Architecture design of a virtualized embedded system

A.Sbaa[1], R. El Bejjet[2] and H.Medromi[3]

Abstract - Nowadays, embedded systems have become a major driver of technological developments particularly in the industrial sector. In an embedded system, the hardware and software components are so intertwined that evolution is often a big issue. Thanks to the contributions of virtualization it is possible to design more flexible and scalable embedded systems. In fact, virtualization adds an abstraction layer between the hardware and the software to make it less intimately dependent. In this paper we propose system architecture based on embedded virtualization that will be modeled by the multi-agent systems. In the first part, we will discuss the state of the art with respect to embedded systems, multi-agent systems and the concept of virtualization. In the second part, we explain the issues related to traditional embedded systems and the constraints associated with their architecture. Subsequently, we propose a virtualization based architecture for embedded systems. In the last part we will showcase this architecture through a prototype.

Keywords: virtualization, embedded systems, multi-agent system, Linux KVM.

I. Introduction

In recent years we are witnessing a technological breakthrough with the classical model of infrastructure. With the arrival of virtualization, infrastructure concepts have so profoundly evolved as to be set as standard planetary clouds called "cloud computing." In addition to cost advantages and deployment systems, virtualization offers a flexible architecture based on an abstraction layer between the hardware and the software layers. Embedded systems whose major issue is the relationship between hardware and software can also benefit from this architecture to avoid designing embedded systems which are difficult to develop and maintain.

In this paper, using the concept of virtualization we will explain how to design an embedded system enjoying all the benefits and contributions offered by virtualization. The first part is devoted to the state of the art in this area, the second one presents issues related to traditional embedded systems. In the third part we discuss the design of the proposed architecture based on multi-agent systems. The last part is devoted to the presentation of the prototype that we carried out to illustrate the concept.

II. State of the Art

In this part, we will give an overview of the state of the art in terms of multi-agents systems, embedded systems and virtualizations techniques.

II.1 Multi-Agents Systems

The multi-agent systems (MAS) and autonomous agents provide a new method for analyzing, designing and implementing sophisticated applications because they are part of the IAD domain (Distributed Artificial Intelligence) also benefiting other disciplines such as cognitive science sociology, and social psychology.

An agent can be seen here, as any entity, hardware or software which perceives its environment using sensors and act on it using actuators.

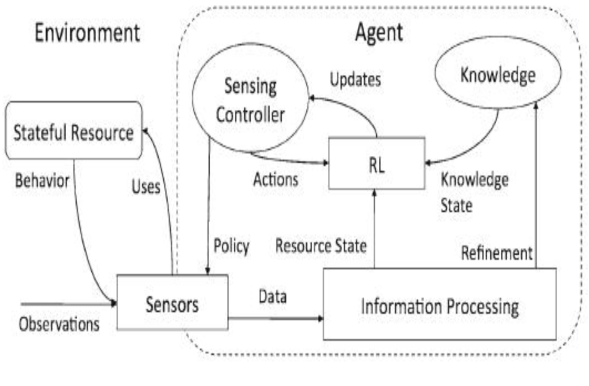

Figure 1. General architecture of an agent in its interaction with its environment

Today, most applications require distribution of tasks between "entities" autonomous (or semiautonomous) to achieve their objectives in an optimal way. Since traditional approaches are generally monolithic and their concept of intelligence is centralized, current applications are established based on multi-agent system.

II.2 Embedded systems

Embedded systems are devices or software components and hardware being intimately linked. The interest in embedded systems is becoming more commonplace with the development of means of communication (Smartphone, switch-boards, etc...) as well as the permanent need for life easing through technological means (MP3 player storage data). In Industry, embedded systems are often a necessary choice for reasons of criticality of safety features and underlying risks (ABS braking system, alarm and detection, aviation, etc ...). Designed to precise spots, embedded systems bring several advantages by providing traditional systems based on conventional computers. The main constraints to meet embedded systems which are:

- System stability: an embedded system is often dedicated to a precise running, malfunctions must be mastered.
- Mastery of the security, integrity and access: with a minimum of features and services enabled, an embedded system is designed to be safer.
- The cost of production: a computer system can be produced through industrial processes that greatly limit the cost of production.
- Low energy consumption: Unlike a conventional computer, an embedded system has the minimum resources tailored. Energy consumption is adjusted and optimized.
- Responsiveness or response time required in an embedded system often require real-time systems.
- Autonomy: Embedded systems must operate without human intervention to fulfill automatic tasks.

In the Models design of embedded systems there are two design approaches:

- Approach based on the classic design: the design of software and hardware components are made by the same engineering team. This approach can lead to proprietary systems and generally not scalable.

- Approach based on CODESIGN: the hardware and software components are designed by separate teams. The component integration is being done at the end by a joint team. This approach allows to validate many different components before integrating them, giving rise to scalable embedded systems.

II.3.Virtualization

Virtualization is all the technical hardware or software that allows to operate on a single machine multiple operating systems or multiple applications.

Virtualization complies with two fundamental principles that are the partitioning and transparency.

Partitioning: each operating system operates in independency, and cannot interfere with the other in any way.

Transparency: the act of running in virtualized mode does not change the operation of the operating system and applications.

There are three main types of virtualization solutions that are insulation, paravirtualization and full virtualization.

Insulation: Insulation is a technique that takes place in a single operating system. It allows a system to separate multiple contexts or environments. Each of them is governed by the host OS, but programs of each context are able to communicate only with the processes and resources involved in their own context.

It is thus possible to partition a server into several dozen contexts, without significant slowdown. The insulation is used on Unix systems for protection reasons. Through mechanisms such as chroot or jail it is possible to run applications in an environment that is not the host system, but a mini system containing only what the application really needs, and having only access limited resources. It is also possible to run programs in a different distribution than the main system.

With insulation, kernel space is not differentiated; it is unique, shared between contexts. But defines multiple user spaces partitioned. Thus we can make possible the coexistence of different distributions of the operating system, as long as they share the same kernel. The contexts insulation is a lightweight solution, especially in Linux environments. For the most common needs of virtualization, ease of implementation and low overhead are excellent arguments.

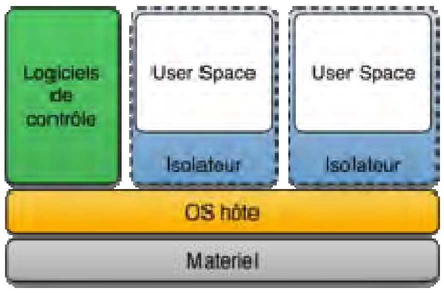

Figure 2. virtualization for isolation

Paravirtualization: Paravirtualization is based on a hypervisor layer, which manages the interface completely with the material, and on which you can install different operating systems. Paravirtualizationoffers to an operating system a generic special system, which therefore requires special interfaces integrated guest systems in the form of drivers. Paravirtualization is a virtualization technique of lower level than the insulation. It shares with it the need to use a modified OS. More specifically, paravirtualization is no longer only the host OS to be changed but also the OS required to run on virtual environments. The heart of paravirtualization hypervisor is running close to the hardware, and provides an interface that allows multiple hosts to concurrently access to resources. Each virtual machine must be modified to use this interface to access the hardware. Unlike the isolation of several different families OS can run on a single physical server. It is possible to run Linux, NetWare, Solaris (and others) simultaneously on the same machine. Each OS will have access to their own storage devices, its own memory, her or its own network interfaces or its own processors; each virtualized hardware resource is shared with other environments. The need of small changes to the guest operating system does not support closed systems, especially Microsoft Windows.

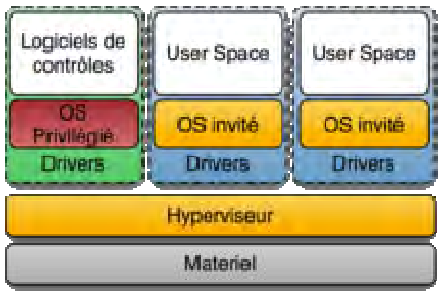

Figure 3. paravirtualization architecture

Full virtualization: Full virtualization is an operating system that runs software called hypervisor. The hypervisor will allow running multiple virtual machines on the physical machine. It manages memory access, allocation of CPU and all the necessary resources to virtual machines. In full virtualization, the hypervisor manages all requests for virtual machines that allow virtual machines to run without any change in their core. In other words, the virtual machines do not even know they run in a virtual manner.

The hypervisor: The hypervisor is the software layer that fits between the hardware and different operating systems. It is a key component, found in most low-level virtualization technologies. Thus, compared to the base schema of a distinguished server hardware, operating system, and applications, the hypervisor can be self-manage all the hardware resources of the server, or relies on it for an existing operating system.

Emulation: The emulation consists in simulating the execution of a program by interpreting each of the instructions for the microprocessor. It is thus possible to emulate any processor and the complete environment. Emulation is the technique that provides the highest level of abstraction of the platform. For other virtualization techniques all executable must be compiled for the processor physically available on the server.

Emulation removes this constraint because the instructions are never executed by the processor; they are interpreted by simulating the processor. This interpretation is costly in performance, so that the emulation is rarely used outside of game or research applications. The project QEMUis an open source solution for virtualization emulation.

Fields of application: Virtualization allows to implement virtual dedicated servers (VDS) to provide accommodation for resource sharing, autonomy and full control of the server unlike traditional offerings that are dedicated hosting server (available servers configured as appropriate) and shared hosting (the same server serves multiple clients, no resource control, limited power). Virtualization also acts to improve the availability of servers or services. We talk about load balancing and failover. Virtualization has helped democratize Appliances (routers, firewalls, security solutions ...).

Full virtualization can reduce costs (equipment and operations) for development and testing. On a single physical machine, you can install multiple operating systems to test different scenarios.

III. Problem related to the conventional embedded system

In any architecture of embedded system, the hardware and software components are intimately linked. This linkage is in many cases a major drawback for many reasons:

III.1.Constraint related to the software update of an embedded system

In the case of major software update of embedded system, it is rarely possible to keep the same hardware because:

The software update requires more performance and current hardware is insufficient.

The new software libraries require new hardware for compatibility reasons

III.2 Hardware related constraint

In most embedded systems, the introduction of new features requires a hardware change by adding or replacing the entire embedded system. Consider the case of a car which is not provided with a level sensor. The onboard computer requires a software update for processing data related to the sensor. Obviously, the embedded system must be equipped with a slot for data acquisition from the level sensor. If this interface is not included in the design of embedded system, we may have to completely change the onboard computer.

III.3.Hardware Design

When designing the embedded system hardware specifications are provided at the beginning so that the material becomes owner and for a specific purpose. Sometimes for the same feature is specific equipment from one manufacturer to another (eg on-board computer of a car).

It should therefore be noted that the development of an embedded system is determined by the flexibility of the dependence between the hardware and the software part. In most cases this interdependence is so strong that the evolution is to change the entire embedded system. This prevents the reuse and recycling systems embedded with all issues related to the environment and economic waste.

In the next section, we propose a solution to this problem by introducing the virtual architecture of embedded systems.

IV. About the proposed solution

IV.1 Modeling ADMs

In order to better understand the problem we have opted for a model of our architecture using multi-agent systems.

IV.1 .1 case of classic embedded system

We will consider the case of an embedded system whose functionality is the data acquisition through sensors, their processing and visualizing data on a screen.

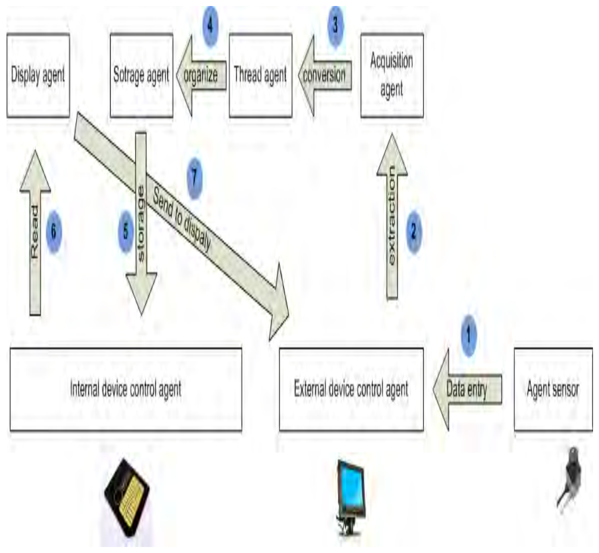

Figure 4. Classical architecture of an embedded system

In classical architecture of embedded systems, data acquisition from the sensor agent is performed by an agent of external devices control. Subsequently acquisition agent will convert the extracted data in data recognized by the processing agent who will be responsible to organize and communicate with the storage agent to be stored on an internal memory. Data storage will be used to display historical data or comparison with previous data to detect the distance (eg average speed for a period of one hour, each 5 seconds to store a value to be able to calculate the average). Internal device control is through control agent. To display the values on the screen, a display agent must read the data stored on the internal memory. It is important to note that in this architecture the information processing agents and devices control agents communicate directly.

In an improved architecture, we can add a layer called virtualization layer that will be able to communicate with all internal and external devices.

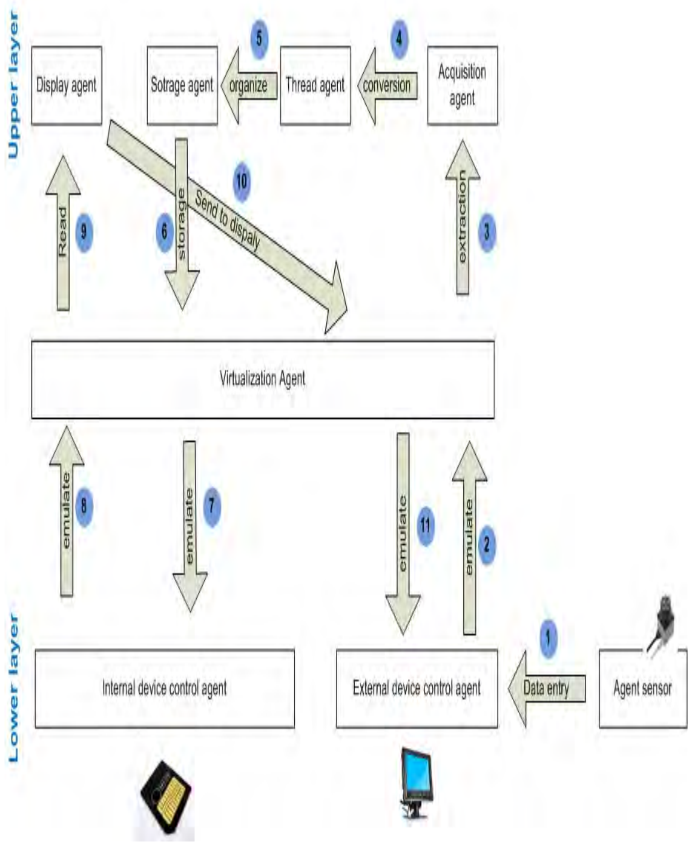

Figure 5. improved architecture of an embedded system

In this new architecture, all applications for writing, reading and data acquisition are performed by the virtualization agent who is the monitor of internal and external control devices agents via emulation interfaces. Agents in the top layer never communicate with the agents of the base layer. If the virtualization agent remains unchanged, the behavior of agents of the top layer does not change however. A modification of the lower layer does not impact the higher one since we keep the virtualization agent. It is clear that in practice if we change a material in the lower layer, we can keep the same operating system without changing it completely. This brings us later to ask about the methodology of designing an embedded system. Should we design the hardware before the software at the same time or separately? We will get back to this issue after discussing the architecture of the embedded system based on virtualization.

IV.1 .2.Description of the architecture of the virtual embedded system

Virtualization layer:

The proposed architecture of the embedded system based on adding a virtualization layer that will have the role of monitoring hardware devices.

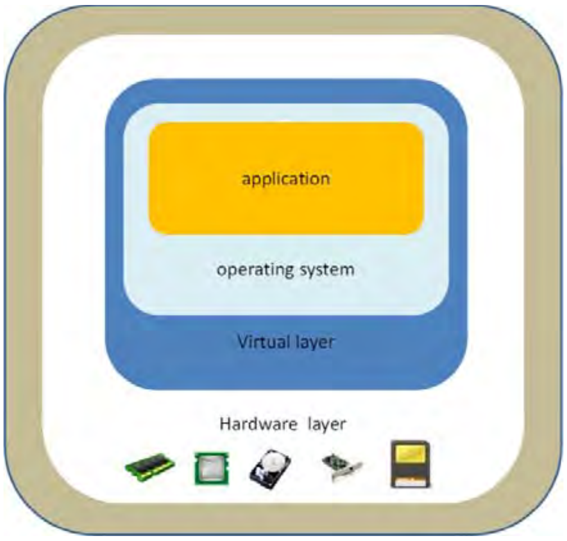

Figure 6. Virtual architecture of the embedded system

In this new architecture, applications and the operating system of the embedded system do not communicate in any way with the hardware devices. Control equipment is provided by the virtualization layer that has the operating system standard interfaces. In the case of internal memory, the virtual layer emulates a virtual internal memory and reading and writing operations are performed through the virtual layer. It is possible with this architecture to emulate all types of internal and external devices (network card, sensor, hard disk etc. ....). The major challenge comes in developing this virtual layer that will manage the interfaces with all types of equipment an embedded system may use.

IV.2 Technical Architecture:

In the following diagram we explain the technical details of our architecture:

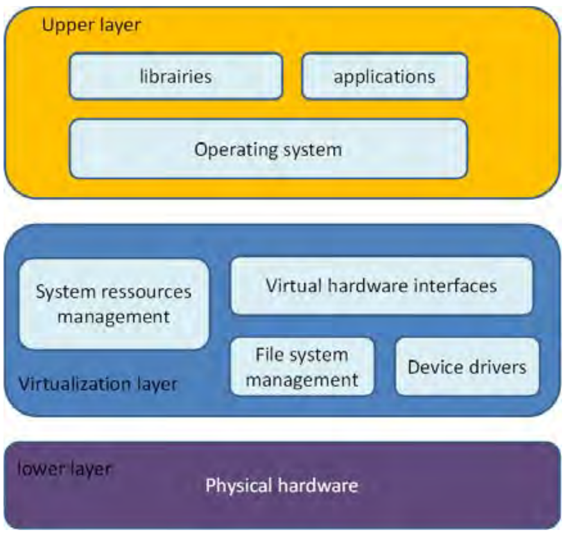

Figure 7. Technical architecture of the visualization layer

In this architecture, the virtualization layer consists of an operating system virtualization with the main objective:

Managing access to physical memory

Management hardware drivers

Use management of system resources (memory, CPU, etc. ...).

Emulation of all hardware devices via virtual interfaces

The operating system at the top layer will be used only for the execution of processes embedded systems. Applications and libraries depend only on the operating system of the top layer.

In this architecture, if you change equipment at the lowest layer, you just need to adjust the virtual interface at the virtualization layer. The change is usually done by adding the new hardware driver in the operating system of virtualization. On the other hand, the operating system of the top layer will remain unchanged.

V. Prototype implementation:

To illustrate our technical architecture, we built a prototype of a virtualized embedded system. We opted for the following technical components:

Hardware Layer :

- 1.6Ghz Intel Atom E6xx single chip processor companion chip with EG20T
- 512Mbyte DDR2-SDRAM, soldered on board
- 8 Mbit BIOS / BOOT Flash
- Internal Low Profile USB socket, bootable
- 2x SATA 3Gbit interfaces with +5 V and +12 V power header
- 4x Intel 82574L Gigabit Ethernet ports, Auto-MDIX RJ-45, protected to 700W/40A Surge
- 2x Serial ports, DB9 and 10 pins internal header
- USB 2.0 interface, 2x internal, 1x external port, bootable
- Power LED, Disk LED, Error LED, Status LED, Network LED's
- 1 Full Mini-PCI Express shared with mSATA socket.
- 1 USB only Mini-PCI Express shared with mSATAsocket
- 2x PCI Express Slots, right angle

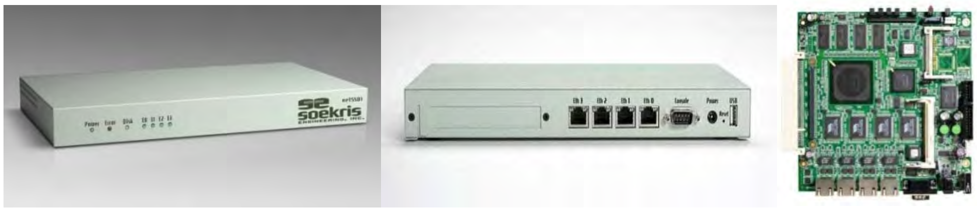

Figure 8. Technical Components used in the visualization layer

Virtualization layer:

The choice is centered on the nucleus linux KVM as a derivative of QEMU project. KVM is basically integrated in the Linux distributions with a kernel 2.6.20 or higher. Due to its flexibility Kernel Based Virtual Machine became the virtualization solution for Linux reference. Several hardware architecture includes hardware-level extensions based on KVM virtualization (Intel VT ™, AMD-V ™). These two families of processors have hardware virtualization extensions. KVM is composed of a core module and a client derived from the QEMU project.

KVM architecture is as follows:

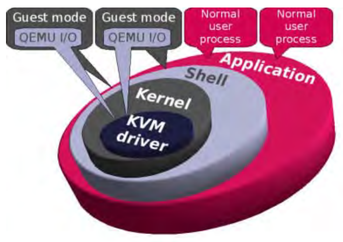

Figure 9. KVM architecture.

For the OS of the top layer we opted for the OS OPENBSD which has been used as part of another task and that runs an application based on PERL libraries.

We loaded the kernel on KVM flash memory box Soekris 6501. The OpenBSD operating system is loaded from an ISO image.

We used a USB external GSM modem which is emulated via KVM. PERL libraries that run at the OPENBSD are questioning the modem via AT commands. The interface of the modem at the KVM system is seen as a serial port. At OPENBSD we emulated serial port modem GSM. When changing the external USB modem, you just have to adjust the level KVM kernel to recognize the new serial port.

VI. Benefits of this architecture in the design of embedded systems

In the design of embedded systems, it is possible to add the virtualization layer at the hardware layer. All internal and external devices are recognized by this layer and emulated operating systems of the top layer. Thanks to this architecture, it is possible to separately design the hardware and the software part of embedded system to finally break with the old model that remained for many years an obstacle to the development of embedded systems in the industry.

It is also possible to design a virtual model of embedded system so that all manufacturers comply with this standard and avoid wasting huge efforts in the hardware design and focus on the design and functionality of software systems on board. It is also possible to use an embedded system regardless of manufacturer and functionality seen that this standard will bear the majority of internal and external devices through a simple adjustment. We can imagine a TV receiver that will serve as the dashboard of a car or a moisture sensor controller as a controller level. This will reduce industrial waste which is a major cause of environmental problems.

VII. Conclusion and further work

In this paper we discussed one of the fundamental principles of embedded systems design that remained for a long time a drawback to the development of embedded systems. We have designed an architecture based on virtualization layers associated with low control of internal and external devices. We were able to implement a prototype based on the Linux kernel KVM. This allowed us to conclude that it is possible with this architecture to benefit from the contributions of virtualization in embedded systems, mainly:

- Separation between the hardware and software aspects in the design of embedded systems
- Flexibility in terms of hardware and software developments.
- Reuse of embedded systems
- Portability of applications and features of embedded systems.

It is clear that the future of embedded systems must address the implementation layers of virtualization at the hardware level as well as standardization of these layers.

References

[1] Karim Yaghmour « Building Embedded Linux Systems » ; O'Reilly 2003.
[2] Alessandro Rubini et Jonathan Corbet, Linux Device Drivers; 3ème edition ; O'Reilly 2005.
[3] Evi Nemeth et al, Linux Administration Handbook; Prentice Hall ; 2002
[4] Karim Yaghmour, Building Embedded Linux Systems; O'Reilly 2003.
[5] Craigh Hollabaugh, Embedded Linux; Sams 2002
[6] "Smartphone definition from PC Magazine Encyclopedia". PC Magazine. http://www.pcmag.com/encyclopedia_term/0,2542,t=Smmrtphone&i=51537,00.asp . Retrieved 13 May 2010.
[7] Drogoul A., De la simulation multi-agent à la résolution collective de problèmes, Thèse de doctorat, Université Paris 6, 1993.
[8] Guessoum Z., Un environnement opérationnel de conception et de réalisation de systèmes multi-agents, Thèse de doctorat, Université Paris 6, mai 1996.
[9] Carabelea C., Boissier O., Florea A., «Autonomie dans les systèmes multiagents », JFSMA '03, 2003.
[10] J. Ferber: Les systèmes multi-agents, vers une intelligence collective, Paris, InterEditions, 1995.
[11] R. El Bejjet, H. Medromi, « A Generic Platform for a Multi-Agent Systems Simulation », September 2010, Vol. 5. n. 5, pp. 505-509.
[12] Dave Pfaltzgraff, The Use of Linux in an Embedded System, December 1999,*Linux Journal*. Putting together a Card Access System.http://www.linuxjournal.com/lj-issues/issue68/3555.html
[13] Sam Williams, Open Season: Embedded Linux Comes to the Fore, August 17, 2000, *Upside Today*. The San Jose Linux World Conference and Expo activity is impressive. http://www.upside.com/Open_Season/399c565b0.html
[14] Stan Runyon, Linux Catches Eyes in Test World, January 4 2000, *EE Times*. National Instruments implement modular automated equipment with Linux. http://www.eet.com/story/OEG20000104S0027
[15] Doc Searls, The Next Bang: The Explosive Combination of Embedded Linux, XML and Instant Messaging", , September 2000, *Linux Journal*, http://www.linuxjournal.com/lj-issues/issue77/4195.html
[16] Embedded Systems Programming Magazine : http://www.embedded.com/mag.htm
[17] D. Kalinsky, R. Kalinsky ; « Introduction to I2C », Embedded.com. 2001. http://embedded.com/story/OEG20010718S0073
[18] M. Khemakhem, A. Belghith, « Agent Based Architecture for Parallel and Distributed Complex Information Processing », January 2007, Vol. 2. n. 1, pp. 38 - 44

Authors' information

1A. Sbaa is an engineer in computer science since 2002, from EMI an engineering school in Rabat, Morocco. In 2009 he got a Certificate of Information System Auditor (CISA). He joined the system architecture team of the ENSEM, Casablanca, Morocco.
As a Ph.D. Student since 2009, his current and previous research interest is about intelligent systems architecture based on Multi Agents Systems especially for mobile communication.

2Rachid El Bejjet is an engineer in computer science since 1995, from ENIM an engineering school in Rabat, Morocco. In 2009 he got his Master Business of Administration (MBA) in ENPC, Paris, France. In 2009 he joined the system architecture team of the ENSEM, Casablanca, Morocco. His actual main research interest is about platforms of Multi agents Systems' simulation and implementation.

3Hicham Medromi received the PhD in engineering science from the Sophia Antipolis University in 1996, Nice, France. He is responsible of the system architecture team of the ENSEM Hassan II University, Casablanca, Morocco. His actual main research interest concern Control Architecture of Mobile Systems Based on Multi Agents Systems. Since 2003 he is a full professor for automatic productic and computer sciences at the ENSEM, Hassan II University, Casablanca, Morocco.