eEPC: an EPCglobal-compliant Embedded Architecture for RFID-based solutions

F. Fummi
Department of Computer Science, University of Verona, Verona ,Italy
Email: franco.fummi@univr.it

G. Perbellini
EDALab S.r.l., Verona ,Italy
Email: giovanni.perbellini@edalab.it

***Abstract*— Radio Frequency Identification (RFID) technology has a lot of potential to improve visibility across the supply chain and automate the business processes.**
This paper describes an EPCglobal-compliant embedded architecture, called *eEPC*, aiming at providing RFID-based solutions for networked embedded systems. The design issues related to the EPCglobal architecture porting on the embedded systems field have been analyzed. These problems concern the role of the operating system adopted by the embedded system, the SW architecture implementation (Single-Thread or Multi-Thread) to satisfy the EPCglobal Architecture constraints and the HS/SW partitioning alternatives. The proposed design alternatives exploration allows to evaluate which *eEPC* implementation, running on different HW/SW platforms, better satisfies the multiple tag readings and processing architectural constraints. Its effectiveness has been evaluated on a real application where products information is automatically put in relation to user's medical data to retrieve information able to improve the quality of life.

***Index Terms*—Networked Embedded Systems, RFID, EPC, EPCglobal, supply-chain.**

I. INTRODUCTION

Radio Frequency Identification (RFID) technology has recently seen growing interest not just from the research community [1], but also from a wide range of industries such as retail, pharmaceutical, and defense. RFID is a generic term used to describe a system that transmits the identity (in the form of a unique serial number) of an object or person wirelessly, using radio waves [2]. It is grouped under the broad category of automatic identification (auto-ID) technologies. Auto-ID technologies include bar codes, optical character readers and some biometric technologies, such as retinal scans. The auto-ID technologies have been used to reduce the amount of time and labor needed to input data manually and to improve data accuracy.

In traditional RFID applications, such as access control, there was little need for an RFID middleware because the RFID data were only consumed by a single application. In novel application domains, such as supply chain management and logistics, to support high-volume data management the companies need an infrastructure software (middleware) that has to be integrated into the whole enterprise business system. In this sense, RFID integrated into a IT infrastructure can help businesses, improve their bottom line and gain control over business processes through improved visibility, increased efficiency, and enhanced productivity [3]. This introduces the need for an RFID infrastructure that hides proprietary reader device interfaces, provides configuration and system management of the reader devices and filters and aggregates the captured RFID data. The result is that applications no longer need to maintain connections to individual reader devices or even need to know how to trigger a read cycle at a particular RFID reader device.

When combined with wireless communication, these tags and sensors enable objects communicating directly with customers, suppliers, employees-and even each other-to create business value. RFID technologies allows to create an identification global system for the products tracing based on open-standards, using Internet as a infrastructure, to facilitate the products information exchange and to improve the product management in the supply chain. A networked RFID solution allows to link product identity information from a tag to other information stored on networked databases [4].

In 1999, the Uniform Code Council and EAN International teamed with Gillette and Procter & Gamble to fund the Auto-ID Center at the Massachusetts Institute of Technology. The center changed the equation by working with private industry to develop an RFID tag that would be very low cost (the goal was five cents) when manufactured in high volumes. That way, companies could put tags on everything they own and then connect them to the Internet through a secure network. The Auto-ID Center's contribution went beyond trying to create an inexpensive tag. It developed the

Electronic Product Code (EPC), a numbering scheme that makes it possible to put a unique serial number on every item manufactured. It developed a way for tags and readers to communicate (the air interface protocol) and designed a network infrastructure that stores information in a secure Internet database. A virtually unlimited amount of data associated with a tag's serial number can be stored online, and anyone with access privileges can retrieve it.

The EPC Network, originally proposed by the Auto-ID Center has been called EPCglobal Architecture [5] when it has been further developed by the members of EPCglobal which has defined a second-generation air interface protocol and has developed the network infrastructure. This is currently the predominant standardization effort of the RFID community.

The EPCglobal Architecture framework enables companies to share data in real time; it represents a collection of interrelated standards for hardware, software, and data interfaces, all in service of a common goal of enhancing the supply chain through the use of Electronic Product Codes (EPCs) [6]. Therefore the EPCglobal Architecture is a set of technologies that enable immediate, automatic identification and sharing of information on items in the supply chain [7]. In that way, the EPCglobal Architecture will make organizations more effective by enabling true visibility of information about items in the supply chain.

The EPCglobal Architecture consists of a number of roles and interfaces that need to be deployed within a company in order to process EPC tags in an EPC-compliant way [8]. The EPCglobal Architecture specifications do not define the individual components, but rather roles and interfaces that must be implemented. These roles (in bold type) and interfaces (in italic type) are illustrated in Figure 1 and explained in the following:

1. The RFID reader is responsible for interacting with EPC tags within range of its antenna, on one hand, and a host application on the other hand. Communication with the latter is done through the *Reader Protocol* interface, which provides commands to inventory tags (i.e., to read its EPC).
2. The RFID Middleware role is responsible for controlling the RFID readers. In particular, this role is responsible for the following tasks:
   - Reader Coordination;
   - Access Coordination for concurrent access;
   - Data Filtering and Aggregation;
   - Logical Readers (readers grouping).

   This functionality is exposed through the Application Level Events (ALE) interface, which can provide data both in a synchronous and in an asynchronous way.
3. The EPC Information Service (EPC-IS) is the main element in an EPC-enabled information system. It allows EPC related data to be shared, stored and managed within and across organizations. This task is implemented through three components: EPCIS Capturing Application, EPCIS Repository and EPCIS Accessing Application. EPC-IS includes also a component named Object Name Service (ONS) [9] which returns the address of EPC-IS server containing the requested information.

XML is used as a common language in the EPCglobal Architecture to define data exchange on the physical objects.

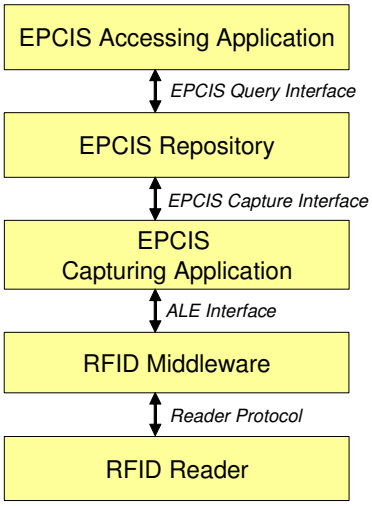

Figure 1: EPCglobal Architecture Framework: roles and interfaces.

II. MOTIVATIONS

To guarantee the use of the EPC architecture in any step of the supply-chain, using a homogenous environment, it is necessary to design an open-platform solution in order to apply the architecture to any hardware/software configuration (e.g., portable PC, Tabled-PC, Palm, touch-screen, mobile terminal, etc.).

Embedded systems are pervasive in today's world such as homes, offices, cars, factories, hospitals, plans and consumer electronics. Embedded systems consist of a microprocessor, ASICs and/or field programmable gate arrays (FPGA) as well as other programmable computing units such as Digital Signal Processors (DSPs). Moreover, since embedded systems interact continuously with an environment there must typically be some network interfaces. An important issue in the embedded systems design is the limited amount of resources of an embedded system. This leads the application developer to use ad hoc technologies to save memory usage and to meet the reduced processing power. Therefore, a significant part of the embedded systems design problem consists of deciding the software and hardware architecture for the system, as well as deciding which parts should be implemented in software running on the programmable components and which should be implemented in more specialized ad-hoc hardware.

Works about the EPC integration in the supply chain for efficient tracking of objects ([10] - [11]), describe EPC frameworks not usable on embedded systems. Other

works concern the timing analysis of networked RFID solutions [12], to meet the timing constraints set by the real-time industrial control applications [13], or propose services to facilitate the business process debugging [14].

Moreover, the state of the art provides some industrial solutions like [15] and [16]. They are typically built by using software libraries (like ASP .NET, Java Enterprise Edition) for general purpose HW architecture; however, the integration of these architectures on embedded systems remains problematic.

Such difficulties are the main motivations of this work:

- Lack of embedded open-platforms usable on all the steps of the supply chain.
- Existing of tight requirements on performance, cost and power consumption, which do not allow to implement such services by using a general purpose architecture.

In this paper we discuss a possible embedded implementation of the EPCglobal Architecture, called *eEPC*, for ubiquitous objects tracking based on RFID tags. Moreover, the paper analyzes, with respect to different HW/SW embedded platforms, the following aspects:

- Two different *eEPC* implementations (Single-Thread or Multi-Thread) to satisfy the EPCglobal Architecture constraints (in particular to support multiple-tag reading). The paper shows that the latter implementation is necessary in case of limited computational resources and/or embedded OS with high overhead to manage threads.
- HS/SW partitioning alternatives to increase the *eEPC* performance. The paper concludes that the huge effort necessary to implement in hardware an *eEPC* module does not produce a significantly gain in terms of overall speedup.

Results obtained by using the *eEPC* networked embedded architecture on real embedded platforms are used to estimate drawbacks and advantages of the proposed solution.

The *eEPC* v2.0 presented in this paper represents an enhanced and extended version of the embedded architecture described in [17] which was implemented by using the outdated architecture of the EPC Network proposed by the Auto-ID Center [18].

The paper is organized as follows. Section 3 describes the embedded EPCglobal Architecture design and its design constraints introducing the *eEPC* networked embedded architecture and its implementation for different embedded operating systems. Section 4 deals with architectural exploration and finally Section 5 presents results on the experimental analysis.

III. EMBEDDED EPCGLOBAL ARCHITECTURE DESIGN

Based on the *EPCglobal Architecture* specifications, a compliant embedded architecture, shown in Figure 2, has been designed. It is a method for using RFID technology in the global supply chain by using inexpensive RFID tags and readers to pass Electronic Product Code numbers, and then leveraging the Internet to access large amounts of associated information that can be shared among authorized users.

This networked embedded architecture, called *eEPC*, guarantees a distributed solution across sites, companies or even across countries. This feature is implemented supplying an interface, called *Remote-EPC Channel*.

eEPC integrates data from RFID-EPC devices with enterprise applications, to manage the information related to the RFID code; this integration is realized through the *Application Channel*.

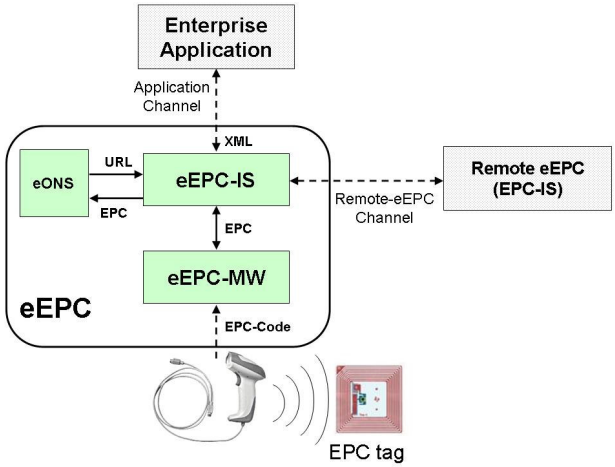

Figure 2: *eEPC* Networked Embedded Architecture.

Figure 2 shows the *eEPC* main modules: embedded EPC-Middleware (eEPC-MW), embedded ONS (eONS) and embedded EPC Information Service (eEPC-IS). After the description of these embedded components, the paper will discuss the architecture design and modularity of the embedded solution.

A. *eEPC: Architecture*

The goal of this section in to explain the role and interfaces of each component of the *eEPC* architecture. Figure 3 shows its functional modules.

eEPC Middleware

eEPC-MW, conforms to the EPC-Middleware specifications defined by the EPCglobal Inc., is the nervous system to manage the flow of information. It is the module software designed to process the streams of RFID code coming from one or more reader devices. This component manages the data read to verify the correctness of the EPC code. It includes two components: *Reader Manager* and *Application Level Event (ALE)*.

Reader Manager implements the Reader Protocol interface to manage the communication with the RFID Reader (e.g., RS232, Bluetooth, etc.).

ALE module implements data transforming operation for the encoding of RFID (64 or 96 bit) as a Uniform Resource Identifier (URI). For example, a sequence of

bits denoting an EPC code of 64 bit (8000000030000190 in hexadecimal format) is translated into the URI form (containing l'EPC Manager, the object class and the object serial number) as follows:

`urn:epc:id:sgtin:0614141:000024.400`

Moreover, it implements filtering and grouping mechanisms to recognize which RFID code to use.

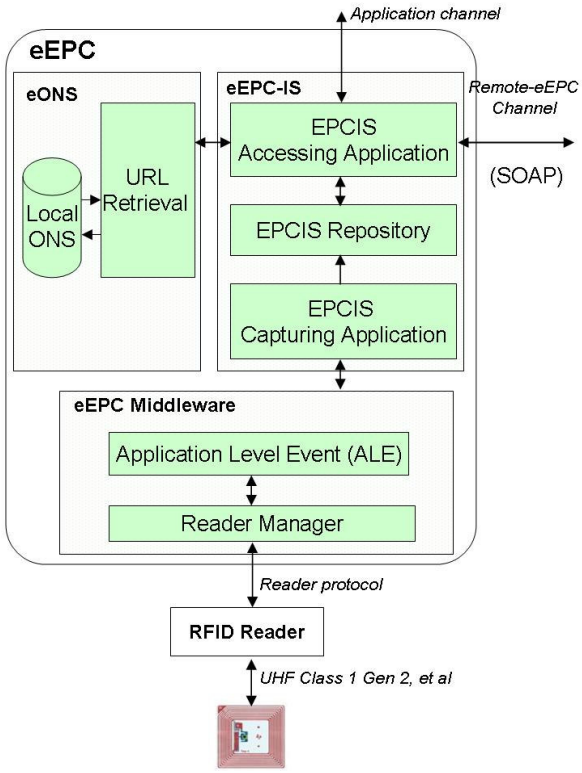

Figure 3: Detailed *eEPC* Network Embedded Architecture.

The eEPC-MW module implements the following functions to process the RFID read:

- **`getEPCcode`**: to get the streams of RFID code coming from one or more reader devices.
- **`translateEPCcode`**: to translate the streams of RFID code in the URI form `urn:epc:id:x:y:z` established by the EPCglobal Inc. Moreover this functions allows to verify the EPC code and to process it. Sometimes a tag is read incorrectly. By using this algorithm the system is able to correct these errors.

eONS

The embedded Object Name Service provides a lookup process to translate an EPC code into a Internet Uniform Reference Locator (URL) where further information about the object may be found. These URLs often identify remote EPC-enabled systems; though eONS may also be used to associate *EPCglobal Architecture* enabled system with web sites and other Internet resources relevant to an object.

eONS is built using the same technology as DNS, the Domain Name Service of the Internet, which routes information to appropriate web sites. The eONS will be used every time information is needed about a physical object. The eONS module has been implemented using an embedded Database; it implements the following functions to translate an EPC code into an Internet Uniform Reference Locator (URL):

- **`initONS`**: to define the SQL table that contains the URL related to the EPC code. It creates the local Database containing the URL related to the RFID, loading these information (URL and RFID) from a configuration file.
- **`getURL`**: returns the URL where further information about the EPC code can be found.

eEPC Information Service

The *eEPC Information Service* makes networked embedded architecture related data available in XML format (the standard language for describing physical objects) for requesting services (Enterprise Application or a remote EPC-enabled system).

EPCIS Capturing Application is the module able to transform the events obtained through the eEPC-MW into EPC-related business events, by being aware of the business process in which the *eEPC* has to work. These business events are stored by the *EPCIS Repository* into a database for a long-term storage. Moreover *EPCIS Repository* receives query coming from *EPCIS Accessing Application* and returns data request. For example, data available through the *EPCIS Repository* may include event data (e.g., RFID tag read data collected) and master data (such as product information, date of manufacture, etc).

The eEPC-IS retrieves XML data from remote EPC-enabled systems (located by the eONS) through a dedicated channel named *Remote-EPC Channel*; the eEPC-IS exchanges data with remote EPC-enabled systems using SOAP protocol. The XML code representing the information object associated to the EPC-code, is transferred to the Enterprise-Application, that it manages this information. The communication between eEPC-IS and Enterprise-Application is established by an Socket channel, called *Application Channel*.

The eEPC-IS module implements the following functions to get XML data from remote EPC-enabled systems:

- **`getXMLDataFromEPC`**: connects to remote EPC-enabled systems, via *Remote-EPC Channel* , to get XML code.
- **`sendDataToApplication`**: sends the XML code associated to the EPC code to the *Enterprise Application* module, through *Application Channel*.

B. eEPC: modularity

Modularity is an important requirement in embedded

systems design where the resources are limited.

eEPC is a modular architecture depending on the platform used to run it. In fact, *eEPC* can be used on different embedded platforms with strict constraints in terms of memory, processing power, communication interfaces (serial ports, ethernet communication), etc.

In order to satisfy this modularity, *eEPC* can be configured in two different mode, as shown in Figure 4:

1. *eEPC-Client* mode: it represents a requestor in a *eEPC* Network. It includes the whole set of *eEPC* components previously described (eEPC-MW, eEPC-IS and eONS).
2. *eEPC-Server* mode: in this case *eEPC* represents the last node of the *eEPC* network, providing EPC-code related information through SQL query to retrieve data stored in the enterprise database. Typically, in this case is not required to have the eEPC-IS Capturing Application component and eEPC-MW module to interact with the RFID Reader, because the *eEPC* architecture can be only used to maintain the EPC-code related information.

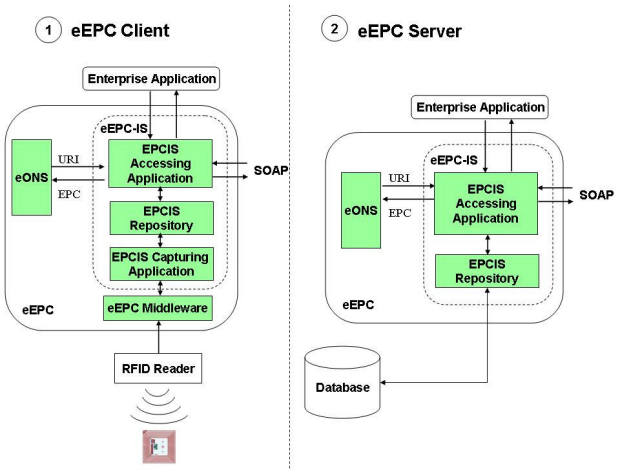

Figure 4: *eEPC* modularity.

C. *eEPC: Embedded implementation*

eEPC networked embedded architecture has been implemented in C++ language for three different operating systems: WinCE embedded OS, eCos RTOS and Linux OS. The C++ implementation of the embedded *EPCglobal Architecture* enabled system involves four design decisions, that concerns eONS module, *remote-EPC Channel* and the *Application Channel* of the eEPC-IS component.

1. eONS has been realized using an embedded Database named SQLite [19]. SQLite is a small C library that implements a self-contained, embeddable, zero configuration SQL database engine. Its main features include:
    - Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.
    - Implements most of SQL92.
    - A complete database is stored in a single disk file.
    - Small code footprint: less than 250KB fully configured or less than 150KB with optional features omitted.
    - Simple, easy to use API.
    - Self-contained: no external dependencies.

    SQLite has been chosen to implement the eONS, because it is a light-weight database engine and because it allows to implement SQL databases for some operating systems, like Windows, Windows CE embedded OS, Windows Mobile OS and Linux.
2. The eEPC-IS exchanges data with remote EPC-enabled systems using the SOAP protocol. The SOAP communication has been implemented by using gSOAP Toolkit [20]. gSOAP enables the integration of (legacy) C/C++ codes, embedded systems, and real-time software in Web Services, clients, and peers that share computational resources and information with other SOAP-enabled applications, possibly across different platforms and language environments. Therefore, the main advantages to use gSOAP are interoperability, legacy code integration, scalability, and performance. Moreover, most toolkits for C++ Web services adopt a SOAP-centric view and offer APIs that require the use of class libraries for SOAP-specific data structures. This often forces a user to adapt the application logic to these libraries. In contrast, gSOAP provides a transparent SOAP API.
3. The communication between eEPC-IS and Enterprise-Application is established by an Socket channel. The Socket library used depends on the operating system running on the embedded platform. For Windows CE OS it is possible to use WinSocket library; on the other hand, using a platform running eCos OS or Linux OS it is possible to use the standard Socket library.
4. Data manipulation is implemented by eEPC by using XML format; therefore an embedded XML parser [21] (called here XML-light) has been integrated into the architecture. XML-light has been chosen because it's suitable for an embedding solution: the parser is small, simple, cross-platform and fast. The main characteristics of the XMLparser:
    - Non-validating XML parser written in standard C++
    - The parser builds a tree structure that can be "explored" easily (DOM-type parser)
    - The parser is able to read a string and build a XML-tree, which can be easily

modified. On the other hand the XML-parser can store the XML-tree into a string.

Table 1 shows the different designing choices to realize the embedded *EPCglobal Architecture* enabled system for WinCE, eCos RTOS and Linux OS. It points out the minimal differences between the three embedded

TABLE I
EMBEDDED SYSTEM PORTABILITY

	WinCE	**eCos**	**Linux**
Remote-EPC Channel	gSOAP	gSOAP	gSOAP
eONS	SQLite	SQLite	SQLite
Application Channel	WinSocket	Socket	Socket
XML parser	XML-light	XML-light	XML-light

implementations completely focused on Socket library used to implement the *Application Channel*.

IV. ARCHITECTURAL EXPLORATION

The *EPCglobal Architecture* constraints have been analyzed to evaluate the proposed *eEPC*. In particular, let us examine the multiple tag reading and the processing functionality implemented by the architecture.

Performance of the *eEPC* has been evaluated by measuring the response time necessary to manage multiple tag reading and processing with respect to two *eEPC* implementations (*Single-Thread* and *Multi-Thread*) and different HW/SW partitioning.

This section describes at first the HW/SW embedded platforms used for this performance evaluation. Then, the *Single-Thread* and *Multi-Thread* implementations are presented and finally the experimental analysis is shown that allows to discover which advantages can be obtained by partially implementing *eEPC* in HW.

A. HW/SW platforms

The *eEPC* networked embedded architecture has been evaluated using three embedded platforms; these platforms include different processors (MIPS-NEC VR4131 or Intel Xscale PXA255), with different embedded operating systems. The three platforms are described in the following:

1. The eTOP platform [22] is a state-of-the-art HMI device designed to meet the most rugged embedded application needs. Diskless and fanless operation makes the product ideal for harsh environment. The eTOP workstations are available with a touchscreen interface and support a variety of display technologies and sizes, from 5.6" monochrome to 15" color TFT. eTOP provides a powerful and convenient platform to advantage from the capabilities of the Windows CE .NET 4.2 operating system. This multithreaded, multitasking, fully preemptive OS environment is designed specifically for hardware with limited resources. The main hardware characteristics of this embedded platform are:
   - CPU: MIPS (NEC VR4131) 200 MHz
   - RAM Memory: 64 MB
   - Communication port: Ethernet, RS-232,USB
2. The other embedded system used to evaluate the *eEPC*, is the Ultimodule SCM240 board [23]; it contains a MIPS - 200 MHz (NEC VR4131) 64-bit processor, a 200-600K Xilinx Spartan IIE FPGA containing system control logic and peripherals, 8KB serial EEPROM, and CAN 2.0B, Ethernet, and USB controllers, and serial interfaces. The MEB-01 contains 16MB Flash Disk, 16MB DDR SDRAM, 2 MB bootstrap Flash, and an additional 256KB EEPROM. The module is ideal for industrial applications requiring any combination of advanced operator interface, display or camera control, or video input; protocol bridging; web services and connectivity; process control with softPLC.
3. The GX080-TFT [24] is a compact-size touch computer. It equips with 8.0" TFT LCD display, Windows CE.Nnet (Version 4.2) and low power consumption CPU, Intel XSCALE PXA255. The main hardware specifications are the following:
   - CPU: ARM (Xscale PXA255) 400 MHz.
   - DRAM: 128 MB on board.
   - Storage: 32 MB flash memory on board (Compact flash slot).
   - Communication interfaces: RS-232/422/485, Ethernet, USB, PS/2.

B. Single vs. Multi thread implementation

Figure 5 shows the pseudo-code of the *Single-Thread eEPC* implementation.

After the `initONS` operation (line 2) executed by the eONS module to setup the Database, the *eEPC* captures the streams of RFID code coming from one or more reader devices, by the `getEPCcode` function (line 3) of the eEPC-MW module. After receiving the EPC codes vector, it analyses any EPC code to satisfy the request (line 5). The procedure `satisfyRequest` (line 7) manages the EPC code and finally send XML code to the *Enterprise Application*.

```
1   eEPC_SingleThread() {
2      initONS();
3      vectorEPC = getEPCcode();
4      for (any EPC code in vectorEPC) {
5         satisfyRequest(EPC);
6      }

7   satisfyRequest(EPC) {
8      EPC-URI = translateEPCcode(EPC);
9      URL = getURL(EPC-URI);
10     XML = getXMLDataFromEPC(URL, EPC-URI);
11     sendDataToApplication(XML);
12  }
```

Figure 5: *eEPC Single-Thread* implementation.

Figure 6 shows the pseudo-code of the *Multi-Thread eEPC* implementation. Like the *eEPC Single-Thread* implementation, after the `initONS` operation (line 2), the *eEPC Multi-Thread* implementation captures the streams of RFID codes coming from one or more reader devices, by the `getEPCcode` function (line 3) of the eEPC-MW module. However, in this case, any `satisfyRequest` function is invocated creating and starting an ad-hoc thread (line 5-6), thus paralleling the satisfy request computation.

```
1   eEPC_MultiThread() {
2      initONS();
3      vectorEPC = getEPCcode();
4      for (any EPC code in vectorEPC) {
5        createThread(satisfyRequest(EPC));
6        startThread();
7      }
8   }

9   satisfyRequest(EPC) {
10     EPC-URI = translateEPCcode(EPC);
11     URL = getURL(EPC-URI);
12     XML = getXMLDataFromEPC(URL, EPC-URI);
13     sendDataToApplication(XML);
14  }
```

Figure 6: *eEPC Multi-Thread* implementation.

C. HW/SW partitioning

HW/SW partitioning is a key problem in the effective design of embedded systems. To increase the *eEPC* performance, let us evaluate which parts of proposed *eEPC* architecture can be mapped on a specific hardware.

The embedded system usually includes a processor, memory, and various hardware devices, as shown in Figure 7; some devices are conventionally implemented in hardware (ASICs), or reconfigurable hardware (FPGA). The processor runs application software that accesses hardware functionalities by using device drivers provided by a RTOS.

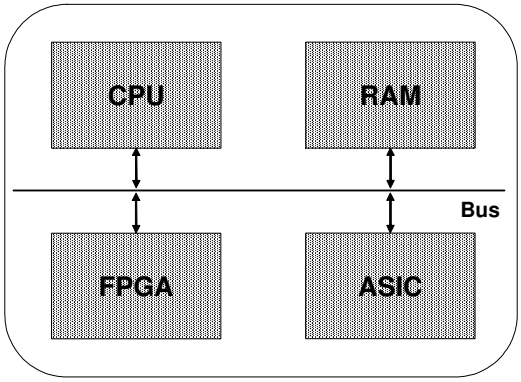

Figure 7: Embedded board architecture.

We used the *gprof* [26] utility to identify the most computational expensive routines of the *eEPC* application software .This software performance measurement has emphasized the following results:

- 50% of the total execution time is spent by the eONS module to query the database.
- 40% of the total execution time is spent by the eEPC-IS routine to establish the SOAP communication with the remote EPC-enabled systems and to retrieval the XML data.
- 10% of the total execution time is spent by the eEPC-MW module to process the streams of RFID code coming from one or more reader devices.

The only software routine synthesizable in hardware (into AISC components or FPGA), to increase the *eEPC* performance, is the eEPC-MW component, by using the Amdahl's Law, as in (1). We can calculate the maximum speedup of the overall *eEPC* architecture when the only eEPC-MW module is mapped on a specific hardware.

This law is used to find the maximum expected improvement of an overall system when only part of the system is improved, and it can be written ad follows:

$$S_{TOT} = \frac{1}{1 - P + \frac{P}{S}} \qquad (1)$$

That is, S_{TOT} is the speedup achievable from the improvement S of a computation that affects a portion P of the overall system.

Based on the results of the plot in Figure 8, potentially we could have a speedup equal to 1.11, if the implementation of eEPC-MW in HW produces a speedup of 100x. It is clear that the huge effort to implement in hardware an *eEPC* module does not produce a significantly gain in terms of overall speedup.

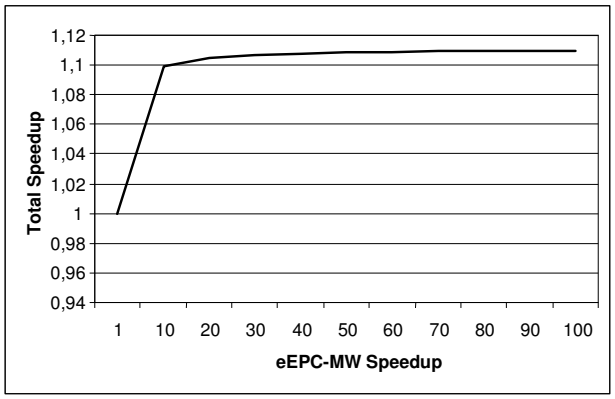

Figure 8: *eEPC* potential speedup with HS/SW partitioning.

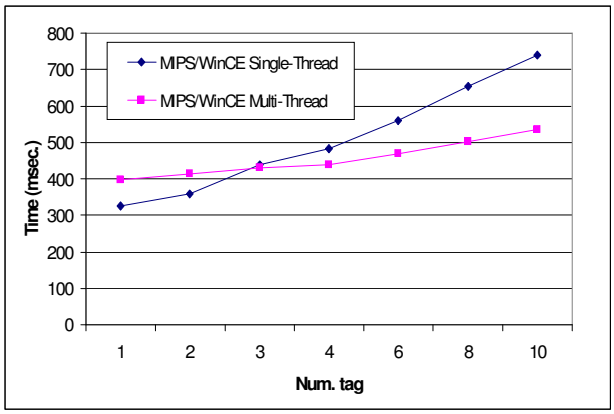

Figure 9: *eEPC* execution time for MIPS/WinCE configuration.

V. EXPERIMENTAL ANALYSIS

A temporal analysis on the following different HW/SW configurations has been performed to select which *eEPC* implementation (*Single-Thread* or *Multi-Thread*) better satisfies the multiple tag readings and processing architectural constraints:

- MIPS/WinCE (eTOP embedded platform).
- MIPS/eCos (SCM240 embedded platform).
- x86/WinCE (x86 emulator simulating a Windows-CE based platform provides by the Microsoft eMbedded Visual C++).
- x86/Linux (x86 architecture running Linux OS).

As a referring application, we use a real scenario where products information is automatically put in relation to user's medical data (e.g., hypertension, diabetes, allergies, drug therapy, current diet, etc.) to retrieve information able to improve the quality of life. The user extracts the product information of a food labeled with a RFID tag, using *eEPC*; the product information is transmitted to the *Enterprise Application*; this module also retrieves the user medical profile from the PIPS[1] platform (*Personalized Information Platform for Health and Life Services*) [25]. The *Enterprise Application* merges data and show information to the user.

Figure 9 shows the results obtained by executing the *eEPC* networked embedded architecture (*Single-Thread* and *Multi-Thread*) on the eTOP embedded platform (MIPS/WinCE). We can observe that for a small number of multiple tag readings (less than 3 tags) the *eEPC Single-Thread* implementation is faster than the *Multi-Thread* version, due to the overhead to initialize and run the threads.

[1] The PIPS project is a international collaborative and interdisciplinary eHealth research & development project funded under the EU FP6 IST programme. PIPS is a dynamic knowledge environment concerning the public's welfare.

The second experiment concerns the comparison of the *eEPC Single-Thread* versus *Multi-Thread* implementation on the SCM240 embedded platform (MIPS processor running eCos RTOS). The results are summarized in Figure 10.

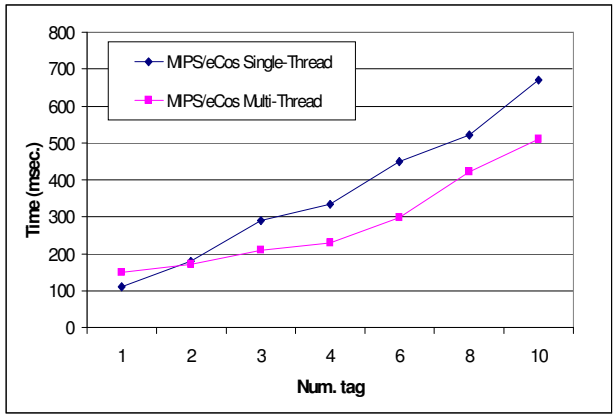

Figure 10: *eEPC* execution time for MIPS/eCos configuration.

The graph shows that eCos RTOS executes the *eEPC* more efficiently with respect to WinCE OS, on the same MIPS processor. In particular the *eEPC Multi-Thread* eCos implementation is faster than the same implementation for WinCE OS, due to the lower overhead to manage threads.

The third experiment consists of the execution of the *eEPC Single-Thread* versus *Multi-Thread* implementations on the x86 emulator simulating a Windows-CE based platform provides by the Microsoft eMbedded Visual C++ (x86/WinCE). Results are showed in Figure 11. We can observe that the simulation times obtained to execute *eEPC Single/Multi-Thread* follow the plots of the previous experiments; therefore, the emulator does not impact in a wrong way on the experiments analysis. Moreover, the results show that the simulation speed even decreases because of the execution speed is bounded by the heavy emulator computation.

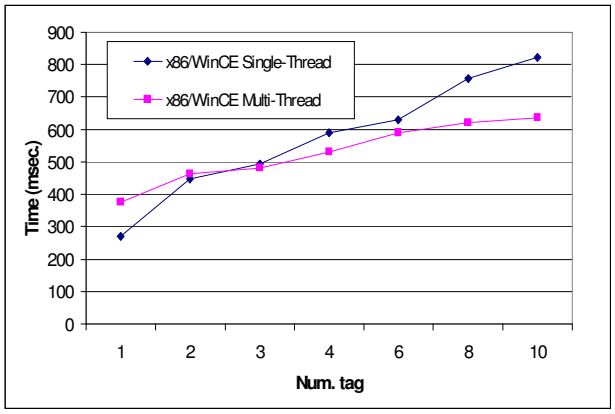

Figure 11: *eEPC* execution time for x86/WinCE configuration.

Finally, the last experiment concerns the *eEPC* execution on the x86/Linux platform. Figure 12 depicts that executing the *eEPC* architecture on a high-performance HW/SW configuration, like x86 processor running Linux OS, gets a significantly improvement of the performance. Note that, in this case there is a small time difference between the *Single-Thread* and *Multi-Thread* *eEPC* implementations due to the power computation of this HW/SW configuration.

Finally, Table 2 compares the *eEPC* networked embedded architecture implementations proposed in this paper, for different operating systems (WinCE, eCos RTOS and Linux), in terms of execution time, lines of code and memory occupation. Column WinCE and eCos represent the *eEPC* architecture run on the same MIPS processor, while the last column shows the *eEPC* architectural characteristics for the Linux implementation executed on the x86 hardware. The table shows that the three *eEPC* implementations are similar (underlined by the lines code parameter), as already shown also in Table 1.

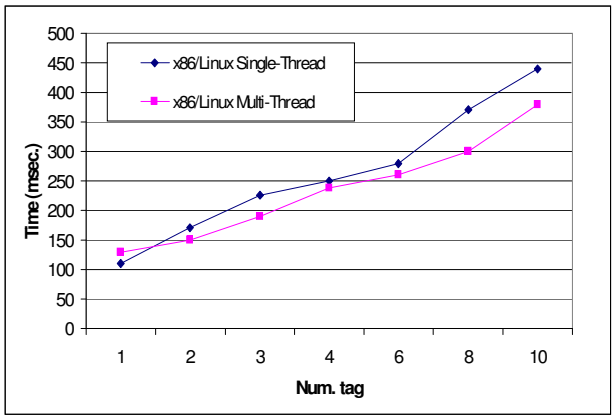

Figure 12: *eEPC* execution time for x86/Linux configuration.

Note that, there is significantly memory occupation saving using the *eEPC* implementation for eCos or Linux OS with respect to WinCE, due to the libraries implementation, like Socket, on the different operating systems.

By comparing all results, we can conclude that the *Multi-Thread* implementation of *eEPC* is necessary only in case of limited computational resources and/or embedded OS with high overhead to manage threads. In this other cases, the *Single-Thread* implementation can satisfy the *EPCglobal Architecture* constraints.

TABLE II
ARCHITECTURAL COMPARISON

	WinCE	**eCos**	**Linux**
Execution time	326 ms	207 ms	154 ms
Lines code [2]	1682	1869	1802
Memory occupation	1348 KB	860 KB	880 KB

[2] This parameter doesn't include the lines code used by SQLite, gSOAP and XML-light

VI. CONCLUSIONS

A networked embedded architecture (called *eEPC*) has been presented in this paper. It analyzes some different design alternatives for the embedded implementation of *EPCglobal Network*, a standard in the RFID tags manipulation.

The main applications that can benefit from *eEPC* embedded architecture can be summarized as follows:

- Tracking goods through the supply chain.
- Enabling consumers to get more information about the products they want to purchase, such as when the items were made, where, whether they are under warrantee and so on.
- Payment Systems (e.g., to pay for bus, subway and train rides).
- Reducing counterfeiting of products. People purchasing expensive goods might have low quality products or, worse, unhealthy products.
- Security and Access Control (e.g., to control who has access to office buildings or areas within office buildings).

We proposed two embedded implementations (*Single-Thread* and *Multi-Thread*) of *eEPC*. The experimental analysis shows that the *Multi-Thread* implementation of *eEPC* is necessary only in case of limited computational resources and/or embedded OS with high overhead to manage threads. In the other cases, the *Single-Thread* implementation can satisfy the *EPCglobal Architecture* constraints. Moreover, the unique possible HW/SW partitioning alternative has been analyzed, thus concluding that the huge effort necessary to implement in hardware an *eEPC* module does not produce a

significantly gain in terms of overall speedup.

With the proposed exploration of the design space, *eEPC* can be used as a reference for any embedded implementation of the *EPCglobal Architecture*.

ACKNOWLEDGMENT

We wish to thank the Scientific Institute H. San Raffaele for providing the application scenario to evaluate the *eEPC* networked embedded architecture.

REFERENCES

[1] RFID Journal, http://www.rfidjournal.com/.
[2] AIM Inc., "Radio Frequency Identification RFID - A basic primer", http://www.aimglobal.org.
[3] R. Weinstein, *RFID: A Technical Overview and Its Application to the Enterprise*, IT Professional, Vol 7, Num. 3, pagg. 27-33, 2005.
[4] D. McFarlane, *Networked RFID in Industrial Control: Current and the Future*, Emerging Solutions For Future Manufacturing Systems. Springer, 2005.
[5] EPCglobal Inc., *The EPCglobal Architecture Framework*, http://www.epcglobalinc.org, July 1, 2005.
[6] EPCglobal Inc., *The EPCglobal Tag Standards Version 1.3*, http://www.epcglobalinc.org, March 8, 2006.
[7] EPCglobal Inc., *The EPCglobal Network: Overview of Design, Benefits, & Security*, http://www.epcglobalinc.org, September 24, 2004.
[8] Mark Harrison, *The EPC Network*, Auto-ID Labs, 2004.
[9] EPCglobal Inc., *Object Name service (ONS) Version 1.0*, http://www.epcglobalinc.org, July 4, 2005.
[10] C. Bornhovd, T. Lin, S. Haller and J. Schaper, *Integrating smart items with business processes an experience report*, In proceedings of the 38th IEEE Hawaii International Conference on System Sciences, 2005.
[11] Pradip De, Kalyan Basu and Sajal K. Das, *An Ubiquitous Architectural Framework and Protocol for Object Tracking using RFID Tags*, In proceedings of the IEEE International Conference on Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS'04), pagg. 20-22, Dec. 2004.
[12] Wei Wang, Duncan McFarlane, James Brusey, *Timing Analysis of Real-Time Networked RFID Systems*, White Paper, 2004.
[13] K. Penttila, L. Sydanheimo, and M. Kivikoski, *Performance development of a high-speed automatic object identification using passive RFID technology*, In proceedings of the 2004 IEEE International Conference on Robotics and Automation, pagg. 4864-4868, New Orleans, LA, April 2004.
[14] C. Roduner, C. Floerkemeier, *Towards an enterprise location service*, International Symposium on Applications and the Internet Workshops (SAINT'06), pagg. 23-27, Jan. 2006.
[15] Sun Microsystems Inc., *The Sun Java™ System RFID Software Architecture*, A Technical White Paper, http://www.sun.com, March 2005.
[16] Smart Label Solutions, *smartEPC*, http://www.slsrfid.com.
[17] F. Fummi, G. Perbellini, *Embedded Design Exploration of EPCglobal Architecture*, IEEE International Conference on RFID, Grapevine, Texas, USA, March 26-28, 2007.
[18] Auto-ID Center, Auto-ID Savant Specification 1.0, September 1, 2003.
[19] SQLite, http://www.sqlite.org/.
[20] R.A. van Engelen, K.A. Gallivan, *The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks*, In proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID'02), May 2002.
[21] XML Parser, http://www.applied-mathematics.net/tools/xmlParser.html.
[22] Ultimodule Inc., http://www.ultimodule.com/.
[23] Exor Electronic R&D, http://exor-rd.com.
[24] FA Controls, http://www.fa.com.my.
[25] PIPS - *Personalized Information Platform for Health and Life Services*, http://www.pips.eu.org.
[26] GNU gprof - The GNU profiler, http://www.gnu.org.

Franco Fummi received the Laurea degree in Electronic Engineering at Politecnico di Milano in 1990 and the Ph.D. in Electronic and Communication Engineering in 1994 at Politecnico di Milano. In 1993 he was Research Assistant at the department of Computer Science of the University of Victoria (B.C.). In 1996 he obtained the position of Assistant Professor in Computer Science at the Dipartimento di Elettronica e Informazione of

Politecnico di Milano where he remained until October 1998. In July 1998 he obtained the position of Associate Professor in Computer Architecture at the Computer Science Department of Università di Verona.

Since March 2001 he is Full Professor in Computer Architecture at the Computer Science Department of Università di Verona.

His main research interests concern hardware description languages and electronic design automation methodologies for modeling, verification, testing and optimization of hardware/software systems. He published more than 150 papers in the EDA field; two of them received the "best paper awards" respectively at IEEE EURODAC'96 and IEEE DATE'99.

Giovanni Perbellini received the Laurea degree in Computer Science at University of Verona in April 2001. Since June 2001 he works at the Computer Science Department of University of Verona as Research Assistant and his research activities concern networked embedded systems design.

Since 2004 he is Teaching Assistant for "Information Processing Systems" course.

Since 2006 Giovanni Perbellini is a PhD student at Computer Science Department of University of Verona.

His main activities concern: parametric design methodologies to allow the re-use of hardware components and IP modules (Intellectual Property modules); modeling, simulation and synthesis of Networked Embedded Systems; design of middleware for Networked Embedded Systems.

The last activity has allowed the design of distributed programming environment for ambient intelligences based on RFID (Radio Frequency Identification) and the implementation of middleware for mobile terminal and Wireless Sensors Network.