JubiTool development to validate generated codes and therefore the complete compilation flow. The presented codes are extracted from a PERPLEXUS application to ensure that all requirements are fulfilled by the compiler.

C.1 Test 1

The test 1 family basically check if arithmetical functions are well implemented. The following code tests additions,subtractions and multiply operations inside a single expression.

```
1       {
2               final int Initial = 5;
3               final int K = 2;
4
5               int var1 [NPE] = {1,2,3,4};
6               int var2 [NPE] = {0,10,5,15};
7               int var3 [NPE] = {-5,5,0,-2};
8               int var4 [NPE];
9
10              var4 = (Initial + (var1 + 1) *
11              (var2 - Initial) * K) + var3;
12      }
```

Equivalent tests have been done with divide operations. The whole family tests the ability of the compiler to analyse arithmetical expressions and to produce the code to compute such expressions in the SIMD parallel Ubichip assembly.

C.2 Test 2

The second family is oriented on testing logical expressions and conditional structures. The SIMD scheme brings specific constraints as every PE is operating on different values test results may be different. This combine with the strict restriction on a single instruction execution on the whole array leads to execute both branches of a software test as in the following example.

```
1       {
2               final int Initial = 5;
3               final int K = 2;
4
5               int var1 [NPE] = {1,2,3,4};
6               int var2 [NPE] = {0,10,5,15};
7               int var4 [NPE];
8               int var5 [NPE];
9
10              if(var1 == K)
11              {
12                      var4 = var2 + Initial;
13              }
14              else
15              {
16                      var4 = var2 * Initial;
17              }
18              var5 = var4;
19      }
```

In this 4 PEs case, 1 PE has a “true” answer to the “if” test and 3 have a “false” condition. The FREEZE instructions are therefore used to lock the register writings on the false PEs during execution of the “true” branch and vice-versa.

C.3 Test 3

The third test family checks basic iterative structures. Basically for, while and do while loops with a fix and PE constant number of iterations as presented in the following example. This kind of loop does really make sense in the SIMD paradigm as PEs are always executing the same instructions.

```
1       {
2               final int Initial = 5;
3               final int K = 3;
4
5               int var1 [NPE] = {1,2,3,4};
6               int var4 [NPE] = new int [5];
7               int var5 [NPE];
8
9               for(int i = 0; i < Initial; i=i+1)
10              {
11                      var4 [i] = K*i+1;
12              }
13              var5 = var4[2];
14      }
```

This kind of loops are quite restrictive and the `freeze` state allows using more flexible loop as presented in the fifth test family.

C.4 Test 4

The forth test family is intended to check nested statement blocs as the `if/else` statement embedded in the `for` loop of the presented example.

```
1       {
2               final int Initial = 5;
3               final int K = 3;
4
5               int var1 [NPE] = {1,2,3,4};
6               int var2 [NPE] = new int [5];
```

```
7               int var3 [NPE];
8
9               for(int i = 0; i < Initial; i=i+1)
10              {
11                      if(var1 == K)
12                      {
13                              var2 [i] = (var1*i)+1;
14                      }
15                      else
16                      {
17                              var2 [i] = (var1+i)+1;
18                              var1 = var1 + 1;
19                      }
20              }
21              var3 = var2[0];
22      }
```

These tests allow checking the support for multiple levels codes and the nested freeze statement. This is a critical point as PEs may be frozen at different levels and should not be restored until their own level restoration point is reached.

C.5 Test 5

The fifth test group proposes advanced nested structure with PE changing number of iterations as proposed in the presented example. This test family is also used to check data vectors maniulation.

```
1       {
2               final int Initial = 5;
3               final int K = 2;
4
5               int var1 [NPE][] = {{3,6,9,12},{1,2,3,4},
6               {2,4,8,12},{5,10,15,20},{6,12,18,24}};
7               int var2 [NPE][] = {{2,2,3,2},{3,2,2,2},
8               {2,3,2,2},{2,2,2,3},{0,1,0,2}};
9               int tmp [NPE];
10              int var3 [NPE];
```

```
11              int var4 [NPE];
12
13              for(int i = 0; i < Initial; i=i+1)
14              {
15                      for(int j = 0; j < K; j=j+1)
16                      {
17                              tmp = tmp + var1[j];
18                      }
19                      tmp = tmp * var2[i];
20                      var3 = var3 + tmp;
21                      tmp = 0;
22              }
23              var4 = var3;
24      }
```

Thanks to this group, freeze states memory writing policy and indexed addressing are checked. Therefore data vector can be used for application programming.

C.6 Test 6

The sixth test family is based on the fifth family principles with an increased complexity as all loops have dynamic and PE varying sizes.

```
1       {
2               int var1 [NPE] = {1,2,3,4};
3               int var2 [NPE] = {1,5,5,3};
4
5               int var3 [NPE][] = {{1,1,1,1},{2,2,2,2},
6               {3,3,3,3},{4,4,4,4},{5,5,5,5},{6,6,6,6},
7               {7,7,7,7},{8,8,8,8},{9,9,9,9},{10,10,10,10}};
8               int var4 [NPE][] = {{2,2,2,2},{4,4,4,4},
9               {6,6,6,6},{8,8,8,8},{1,1,1,1},{3,3,3,3},
10              {5,5,5,5},{7,7,7,7},{9,9,9,9},{0,0,0,0}};
11
12              int tmp [NPE];
13              int var5 [NPE];
14
```

```
15              int i;
16              for(i = 0; i < var1; i=i+1)
17              {
18                      int j;
19                      for(j = 0; j < var2; j=j+1)
20                      {
21                              tmp = tmp + var3[j];
22                      }
23                      var5 = var5 + tmp*var4[i];
24                      tmp = 0;
25
26              }
27              tmp = var5;
28      }
```

Such tests are useful to check the complete support of indexed addressing, memory storage policy and freeze policy of the compiler.

C.7 Test 7

The seventh test family is oriented to check the compiler support for the Ubichip internal Random Number Generator (RNG). This hardware Linear Feedback Shift Register (LFSR) is able to generate random number by shifting a given seed and applying some logical xor operations on different bits. The compiler includes the required features to support this RNG.

```
1       {
2               int i = 0;
3               int var [NPE] = {0,0,0,0};
4               Random RandTmp = new Random(26835);
5               final int threshold = 6835;
6               int Rand [NPE];
7               int iterations[NPE];
8
9               while(i < 2)
10              {
11                      Rand = RandTmp.nextInt();
```

```
12                          if(Rand >= threshold)
13                          {
14                                  i=i+1;
15                          }
16                          var=var+1;
17                  }
18                  iterations = var;
19          }
```

The complete set of these functional tests ensures that the compiler generated code is compliant with the Ubichip SIMD assembly and functionally correct.

List of Figures

1.1 Pascaline (Blaise Pascal, 1642) 10
1.2 ITRS 2007 CMOS technology forecast. 12
1.3 Evolution of the computing paradigm. 15

2.1 Example of parallel architecture with a shared memory 25
2.2 Example of parallel architecture based on memory Hierarchy 26
2.3 Example of a parallel architecture with bus and memory hierarchies . 27
2.4 Example of parallel architecture based on private memories and network based communications 28
2.5 Thread programming on parallel architectures 29
2.6 Parallel Computing paradigm 30
2.7 Distributed Computing solution 31
2.8 HPC general principle 32
2.9 Grid general principle 33
2.10 Wireless Grid general principle 34
2.11 Sensor Network solution 36
2.12 Pervasive Computing challenges 45
2.13 Early Perplexus platform overview 51
2.14 Ubichip overview 51
2.15 Perplexus mobile bio-inspired overview 53

3.1 FIPA classical framework 58
3.2 JADE platform 60
3.3 Agent and Behaviours in JADE 61
3.4 Overview of the agents of the framework 63
3.5 Evolution on JADE framework for bio-inspiration support 67
3.6 Mobile Ad-hoc NETwork principle 68
3.7 OLSR routing alternatives 71
3.8 Distributed hostname/IP translation 72
3.9 Evolution on JADE framework for network support 74
3.10 Peer search service 75
3.11 Service search protocol 75
3.12 GUI main window of the BAF 77

3.13 Framework agents command interactions 77
3.14 Application parameters examples 78
3.15 Node received its parameters 79
3.16 Application *start* and *stop* commands 80
3.17 Evolution on framework: Host agent 81
3.18 JADE enhancement for HW/SW interfaces and activity tracing 83
3.19 Example of a POE application scheduling with three nodes 84
3.20 General application environment 86
3.21 Evolution-driven application overview 87
3.22 Application involved agents 88
3.23 Application overview 89
3.24 Collaborative learning application overview 89
3.25 Application involved agents 90
3.26 Application involved agents 91
3.27 Second generation application steps 92
3.28 Robot genome example 94

4.1 Sequential and parallel mixed parts of a program 100
4.2 Jubi compilation flow, from source code to HW/SW hybrid execution 101
4.3 Jubi splitting scheme 103
4.4 Software and hybrid execution modes 106
4.5 Ubichip configured in SIMD processor 108
4.6 CAM updates neurons' input signals depending on data received via the AER bus 109
4.7 Ubichip SIMD indexed addressing 113
4.8 Ubichip conditional memory writing 114
4.9 Ubichip conditional memory writing 115
4.10 SNN simplified algorithm 121
4.11 nVidia Texture Processing Cluster architecture 122
4.12 nVidia GTX280 architecture overview 123
4.13 nVidia CUDA Compilation flow overview 124
4.14 GPU versus CPU implementations speedup 126
4.15 CPU execution time 127
4.16 GPU execution time 127
4.17 Compared clock cycles to compute 1 neuron 128
4.18 Ubichip possible evolutions performance 130
4.19 100 PEs Ubichip architectures compared performance 131
4.20 Compared clock cycles to compute 1 neuron 132

A.1 Connexion between the Colibri Board and the TankBot Board 140

B.1 OLSR and AODV compared latency 143
B.2 OLSR and AODV compared TCP Control Overhead in B/s 144
B.3 OLSR and AODV compared normalized routing load 145
B.4 Experimental topology and connectivity test protocol 146
B.5 MANET changing topology 146

B.6 Data exchange rates while network topology changes 147
B.7 Direct connection break 147
B.8 Relayed connection break 148
B.9 Observed ping delay vs Num. of transmitter to receiver hops 149
B.10 Observed bit rate vs Num. of transmitter to receiver hops 149

List of Tables

3.1 Qualitative comparison between selected platforms 59
3.2 AODV OLSR comparative table 70
3.3 Ultrasonic sensors areas 90
3.4 Quality function example 94

4.1 Per neuron computation time comparison 125

A.1 class TankBot in UML representation 141

B.1 Mobile test results 148

Bibliography

Introduction

[1] I. Akyildiz, Y. Weilian Su Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Commun. Mag*, vol. 40, pp. 102–114, 2002.

[2] M. Campbell-Kelly and W. Aspray, *Computer: A History of the Information Machine*, B. Books, Ed. Basic Books, 1996, iSBN 0-465-02989-2.

[3] M. Croarken, *Annals of the History of Computing*. IEEE, 2000, ch. Howard Aiken: portrait of a computer pioneer, pp. 89–89.

[4] D. E. Goldberg, "Genetic algorithms in search, optimization and machine learning," 1989.

[5] B. Goldworm and A. Skamarock, *Blade Servers and Virtualization: Transforming Enterprise Computing While Cutting Costs*. John Wiley & Sons, 2003.

[6] A. Greenfield, *Everyware: The Dawning Age of Ubiquitous Computing*. Berkeley, CA: New Riders Publishing, 2006.

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: University of Michigan Press, 1975.

[8] Intel corp., *Intel Core i7 Processor Extreme Edition and Intel Core i7 Processor*, nov 2008.

[9] International Technology Roadmap for Semiconductors, "System drivers report," online, 2007. [Online]. Available: http://www.itrs.net/links/2007ITRS/2007_Chapters/2007_SystemDrivers.pdf

[10] W. J. Kaufmann and L. L. Smarr, *Supercomputing and the Transformation of Science*. W. H. Freeman & Co, 1992.

[11] T. Kleinwort, "The linux pc farms at desy zeuthen," online, October 1998, http://www.ifh.de/computing.

[12] Perplexus.org, "Pervasive computing platform for modeling complex virtually-unbounded systems," online: 2009-03-06. [Online]. Available: www.perplexus.org

[13] S. Pravin, "Sensor networks - a survey," May 2007.

[14] D. Reed, *Balanced Introduction to Computer Science*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[15] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[16] Y. Shoham, "Agent oriented programming: An overview of the framework and summary of recent research," *Lecture Notes in Computer Science*, pp. 123—129, 1994. [Online]. Available: http://www.springerlink.com/content/p94132816510k828/fulltext.pdf

[17] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 83–97, April 1997.

[18] E. Souteyrand, J. P. Cloarec, J. R. Martin, W. C., M. F. Lawrence, and S. Mikkelsen, "Direct detection of the hybridization of synthetic homo-oligomer dna sequence by field effect," *The Journal of physical chemistry. B, Materials, surfaces, interfaces & biophysical*, vol. 101, no. 15, pp. 2980–2985, 1997.

[19] A. M. Turing, "Intelligent machinery," *Machine Intelligence*, vol. 5, pp. 3–23, 1969.

[20] University of California, "Boinc : Open-source software for volunteer computing and grid computing," Online, January 2009. [Online]. Available: http://boinc.berkeley.edu/

[21] M. H. Weik, H. H. Goldstine, and P. N. Gillon, "The ENIAC story," *The Journal of the American Ordnance Association*, vol. 1-2, January-February 1961.

[22] Wikipedia, "Comparison of wireless data standards," online, September 2009, last saw: 19/10/2009. [Online]. Available: http://en.wikipedia.org/wiki/Comparison_of_wireless_data_standards

[23] M. A. Arbib, Ed., *Handbook of Brain Theory and Neural Networks.* MIT Press, 1995, ISBN: 978-0-262-01148-8.

State of the art and proposal background

[24] AMD, "Family 10h amd phenom ii processor - product data sheet," online, February 2009. [Online]. Available: http://support.amd.com/fr/Processor_TechDocs/46878Phenom_II_PDS_3.04_PUB.pdf

[25] ——, "Ati radeon hd 4780 x2 graphics," online, July 2009, seen on 2009.07.22. [Online]. Available: http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-4000/hd-4870X2/Pages/hd-4870X2.aspx

[26] ——, "Ati stream computing technical overview," online, AMD-ATI, Technical overview, 2009. [Online]. Available: http://developer.amd.com/gpu_assets/Stream_Computing_Overview.pdf

[27] ARM, "Arm11 mpcore architecture," Online, 2008, 2008-02-23. [Online]. Available: http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html

[28] J. Baeten, "A brief history of process algebra," Vakgroep Informatica, Technische Universiteit Eindhoven, Tech. Rep. CSR 04-02, 2004.

[29] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, *JADE a White Paper*, Teleconitalialab, sep 2003, http://exp.telecomitalialab.com.

[30] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007.

[31] J. M. Benyus, *Biomimicry: Innovation Inspired by Nature*. Perennial (HarperCollins), May 1998.

[32] T. Brecht, H. S, M. Shan, and J. Talbot, "Paraweb: Towards world-wide supercomputing," in *In European Symposium on Operating System Principles*, 1996, pp. 181–188.

[33] K. Burbeck, D. Garpe, and S. Nadjm-Tehrani, "Scale-up and performance studies of three agent platforms," in *Performance, Computing, and Communications 2004 IEEE International Conference on*, 2004, pp. 857–863.

[34] D. R. Butenho, *Programming with POSIX threads*, ser. Addison-Wesley Professional Computing Series. Addison-Wesley, 1997.

[35] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *Computing in Science and Engineering*, vol. 5, no. 1, pp. 46–55, Jan-Mar 1998.

[36] E. W. Dijkstra, *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1977, ch. The mathematics behind the Banker's Algorithm, pp. 308–312, 1982 edition ISBN 0-387-90652-5.

[37] ——, *Programming Languages.* Academic Press, 1968, ch. Cooperating Sequential Processes.

[38] ——, "Een algorithme ter voorkoming van de dodelijke omarming (in dutch; an algorithm for the prevention of the deadly embrace)," 1977.

[39] FIPA-OS project, *FIPA-OS Agent Toolkit*, FIPA-OS project, http://sourceforge.net/projects/fipa-os, 2007.

[40] I. Foster, C. Kesselman, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," globus.org, Tech. Rep., 2002.

[41] J. Fung and S. Mann, "Computer vision signal processing on graphics processing units," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)*, May 2004, pp. 83–89.

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

[43] D. Green, "Transitioning noaa moored buoy systems from research to operations," in *OCEANS 2006.* IEEE OCEAN 2006, September 2006, pp. 1–3, cRS Report for U.S.A Congress. [Online]. Available: http://www.fas.org/sgp/crs/RL32739.pdf

[44] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.

[45] J. L. Hennesy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 2003, iSBN number 1558605962, third edition.

[46] C. Hewitt, P. Bishop, and R. Steiger, "A universal modular actor formalism for artificial intelligence," *IJCAI*, 1973.

[47] J. L. Hill, "System architecture for wireless sensor networks," Ph.D. dissertation, University of California, Berkeley, 2003, adviser-Culler, David E.

[48] J. L. Hill and D. E. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE MICRO*, pp. 12–24, Nov-Dec 2002.

[49] T. A. Infiniband, "Infiniband solutions : Ibta - infiniband trade association," Online, 01 2010. [Online]. Available: http://www.infinibandta.org

[50] P. B. Johannes, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Proceeding of 2nd Int'l Conference on Mobile Data Management*, 2001, pp. 3–14.

[51] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM, 2002, pp. 96–107.

[52] Khronos, "Opencl," online, July 2009. [Online]. Available: http://www.khronos.org/opencl/

[53] S. M. Larson, C. D. Snow, M. R. Shirts, and V. S. Pande, *Computational Genomics*. Horizon Press, 2002, ch. Folding@home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology.

[54] C. Lavarenne, O. Seghrouchni, Y. Sorel, and M. Sorine, "The SynDEx software environment for real-time distributed systems, design and implementation," in *Proceedings of European Control Conference, ECC'91*, Grenoble, France, Jul. 1991. [Online]. Available: http://www-rocq.inria.fr/syndex/pub/ecc91/ecc91.pdf

[55] C. Lavarenne and Y. Sorel, "Specification, performance optimization and executive generation for real-time embedded multiprocessor applications with SynDEx," in *Proceedings of Real-Time Embedded Processing for Space Applications*. Les Saintes Maries de la Mer, France: CNES International Symposium, Nov. 1992. [Online]. Available: http://www-rocq.inria.fr/syndex/pub/cnes92/cnes92.pdf

[56] J. Lawrence, "LEAP into Ad-Hoc Networks," *ACM Workshop on Agents in Ubiquitous and Wearable Computing*, 2002, aAMAS.

[57] T. Liu and M. Martonosi, "Impala: A middleware system for managing autonomic, parallel sensor systems," in *Procceeding of ACM SIGPLAN Symposium in Principles and Practice of Parallel Programming (PPoPP 03)*, 2003, pp. 107–118.

[58] LSRO-EPFL, "Miniature mobile robots group," online, Jun 2009. [Online]. Available: www.marxbot.org

[59] R. Müller, G. Alonso, and D. Kossmann, "A virtual machine for sensor networks," vol. 41, no. 3. New York, NY, USA: ACM, 2007, pp. 145–158.

[60] J. Maassen, T. Kielmann, and H. E. Bal, "Parallel application experience with replicated method invocation." in *Concurrency and Computation: Practice and Experience*, ACM, Ed., vol. 13, no. 8, July 2001, pp. 681–712.

[61] madkit.org, *MAdKit.org*, http://madkit.org/, 2007, online: 11-22-2005.

[62] S. Magnenat, V. Longchamp, and F. Mondada, "ASEBA, an event-based middleware for distributed robot control," in *IROS*. IEEE Press, 2007. [Online]. Available: http://mobots.epfl.ch/aseba.html

[63] M. Modahl, B. Agarwalla, and T. S. Saponas, "Toward a standard ubiquitous computing framework," in *Middleware 2004 Companion*. ACM, October 2004, pp. 135–139.

[64] M. Modahl, I. Bagrak, M. Wolenetz, P. Hutto, and U. Ramachandran, "Mediabroker: An architecture for pervasive computing," *Pervasive Computing and Communications, IEEE International Conference on*, vol. 0, p. 253, 2004.

[65] J. M. Moreno, "Specification of the ubicell," UPC, Tech. Rep., 2007, internal report. Version 1.1.3.

[66] J. M. Moreno and J. Madrenas, "Detailed functionnal specification of the system architecture in hardware description languages (vhdl & vhdl-ams)," UPC, Tech. Rep., 2007, deliverable 2.2.

[67] P. Nebot and E. Cervera, "A framework for the development of cooperative robotic applications," in *IEEE. July 18-20, 2005, 12th International Conference on Advanced Robotics*, 2005, pp. 901–906.

[68] Nissan, "Nissan eporo robot car "goes to school" on collision-free driving by mimicking fish behavior," Online, October 2009. [Online]. Available: http://www.nissan-global.com/EN/NEWS/2009/_STORY/091001-01-e.html

[69] NVIDIA, "Geforce gtx 295," online, July 2009, seen on 2009.07.22. [Online]. Available: http://www.nvidia.com/object/product_geforce_gtx_295_us.html

[70] ——, "Visual computing for mobile devices - nvidia tegra," Online, September 2009. [Online]. Available: http://www.nvidia.com/page/handheld.html

[71] ——, "Accelerating matlab with cuda," nVidia corporation, White paper, September 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/1_0/Accelerating%20Matlab%20with%20CUDA.pdf

[72] ——, "Cuda architecture overview," online, September 2007. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf

[73] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, University of Bonn, 1962.

[74] M. Pirker, M. Berger, and M. Watzke, "An approach for fipa agent service discovery in mobile ad hoc environments," in *Workshop on Agents for Ubiquitous Computing, in conjunction with AAMS 2004*, Jully 2004.

[75] Qualcomm, "http://www.qctconnect.com/products/snapdragon.html," Online, September 2009. [Online]. Available: http://www.qctconnect.com/products/snapdragon.html

[76] K. Römer and F. Mattern, "The design space of wireless sensor networks," *IEEE Wireless Communications*, 2004.

[77] R. Riem-Vis, "Cold chain management using an ultra low power wireless sensor network," in *WAMES*. Boston, USA: ACM Sigmobile, June 2004.

[78] A. Ruiz, M. Ujaldon, J. Andrades, J. Becerra, T. Kun Huang Pan, and J. Saltz, "The gpu on biomedical image processing for color and phenotype analysis," in *Proceedings of the 7th IEEE International Conference on Bioinformatics and Bioengineering*, October 2007, pp. 1124–1128.

[79] N. Saint Jean, "Study and design of self-adaptive multiprocessor systems for embedded systems," Dicipline: Génie informatique, automatique et traitement du signal, Université de Montpellier 2, December 2008.

[80] Sharklet Technologies, "A material based on sharkskin stops bacterial breakouts," Popular sciences: Online, October 2009. [Online]. Available: http://www.popsci.com/science/article/2009-10/saving-skin

[81] Y. Shoham, "Agent oriented programming," *Journal of Artificial Intelligence*, vol. 60, pp. 52–91, 1993.

[82] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 1–12.

[83] E. Souto, G. Guimar aes, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz, "A message-oriented middleware for sensor networks," in *MPAC '04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*. New York, NY, USA: ACM, 2004, pp. 127–134.

[84] F. Strauß, J. Schönwälder, and S. Mertens, "Jax– a java agent x subagent toolkit," july 2000.

[85] G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma, R. Canham, A. Tyrrell, and J. M. Moreno, "A poetic architecture for bio-inspired hardware," *Conference on the Simulation and Synthesis of Living Systems (Artificial Life VIII)*, pp. 111–115, dec 2002, sydney, Australia,**[IST-2000-28027]**. [Online]. Available: www.poetictissue.org

[86] Texas Instrument, "Omap 4 mobile applications platform," 2009. [Online]. Available: http://focus.ti.com/lit/ml/swpt034/swpt034.pdf

[87] Y. Thoma, "Tissu numérique à routage et configuration dynamique," Ph.D. dissertation, École Polytechnique Fédérale de Lausanne, 2005.

[88] Y. Thoma and A. Upegui, "Specification of bio-inspired features to be supported by the device," HEIG-VD, Tech. Rep., December 2006.

[89] Toradex, "Colibri xscale pxa270 datasheet," online, December 2006. [Online]. Available: http://www.toradex.com/@api/deki/files/29/=Colibri_PXA270_Datasheet_Rev_1.4.pdf

[90] M. Weiser, "Building invisible interfaces," Keynote, November 1994.

Wikipedia, "Bio-inspiration in the micro air vehicle article," Online. [Online]. Available: http://en.wikipedia.org/wiki/Micro_air_vehicle#Bio-inspiration

[91] ——, "Universal plug and play - definition and references," Online, octobre 2009 2009. [Online]. Available: http://fr.wikipedia.org/wiki/Universal_Plug_and_Play#Probl.C3.A8mes_d.27UPnP

[92] D. Zöbel, "The deadlock problem: a classifying bibliography," *ACM SIGOPS Operating Systems Review*, vol. 17, no. 4, pp. 6–15, October 1983, iSSN 0163-5980.

[93] M. Zamith, E. Clua, A. Conci, and A. Montenegro, "Parallel processing between gpu and cpu: Concepts in a game architecture," in *Computer Graphics, Imaging and Visualisation*, August 2007, pp. 115–120.

[94] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi, "Hardware design experiences in zebranet," *Sensys*, pp. 227–238, November 2004.

[95] I. Foster and C. Kesselman, Eds., *The Grid : Blueprint for a New Computing Infrastructure*, 1998, second edition (2005), First edition(1998).

Bio-inspiration Agent Framework

[96] G. Abrami, O. Barreteau, and F. Cernesson, "An agent-group-role based modelling framework for participative water management support," in *IEMSS*, Lugano, Switzerland, June 2002, pp. 497–502.

[97] Apple, "Apple bonjour!" Online, 2007. [Online]. Available: www.apple.com/macosx/features/bonjour/

[98] Avahi Team, "avahi.org," online, 2007. [Online]. Available: http://avahi.org

[99] U. Berkeley, LBL, USC/ISI, and X. PARC, "The ns manual," July 2007, vINT Project.

[100] D. Chakeres, Ian and M. Belding-Royer, Elizabeth, "Aodv routing protocol implementation design," in *Proceedings of the International Workshop on Wireless Ad Hoc Networking*, March 2004, tokyo, Japan.

[101] T. Clausen, P. Jacquet, and L. Viennot, "Comparative study of CBR and TCP performance of MANET routing protocols," *Workshop MESA*, 2002, INRIA Roquencourt, HiPERCOM project.

[102] INRIA-HIPERCOM, "Olsr routing protocol (rfc3626)," Online, 2005. [Online]. Available: http://hipercom.inria.fr/olsr/

[103] B. Johnson, David, A. Maltz, David, and J. Broch, "Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks," *Ad Hoc Networking*, vol. 1, pp. 139–172, 2001.

[104] P. Kaelbling, Leslie, L. Littman, Michael, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[105] G. Pei, M. Gerla, and T.-W. Chen, "Fisheye state routing in mobile ad hoc networks." in *ICDCS Workshop on Wireless Networks and Mobile Computing*, vol. 1, 2000, pp. 71–78.

[106] G. Pei, M. Gerla, and X. Hong, "Lanmar: Landmark routing for large scale wireless ad hoc networks with group mobility," 2000.

[107] Z.-P. Peng and H. Peng, "An improved agent/group/role meta-model for building multti-agen systems," in *International Conference on Machine Learning and Cybernetics*, 2005.

[108] C. E. Perkins and E. M. Royer, "Ad-hoc on-demande distance vector," Dec. 1998.

[109] I. Satoh, "Bio-inspired organization for multi-agents on distributed systems," in *Biologically Inspired Approaches to Advanced Information Technology*, ser. Lecture Notes in Computer Sciences, S. B. . Heidelberg, Ed., vol. 3853, 2006, pp. 355–362.

[110] A. Tønnesen, T. Lopatic, H. Gredler, B. Petrovitsch, A. Kaplan, S.-O. Tücke *et al.*, "Olsrd an adhoc wireless mesh routing daemon," 2009. [Online]. Available: http://www.olsr.org/

[111] T. White, B. Pagurek, and D. Deugo, "Biologically-inspired agents for priority routing in networks," in *In Proceeding of FLAIRS-02*. American Association for Artificial Intelligence AAAI (www.aaai.org), 2002, pp. 282–287.

[112] X. Yu and R. Bala, "Bio-inspired scheduling for dynamic job shops with flexible routing and sequence-dependent setups," *International journal of production research*, vol. 44, no. 22, pp. 4793–4813, November 2006.

[113] E. Mangina, J. Carbo, and J. M. Molina, Eds., *Agent-based ubiquitous computing*, ser. ATLANTIS AMBIENT AND PERVASIVE INTELLIGENCE. World Scientific, August 2009, vol. 1.

Jubi methodology

[114] Altera, "MAX II Device Handbook," online, August 2009. [Online]. Available: http://www.altera.com/literature/hb/max2/max2_mii5v1.pdf

[115] ——, "Stratix IV Device Family Overview," online, June 2009. [Online]. Available: http://www.altera.com/literature/hb/stratix-iv/stx4_siv51001.pdf

[116] K. Boahen, "A throughput-on-demand address-event transmitter for neuromorphic chips," in *ARVLSI '99: Proceedings of the 20th Anniversary Conference on Advanced Research in VLSI*. Washington, DC, USA: IEEE Computer Society, 1999, p. 72.

[117] O. Chibirova, J. Iglesias, V. Shaposhnyk, and A. E. Villa, "Dynamics of firing patterns in evolvable hierarchically organized neural networks," *Lecture Notes in Computer Science*, 2008, lNCS 5216.

[118] gnu.org, "Bison - GNU parser generator," online: 2009-02-11. [Online]. Available: www.gnu.org/software/bison/

[119] ——, "Flex - A fast lexical analyser generator," online: 2009-02-11. [Online]. Available: www.gnu.org/software/flex/

[120] D. Grune, H. E. Bal, C. J. Jacob, and K. G. Langedoen, *Modern Compiler Desing*. John Wiley & sons,Ltd, 2000, england.

[121] 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)*, 2009.

[122] J. Iglesias, J. Madrenas, J. M. Moreno, and A. E. Villa, "D1.1: Specification of the biologically plausible developing neural networks modeling problem," UPC - UJF, Tech. Rep., 2007, uPC Barcelona, Internal report. Version 1.1.3.

[123] Lattice, "MachXO Family Data Sheet," online, June 2009. [Online]. Available: http://www.latticesemi.com/lit/docs/datasheets/cpld/DS1002.pdf

[124] ——, "Lattice XP2 Family Handbook," online, August 2009. [Online]. Available: http://www.latticesemi.com/documents/HB1004.pdf

[125] J. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the aer protocol," in *ARVLSI '95: Proceedings of the 16th Conference on Advanced Research in VLSI (ARVLSI'95)*. Washington, DC, USA: IEEE Computer Society, 1995, p. 158.

[126] J. Madrenas, "Specification of the ubichip sequencer," UPC, Internal report: WP2-T2.3-UPC-1 1.21, July 28 2008.

[127] ——, "Virtex-6 Family Overview," online, June 2009. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[128] ——, "CoolRunner-II CPLD Family," online, september 2008. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds090.pdf

Conclusion

[129] W. Chen, H. Lu, L. Shen, Z. Wang, N. Xiao, and D. Chen, "A novel hardware assisted full virtualization technique," in *The 9th International Conference for Young Computer Scientists*, November 2008, pp. 1292–1297.

[130] COAST, "Mission of coast," Online, december 2009, european Project: STREP-IST-033664. [Online]. Available: http://www.complex-automata.org/

[131] D. Evans, P. Lawford, J. Gunn, D. Walker, D. Hose, R. Smallwood, B. Chopard, M. Krafczyk, J. Bernsdorf, and A. Hoekstra, "The application of multi-scale modelling to the process of development and prevention of stenosis in a stented coronary artery," *Philosophical Transactions of the Royal Society A*, vol. 366, pp. 3343–3360, September 2008.

[132] Wikipedia, "C-to-hdl," Online, May 2009, last saw: 10-22-2009. [Online]. Available: http://en.wikipedia.org/wiki/C_to_HDL

Personal contributions - Publications

[133] O. Brousse, J. Guillot, T. Gil, F. Grize, G. Sassatelli, J. M. Moreno, J. Madrenas, A. E. Villa, H. Volken, and M. Robert, "Jubitool: Unified design flow for the perplexus simd hardware accelerator," in *IEEE Congress on Evolutionary Computation (IEEE CEC 2009)*. IEEE, May 2009.

[134] O. Brousse, J. Guillot, G. Sassatelli, T. Gil, and M. Robert, "A bio-inspired agent framework for hardware accelerated distributed pervasive applications," in *NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2009)*, July 2009.

[135] O. Brousse, G. Sassatelli, T. Gil, Y. Guillemenet, M. Robert, L. Torres, and F. Grize, "Baf: A bio-inspired agent framework for distibuted pervasive applications," in *GEM'08, Las Vegas*, July 2008.

[136] O. Brousse, G. Sassatelli, T. Gil, Y. Guillemenet, M. Robert, L. Torres, E. Sanchez, J. M. Moreno, A. Villa, H. Volken, A. Napieralski, and E. Lavarec, "The perplexus project," October 2007, dASIP Workshop, Grenoble.

[137] O. Brousse, G. Sassatelli, T. Gil, M. Robert, and L. Torres, "Perplexus project," Jun 2007, national SoC-SiP Research Group conference.

[138] O. Brousse, G. Sassatelli, T. Gil, M. Robert, L. Torres, and F. Grize, "Distributed pervasive phylogenetic application using a bio-inspired agent framework," Jun 2008, national SoC-SiP Research Group conference.

[139] O. Brousse, G. Sassatelli, T. Gil, M. Robert, L. Torres, F. Grize, E. Sanchez, A. Upegui, and Y. Thoma, "The perplexus programming framework combinning bio-inspiration and agent-oriented programming for the simulation of large scale complex ystems," in *ICES'08, Prague*, September 2008.

[140] O. Brousse, G. Sassatelli, T. Gil, M. Robert, L. Torres, E. Sanchez, J. M. Moreno, A. Villa, H. Volken, A. Napieralski, and F. Mondada, "Perplexus project modeling framework," in *ICESCA'08, Tunis*, May 2008.

[141] O. Brousse, G. Sassatelli, and F. Grize, "The jubi approach: a tool set for hardware acceleration in sensor network applications field," in *5th International Conference on Ph.D. Research in Microelectronics & Electronics*, July 2009.

[142] J. Iglesias, V. Shaposhnyk, O. Brousse, T. Gil, G. Sassatelli, and A. E. Villa, "Perplexus: a simulation environment applied to large scale spiking neural networks," in *Federation of European Neuroscience Societies (FENS Forum)*, 2008, geneva (CH).

[143] G. Sassatelli, "Bio-inspired systems: self-adaptability from chips to sensor-network architectures," in *ERSA'09, Las Vegas*. WorldComp, July 2009, acknowledged active participation in this Invited paper.

Index

A

Abstract Data Type 42
ACC . . . . . *see* Agent Communication Channel
Ad hoc On-demand Distance Vector 69, 70, 144
Ad-hoc On-demand Distance Vector 68, 69
ADT . . . . . *see* Abstract Data Type
Agent Communication Channel 59
Agent IDentifier 60, 73
Agent Management System 59
Agent Platform 58, 59, 73, 74
Agent-Oriented Programming 13, 14, 47
Agent-oriented programming 46, 47
AID . . . . . *see* Agent IDentifier
ALU . . . . . *see* Arithmetic and Logical Unit
AMS . . . . . *see* Agent Management System
AODV . . . . . *see* Ad-hoc On-demand Distance Vector
AOP . . . . . *see* Agent-oriented programming
AP . . . . . *see* Agent Platform
Arithmetic and Logical Unit 51, 52

B

BAF . . . . . *see* Bio-inspired Agent Framework
Banker's algorithm 29
Bio-inspiration 47, 50
Bio-inspired . . . . . *see* Bio-inspiration
Bio-inspired Agent Framework 55, 62, 65, 66, 73, 82, 93, 95

C

Colibri 50

Complex Systems 19
Cougar 42
CUDA 30

D

DF ..... *see* Directory Facilitator
Directory Facilitator 59
DSR ..... *see* Dynamic Source Routing
Dynamic Source Routing 69

E

Epigenesis 18, 48
Epigenetic ..... *see* Epigenesis
ETX Link Quality 70

F

Field Programmable Gate Array 48
FIPA ..... *see* Foundation for Intelligent and Physical Agents, 59
FIPA OS ..... *see* Foundation for Intelligent and Physical Agents
Foundation for Intelligent
and Physical Agents 58
FPGA ..... *see* Field Programmable Gate Array

G

General Purpose on Graphics Processing Units 26, 30
General Purpose Processor 22, 23, 26
Genetic Algorithm 18
Global Positioning System 39
GP-GPU ..... *see* General Purpose on Graphics Processing Units
GPP ..... *see* General Purpose Processor
GPS ..... *see* Global Positioning System
GPU ..... *see* Graphics Processing Units
Graphical Processing Unit 35
Graphical User Interface 60, 62
Graphics Processing Units 26
GUI ..... *see* Graphical User Interface

H

Hardware 16

HW ..... *see* Hardware

J

JADE 57, 59, 62, 73, 96
Java 57
java 57
Java for Ubiquitous 101–103
JIT ..... *see* Just In Time compilation
Jubi ..... 100, *see* Java for Ubiquitous
Just In Time compilation 44

L

Lookup Table 51, 52
LUT ..... *see* Lookup Table

M

Macrocell 52, 107
Message Passing 34
Message Transport Protocol 72
Middleware 41, 44
MILAN 42
Mires 41
MMX ..... *see* MultiMedia eXtension
Mobile Ad-hoc Network 62, 68, 69, 143, 150
Mote ..... *see* Sensor Network
MP ..... *see* Message Passing
MTP ..... *see* Message Transport Protocol
MultiMedia eXtension 23
Multiple Instruction Multiple Data 23, 24
MW ..... *see* Middleware

N

nameservice 71

O

Object Oriented Programming 29
Object-Oriented Programming 13
Object-oriented programming 46
OLSR ..... *see* Optimized Link State Routing

OLSRD ..... *see* Optimized Link State Routing
Ontogenetic ..... *see* Ontogeny
Ontogeny 18, 48
OOP ..... *see* Object Oriented Programming
OOP ..... *see* Object-oriented programming
OpneCL 30
Optimized Link State Routing 69, 70, 143–145

P

Parallel Thread eXecution 124
Perplexus 19, 20, 50–53
Phylogeny 17, 48
proactive protocol 69
PTX ..... *see* Parallel Thread eXecution

Q

QoS ..... *see* Quality of Service
Quality of Service 42

R

reactive protocol 68
Remote Management Agent 60
RMA ..... *see* Remote Management Agent

S

SECURI*food* 38, 42
Sensor Network 16, 17, 19, 36, 37, 42, 43
SIMT ..... *see* Single Instruction Multiple Thread
Single Instruction Multiple Data 23
Single Instruction Multiple Thread 122
SM ..... *see* Stream Multiprocessors
SN ..... *see* Sensor Network
Software 16
SP ..... *see* Streaming Processors
SSE ..... *see* Streaming SIMD Extension
Stream Computing 30
Stream Multiprocessors 122
Streaming Processors 122
Streaming SIMD Extension 23
superscalar 23, 24

SW ..... *see* Software
SwissQM 44
Synchronized Distributed Executive 29

T

Texture Processing Cluster 122
Thread Scheduler 122
TPC ..... *see* Texture Processing Cluster

U

Ubicell 51, 52, 107
Ubichip 20, 50, 51, 105–108, 112–114
Ubidule 19, 50

V

Very Long Instruction Word 23, 24
Virtual Machine 44
VLIW ..... *see* Very Long Instruction Word
VM ..... *see* Virtual Machine

W

Wireless Sensor Network ..... *see* Sensor Network
WSN ..... *see* Wireless Sensor Network

Z

ZebraNet 39, 43

Acronyms

A

AB Access Box. 38
ACC Agent Communication Channel. 59, 61
ADC Analog to Digital Converter. 40
ADT Abstract Data Type. 42
AER Address Event Representation. 108, 109
AID Agent IDentifier. 60, 73
ALU Arithmetic and Logical Unit. 51, 52, 107, 110, 111, 132
AMS Agent Management System. 59–61, 74
ANN Artificial Neural Network. 18, 63, 64, 66, 67, 88–90, 96, 98
AODV Ad-hoc On-demand Distance Vector. 68–70
AOP Agent-Oriented Programming. 13, 14, 46, 47, 53, 56, 57, 95, 135, 136
AP Agent Platform. 58–60, 62, 72–74, 81
API Application Programming Interface. 28, 30

B

BAF Bio-inspired Agent Framework. . . 55–59, 62, 64–67, 71–73, 76, 82, 83, 86, 93, 95, 96, 98, 101, 102, 106, 133, 136, 137
BASIC Beginner's All purpose Symbolic Instruction Code. 11
BOINC Berkeley Open Infrastructure for Network Computing. 35, 36

C

CAD Computer-Assisted Design. 11
CAM Content-Addressable Memory. 108–110
CMOS Complementary Metal Oxide Semi-conductor. 11, 128
CPLD Complex Programmable Logic Devices. 99
CPU Central Processing Units. 26, 30, 35, 36, 120, 124, 126
CSV Comma-Separated Values. 118

D

DDR3 Dual Data Rate 3. 131
DF Directory Facilitator. 59–61, 74
DMA Direct Memory Access. 25
DNA Deoxyribonucleic Acid. 13, 48
DNS Domain Name Service. 71
DR Dynamic Routing units. 51, 52
DSP Digital Signal Processor. 24
DSR Dynamic Source Routing. 69

E

E Epigenetic. 18, 48, 63–67, 85–88, 91, 93, 95, 96
ENIAC Electronic Numerical Integrator Analyser and Computer. 10, 135
EPFL École Polytechnique Fédéale de Lausanne. 137

F

FET Field Effect Transistor. 11
FIPA Foundation for Intelligent and Physical Agents. 58, 59, 62, 67, 78, 138
FORTRAN Formula Translator. 11
FPGA Field Programmable Gate Array. 48, 99, 128
FSR Fish Eye Routing. 69

G

GA Genetic Algorithm. 18, 62, 67, 96, 98
GP-GPU General Purpose on Graphics Processing Units. 26, 30, 120–122, 124, 126
GPP General Purpose Processors. 22, 23, 26, 27, 105, 108, 110, 125, 128
GPS Global Positioning System. 40
GPU Graphics Processing Units. . . 26, 30, 35, 36, 121–123, 125–128, 131, 132, 138
GUI Graphical User Interface. 11, 60, 62, 76, 77

H

H Host agent. 63, 64, 76–78, 80, 90
HEIG-VD Haute école d'ingénierie et de gestion du canton de Vaud. 50
HPC High Perfromance Computing platform. 31–33, 136
HTTP Hypertext Transfer Protocol. 72, 73
HW Hardware. 16, 20, 37, 58, 66, 79, 80, 82, 85, 101, 102, 104, 105, 115, 116, 136, 137

I

I Interface agent. 64, 82
ILP Instruction Level Parallelism. 24
IP Internet Protocol. 12, 61, 71–73
IPv6 Internet Protocol version 6. 15
ISS Instruction Set Simulator. 137
ITRS International Technology Roadmap for Semiconductors. 12

J

J2ME Java2 Micro Edition. 59
JADE Java Agent DEvelopment framework. 58–62, 71–74, 76, 79, 96, 101, 102
JIT Just In Time compilation. 44
JNI Java Native Interface. 82
JPCL Java Parallel Class Library. 35
JUBI Java for Ubiquitous. 101–103, 105, 112, 115, 116, 120, 136, 137
JVM Java Virtual Machine. 35, 44
JXTA Juxtapose. 58

L

LAN Local Area Network. 11
LANMAR Landmark Routing protocol. 69
LFSR Linear feedback shift register. 108, 110
LUT Lookup Table. 52, 87, 107

M

MADKIT Multi-Agents Developpement Kit. 57, 58
MANET Mobile Ad-hoc Network. 57, 62, 68–71
MF Move Forward. 94
MIMD Multiple Instructions Multiple Data. 23, 24, 137
MIT Massachusetts Institute of Technology. 29
MMX MultiMedia eXtension. 23
MP Message Passing. 27, 31, 32, 34, 35, 41
MPI Message Passing Interface. 29, 32, 35
MPSoC Multi-Processor System on Chip. 27, 137
MTP Message Transport Protocol. 72, 73

N

N Network agent. 63, 64, 68, 73, 74, 76–78, 85
NS2 Network Simulator. 70

O

O Ontogenetic. 18, 48, 63–67, 73, 76, 85–88
OLSR Optimized Link State Routing. 69–72, 76
OLSRD Optimized Link State Routing Daemon. 70, 72, 73
OOP Object-Oriented Programming. 13, 29, 46
OpenCL Open Computing Languge. 30
OpenMP Open Multi-Processing. 28, 29

P

P Phylogenetic. 17, 48, 62, 64–67, 85–88, 91, 93, 95, 96
PC Personal Computer. 11, 121
PCI-E Peripheral Component Interconnect Express. 26
PE Processing Element. 23, 24, 29, 104, 107, 108, 110, 111, 113, 114, 116, 122, 128, 130–132
Perplexus PERvasive computing framework for modeling comPLEX virtually Unbounded Systems. . . 9, 17, 19, 20, 48, 50, 52, 53, 55, 57, 62, 64, 66, 79, 80, 82, 95, 98, 100, 105, 107, 119, 136, 137
PLC Power Line Communication. 15
POE Philogeny Ontogeny Epigenesis. 18, 47, 50, 56, 58, 83, 88, 96
PTX Parallel Thread eXecution. 124, 138

R

RISC Reduced Instruction Set Computer. 107
RMA Remote Management Agent. 60, 62
RMI Remote Method Invocation. 34, 35
RNG Random Number Generator. 110
RREP Route Reply. 69
RREQ Route Request. 69
RTL Register Transfer Level. 118
RTOS Real Time Operating System. 34
RU Relay Units. 38

S

S Spy agent. 63, 64, 81, 82
SIMD Single Instruction Multiple Data. 23, 24, 52, 66, 80, 82, 100, 104–108, 111–114, 116, 118, 121, 122, 128, 129, 131–133, 137
SIMT Single Instruction Multiple Thread. 122, 132
SIP System In Package. 13
SM Stream Multiprocessors. 122, 123
SN Sensor Network. 16, 17, 19, 36–38, 41–44, 46, 78, 98, 135, 136
SNN Spiking Neural Network. 100, 107, 109, 120, 125, 126, 129, 130

SP Streaming Processors. 122, 123
SpiNDeK Spiking Neuron Development Kit. 118
SR Self Reconfiguration units. 51, 52
SRAM Static Random Access Memory. 131
SSE Streaming SIMD Extension. 23
SU Sensor Unit. 38
SW Software. . . . 16, 20, 23, 29, 37, 43, 48, 58, 64, 66, 79, 80, 82, 85, 101, 102, 105, 115, 116, 136, 137
SwissQM Scalable Wireless Sensor Query Machine. 44
SynDEx Synchronized Distributed Executive. 29

T

TCP Transmission Control Protocol. 61, 71, 72
TL Turn Left. 94
TLP Task Level Parallelism. 24
TPC Texture Processing Cluster. 122
TR Turn Right. 94
TUL Technical University of Lodz. 51

U

U Ubicom agent. 64, 82, 86
UMTS Universal Mobile Telecommunications System. 15
UPC universitat politècnica de catalunya. 50, 107
UPnP Universal Plug and Play. 46

V

VHDL VHSIC Hardware Description Language. 99, 100, 118, 119
VLIW Very Long Instruction Word. 23, 24
VM Virtual Machine. 44

W

WIMAX Worldwide Interoperability for Microwave Access. 15
WSN Wireless Sensor Networks. 36

An Agent-Based and Bio-Inspired Programming Environment for Pervasive Platforms

Abstract in English

Ubiquitous Computing is the emerging trend in computing systems. These pervasives platforms have characteristic constraints (environmental and architectural). Such characteristics clearly impact the way pervasive platforms should be programmed. Base on Sensor Network programming experience an agent oriented and bio-inspired approach is proposed as a reliable programming framework for pervasive platforms.

Nevertheless, how to improve performance of such platforms made of modules with small computing resources. The use of parallel coprocessors is then considered. Such distributed and locally hardware dependent programming does not ease platform programming. Taking advantage of Java's portability and of the proposed framework flexibility a coprocessor programming solution is proposed.

These contributions are basic blocks of a pervasive platform programming environment that has been tested on the Perplexus platform.

Résumé en Français

La tendance marqu/'ee de ces dernières années au niveau des systèmes informatiques est à l'Ubiquitous Computing. Ces plateformes pervasives ont des contraintes (environnementales et architecturales) qui leurs sont propre et influence la manière de les programmer. En se basant sur l'étude des réseaux de capteurs une approche orientée agents et bio-inspirée est proposée pour programmer les plateformes pervasives en assurant leur adaptabilité dans un environnement non maitrisé.

La question se pose de savoir comment améliorer les performances de ce type de plateforme construites à partir de modules possèdant peu de ressources de calcul. L'emploi de coprocesseurs parallèles est envisagé. L'utilisation de matériel spécifique à l'un ou l'autre des modules de la plateforme pose des problèmes de facilité de programmation. Une solution de programmation adaptée est proposée tirant partie de la portabilité de Java et de la flexibilité de l'environnement proposé précédemment.

L'ensemble de ces deux contributions qui forme un environnement de programmation pour plateformes pervasives, a été testé sur la plateforme Preplexus.