Wake on LAN over Internet as Web Service System on Chip

Francisco Maciá-Pérez, *Member, IEEE*, Juan A. Gil-Martínez-Abarca, Héctor Ramos-Morillo, Francisco J. Mora-Gimeno, *Member, IEEE*, Diego Marcos-Jorquera, *Member, IEEE* and Virgilio Gilart-Iglesias*, Member, IEEE*

***Abstract*—In this paper we introduce a System on Chip (SoC) designed to run a particular Web Service (WS) in an Application-Specific Integrated Circuit (ASIC). The system has been designed devoid of processor and software and conceived as a hardware pattern for a trouble-free design of network services offered as WS in Service-Oriented Architecture (SOA). Therefore, the chip is not only able to act as SOAP Service Provider but, it is also capable of registering the service on its own in an external Broker Server using the UDDI Standard publication protocol as well. This proposal has been named Web Service on Chip (WSoC) and its main goal of is to implement more cost effective and zero-management SOA network devices. To validate this approach, a prototypical device has been developed using FPGA technology. The particular network service selected has been WoL over Internet thus allowing any WS Client to wake up any network node compatible with WoL technology. A full SOA scenario also has been developed to test the prototype functionalities and show the proposal validity.**

***Index Terms*—System Administration, Web Services, System on Chip.**

I. Introduction

THE importance of Information Technologies (IT) in all areas of human activity in today's world is an indisputable fact. The simpler these technologies are for end users the more intricate become the backend systems which support them. These technologies provide the technological foundation on which most business processes are grounded, so the importance of providing the necessary mechanisms to ensure these infrastructures flawless operation is paramount.

Although IT services may become extremely complex, they tend to be supported by small, more or less standardized services, which carry out repetitive, quite clearly defined tasks, usually playing supporting roles for high level applications like names services, network configuration, time synchronization, activity monitoring, route calculations or discovery services for instance. As a result of that, all these services involve a certain number of simple but recurrent managing tasks (configuration, monitoring, update and the like); and require such a high number of infrastructures for their deployment that, from a practical viewpoint, they end up becoming a real headache for system administrators.

One increasingly popular proposal for the simplification of these infrastructures and IT services management can be summarized as follows: (1) to design services as Web Services (WS), providing a loosely coupled infrastructure, consisting on several standards and protocols, which enables the easy integration of different services from dissimilar providers in spite of technological support disparities; (2) to design services and systems with a zero-maintenance approach and; (3) to incorporate physical devices designed to provide some of the services described in a self-sufficient manner.

Because of its complexity, WS are mainly implemented on PC-oriented architectures. However, during the last decade, huge advances have taken place in the realm of technologies to develop small network devices with a more than acceptable capacity for computation, autonomous operation and able to carry embedded intelligence [1], [2], [3].

Our initial proposal was to provide specific network services with minimal administration and attention requirements, based on self-configuring management models compatible with WS technology, all of which embedded in extremely small and cost-effective network devices [4], [5], [6]. However, although the current trend towards increasingly small devices, with greater computation and communication capabilities and competitive prices, has led to a seemingly ideal scenario, in fact, some significant problems still remain: (1) from an economic point of view, the use of embedded platforms affects the bottom-line of network services; (2) technically speaking, although it is true that with the proposed based on embedded systems and Service-Oriented Architecture (SOA) approach the administration tasks are dramatically reduced, still a hardware and software platform remains to be managed; therefore, there are still several tasks that cannot be completely phased out.

In this paper we aim to present a System on Chip (SoC) architecture designed to run a particular Web Service (WS) in an Application-Specific Integrated Circuit (ASIC). The system

Manuscript received May 29, 2008. Accepted for publication July 2, 2009.

The authors are with the Department of Computer Science and Technology, University of Alicante, P.O. Box 99, 03080 Alicante, Spain (e-mail: pmacia@dtic.ua.es).

This work was funded by the Spanish Ministry of Education and Science as research project TIN2006-04081 and by the Generalitat Valenciana as research project GV/2007175.

has been designed totally devoid of processor and software, but imitating software architecture of an application server instead. The SoC has been devised as a hardware pattern for a trouble-free design of network services that are offered themselves as WS under the Service-Oriented Architecture style. Thus, the chip is not only able to act as SOAP Service Provider but, it is also capable of registering the service on its own in an external Broker Server using the UDDI Standard publication protocol as well.

Of course, not all the network services will be appropriate for this approach, they will be: simple, based on request-response protocols and with low resources requirements, for instance: gateways between Web Services and other existent services, name resolution services, time synchronization services or light network management services.

To be consistent with the line of thought applied for the term SoC, we have named this proposal WSoC (Web Service on Chip). Its main goal is to be able to develop more cost-effective and zero-management network devices.

To validate this approach, a prototypical device was developed using FPGA technology. The particular network service chosen as WSoC Core for the prototype is WoL over Internet. This network service allows the handling of the remote booting of nodes in a LAN from any location by means of a standard WS client. Incorporating a service of this kind into any organization or even at home is reduced to a matter of connecting the appropriate device to the relevant network.

A full SOA scenario has also been developed to show the proposal viability by means of testing the following prototype functionalities: service publication, discovery and requester.

The paper is organized as follows: section 2 consists of an overview of the current status of technologies and latest research; section 3 presents a general vision of the proposal; section 4 describes the WSoC chip architecture; in sections 5 and 6, the selected WoLI service and the test approach are described; section 7 addresses the prototype implementation developed with FPGA technology and a performance scenario is proposed to carry out the test to validate the WSoC capabilities; and, finally, section 8 summarizes the main conclusions.

II. Background

The first open standards which attempted to address problems of IT management in a global manner were SNMP (proposed by the IETF) and CMIP (proposed by ISO and ITU-T). Both protocols were mainly network monitoring and control oriented, and both shared a similar downside: its lack of platform independence.

Based on those pioneers and in the pursuit of integration with heterogeneous systems, two major lines of work were taking shape: On the one hand, the attempt to achieve integration of the systems using the same network management protocol, as is the case of [7] and [8] with the use of CORBA; and, even more ambitiously, on the other, the proposal of a network management protocol infrastructure-dependency free.

The staggering number of tasks associated with network management, as well as their extremely diverse and complex nature, increase maintenance costs of these systems in terms of resources, time and manpower.

The use of multi-agent systems for computer network management provides a series of characteristics which favor automation and self reliance in maintenance processes [9] [10]. The creation of projects such as AgentLink III, the first Coordinated Action on based on Agents financed by the 6th European Commission Framework Program [11] is a clear indicator of the substantial degree of interest gained by the field of software agents' research.

More recently, with the development of Web technologies, further progress towards self-management has been made. As a result, self organization models based on ontologies as information models and on SOA as an operational model has being created, providing administrators with ubiquitous, platform-dependency free interfaces [12].

The diversity of existing network management models leads us to an obvious conclusion: we are in dire need for the creation of mechanisms which interoperate between all the management domains involved [13]. This relation may reach a semantic level using ontologies in such a way that renders possible to work with an abstract, model-independent view of the network management information, enabling administrators (persons or software) to automate management tasks [14].

In [15], a standardization proposal has been made for a group of Web Service operations, all of them essential from a network management point of view. This proposal is made in a similar way like the standardization of the SNMP information model under XML was showed in [16].

There are several different embedded platforms in the market able to support a Web Service, like RabbitCore, SHIP, or Digi ConnectMe to mention a few. These systems, generally based on 32 bits microprocessors, implement a whole Web server with a software-written TCP/IP stack, all of them supported by an embedded operating system. Due to software complexity issues, these low cost solutions present clear drawbacks in terms of performance and liability.

Aiming to adjust production costs even further in order to keep the project on budget without sacrificing performance [17], [18], Systems on Chip have also been included in: basic network services [19], Web Services [20] and general network applications [21].

Web Services technology constitutes the implementation vehicle of choice for service-oriented architectures [22]. As a matter of fact, several authors [23], [24], [25] have developed models based on the Devices Profile for Web Services (DPWS) stack for low resources level devices. The profile arranges several Web Service specifications such as WS-Addressing, WS-Discovery, WS-Metadata Exchange, and WS-Eventing. Due to the FPGA technology inherent resource limitation we opted for Web Services Interoperability (WS-I) basic profile version 1.0 implementation although the

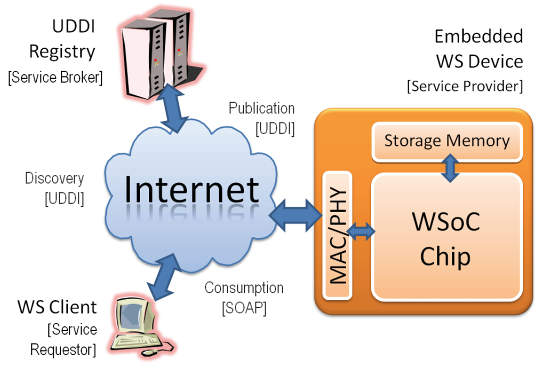

Fig. 1. SOA scenario based on WS Technologies and standards.

inclusion of new WS-* in upcoming works is still open to discussion. DPWS has been ruled out from this work on account of its local scope [23] without including another network components as discovery proxies which is bound to limit some of the offered network functionalities.

In [4], [5] and [6] we develop network administration Web Services as embedded software devices; technology whose usage allowed us to develop cost effective prototypes for the proposed network administration model. However, in order to further on our research and from the mass production process point of view, is important to take into account that: (1) becoming dependant of third-party technologies is a constrainers that must be avoided; (2) although its usage might be cost wise advisable during the prototype development phase, it backfires when it comes to mass production [26]. In a technical report from the ITEA SIRENA Project (within the Eureka framework) a $5 processor on chip can be achieved, which means a huge improvement as compared to the approximately $40 you have to pay for an equivalent embedded solution [26].

It is precisely along these lines that we can frame our proposal: the development of single chips capable to provide more cost effective and self-managed network services as Web Services.

III. Proposal Description

The main purpose of this work is the design of hardware architecture for a chip which, by itself, is able to provide everything required to develop embedded network devices which offer network services in the form of Web Services under SOA model. However, the approach is only suitable for certain type of network services with features like: lightness, based on request-response protocols or with low resources requirements. According to this, the proposal is appropriated to the design of gateways between Web Services and other existent services (NTP, ICMP, SNMP, DHCP, etc.), name resolution services, time synchronization services, light network management services, etc. For this same reason, the proposal would not be suitable to offer complex services like: web servers, file system servers or application servers.

A. SOA approach

SOA implies some additional elements for the device providing the services (see Fig.1). For this reason we begin the overview of our proposal by reviewing the most important components:

1) *Embedded WS Device*. In SOA terminology, this device is known as the *Service Provider* and it has been designed using the WSoC chip which is the subject of this work. The device runs a Web Service and publishes its interface and access information in the service registry. Each provider must decide which services to expose, how to formulate trade-offs between security and easy availability, how to price services, or, when free, how to exploit them for other value. The provider also has to decide what category the service should be listed in for a given broker service and what sort of trading partner agreements are required to use one particular service.
2) *UDDI Registry*. In order to ensure genuine client-server independence, SOA incorporates a *Service Broker* or *Service Registry* which will contain a formal description of the various network services provided. This description is made by the Web Services Description Language (WSDL) and contains information on service addresses, accepted parameters and how to return results. In the case of Web technology, the Universal Description Discovery and Integration (UDDI) specification is in charge of making both the *publication* of the service and its subsequent *inquiry*.
3) *WS Client*. The Web Service client o *service requester* locates entries in the broker registry using various find UDDI operations, getting the WSDL page in which the service is described, and then binds to the Embedded Web Service Device in order to invoke one of its Web Services by means of SOAP protocol.

B. Embedded WS Device

Although all the above mentioned elements are essential for the development of a fully uncoupled Web Service under the SOA model, one of the main objectives of this work is to accomplish the capacity to design embedded Web Service devices using the WSoC chip described in the following section.

An embedded WS Device consists of a Storage Memory, a Network Interface and the WSoC chip (Fig 1). Let's take a more comprehensive look at each of its elements:

1) WSoC Chip: The WSoC (Web Services on Chip) is the device kernel that also provides a hardware framework with all the functional elements needed by a Web Service to register itself in a UDDI registry and to offer their services to a Web Service client. The functionality, in terms of Web Services of each WSoC, as it will be analyzed in the following section, is implemented in its WS Core.
2) Storage Memory: Is an external storage memory that allows configuration and work information to be stored. The memory has been divided into two main blocks: an

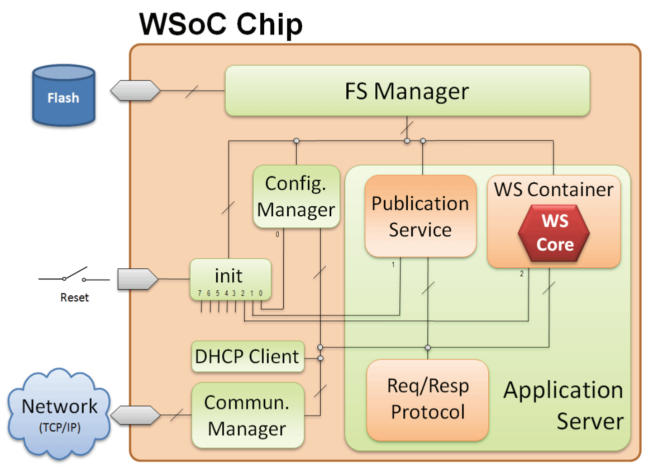

Fig. 2. Web Service on Chip Architecture.

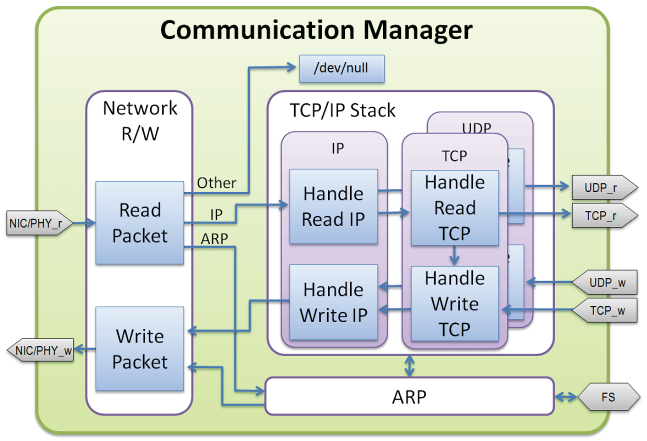

Fig. 3. Communication Manager block flow diagram.

area for the system files and a user area. The system area will harbor system files, WSDL sheets with services descriptions, system registry files and other auxiliary files. The user area can store files defined by the specific service implemented in each WSoC chip.

3) Network Interface (MAC/PHY): Is a communication module with the Network Interface Card (NIC) that provides physical access to the TCP/IP network.

Of course, since this WSoC Chip is the most important component and the essence of this article we will describe it in further detail in the next section.

IV. WSoC Chip Proposal

After this general overview of the system we may now proceed to the fundamental purpose of the work, namely, the WSoC architecture of the chip providing the Web Services.

One of the proposal's main features is that its architecture is not processor or software applications oriented. Due to the power of current design tools it is relatively easy to propose solutions based totally on architectural blocks of hardware.

Anyway, lack of processor or software notwithstanding, the chip's design hardware has been created based on a similar architecture to the typical software architecture for this type of system.

Fig. 2 shows a graph with the main hardware blocks which constitute the chip. Further considerations about the functionality of each of these blocks are coming next.

A. Init, Configuration manager and DHCP client

The *init* module is the first to take control once the chip has begun to act. Its functionality is analogous to the UNIX-like *init* process, since it is the first process which loads and executes the operating system and whose process identifier (PID) is always 1. This module reads from the non-volatile memory of the device through the file system management module, both in the status in which the chip was initiated and the *inittab* file which contains a description of the modules to be activated by the chip based on its status. Currently, three different states are defined (see Table 1).

TABLE I
WSoC Chip possible System States

State	Description	Active hardware block
0	Config state	Configuration Manager
1	Publication state	Service Publication
2	Service state	WS Container

As soon as the chip has been initialized it takes over the *init* module, whose function is to decide which ones among the rest of the modules is going to be active at any given time. At first, it assumes the *configuration status* (*enabling the configuration manager module*) to make readable from the FS configuration parameters such as IP address, mask, default Gateway, UDDI server address, etc. In the event of those parameters non-existence, they will be searched outside by means of a DHCP protocol (managed by the *DHCP client module*). Once the chip has been successfully configured, the *init module* assumes the *publication status* (enabling the *publication service module*) to make possible the Web Service registration in an external UDDI server. Finally, the *init module* assumes the *service status* (enabling the *WS Container module*) giving way to the WS feature contained in the *WS Core*.

In a UNIX operating system, which we have used as a base, the first process (with PID 0) is actually a planner (generally the *sched* process) which enables multitasking. Since the present design does not consider this characteristic, we opted for its elimination from our proposal.

B. FS Manager

The File System Manager administrates the File System I/O located at the storage memory. It allows read/write file operations on the storage memory. The files type may vary (HTML, WSDL, logs, user parameters, and system status or configuration files) depending on the nature of the service. In order to obtain the best out of the reduced storage space available, optimized versions of each type of file are proposed. The FS Manager is in charge to conceal these details from the rest of the modules.

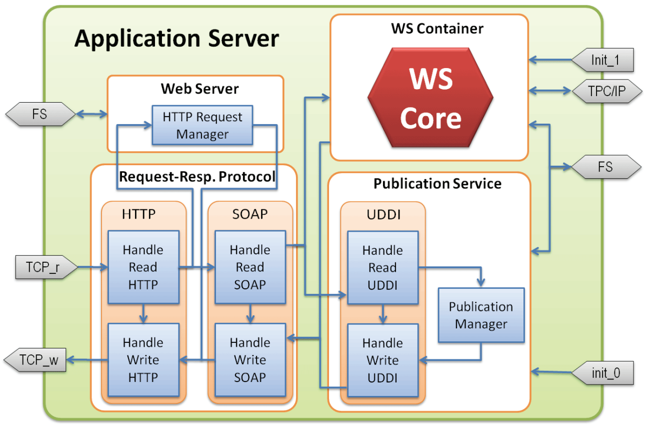

Fig. 4. Application Server flow diagram.

C. *Communication Manager*

The Communication Manager Module provides communication standards like TCP/IP stack and ARP protocol through the MAC/PHY to communicate the platform with other nodes over the data network.

Fig. 3 shows a graph with the main hardware blocks in the module, together with the main data flows between the blocks. Since there is no explicit processor or software in our proposal, the physical connection between the blocks is very similar to the data flow. As may be seen, one of the points of entry to this block is in the *Read Packet* which is part of the *Network R/W* module. Based on type, each frame is transferred to the block required to handle it. If the type of frame cannot be processed by the chip, the chip eliminates the frame. The ARP block implements the ARP protocol in response to the WSoC Chip IP address requests as well as giving support to the TCP/IP stack.

The *IP* module manages the IP traffic and if necessary, it will pass the packet to the *Handle Read TCP* block in order to continue its processing.

In addition, the communications module may receive requests through the *Handle Write TCP* block included in the TCP module in order to send information through the communications network. This block builds a TCP packet and transfers it to the IP module (*Handle Write IP* block) which adds the corresponding information to its network protocol level, subsequently sending it to the *Write Packet* block which will convert it into an appropriate network frame for the type of network interface implemented by the embedded device. Due to current FPGA constrainers, we have not included in this version the implementation of a Transport Security Layer (TSL) between the TCP transportation protocol and the application layer in order to pass on the information with the adequate levels of privacy and authenticity, but that's a possibility that remains open to future discussions.

D. *Application server*

This module imitates the behavior of a conventional application server based on hardware and software (processor, operating system, libraries, software applications, etc.). For this purpose, taking this model as reference, a block was designed for managing the request-response protocols (basically HTTP and SOAP); a block which acts as a Web Server for managing conventional HTML requests; a block for self publication management provided through the UDDI protocol; and last but not least, the most important block for the purpose of this work, namely a WS container known as *WS Core* (Fig. 4), where services to be provided by the chip are located.

As mentioned at the beginning of this article, the design principle behind the WSoC chip was to serve as hardware pattern for the creation of different Web Services embedded in a chip, so that the designer may focus solely on the service details to be implemented. Furthermore, all the architecture was conceived mimicking conventional software architecture. This way, developers of WS Cores may make use of an interface with specified resources. This interface is very similar to the one usually employed by conventional applications server.

1) *Request/Response Protocol*

This block contains the functions that implement the request-response application protocol on which services communication (executed within the applications server) will be based. In our case the protocols are: HTTP and SOAP, which will act as transport and application protocols respectively. We do not rule out for the future communications support via based-on TLS HTTPS, currently discarded due to FPGA limitations.

2) *Web Server*

The web server in our design is significant solely in terms of design compatibility. Since the aim is a hardware design which imitates software architecture of an application server, the task of the web server is to act as the chip's *front-end*. It will receive the HTTP requests, and will evaluate and directly resolve or redirect them to the applications server in case of invocations on static or dynamic objects respectively.

Another purpose of the web server is to enable the provision of an agreeable, easily-configurable, standards-based administration interface for the device and the service.

3) *Publication Service*

When a device is connected for the first time, the *init* block detects this situation reading its status from the initialization file and activating this block. The service's task is to automatically register the services to be provided by the chip in a UDDI Registry. One of the first aspects which we need to address is how to represent the service offered by each machine and how to describe each of the activities and processes which are part of this service. To do so, we shall use a WSDL sheet which describes each of the processes that the device can do.

The implementation of the protocol is based on a series of WSDL templates stored in the non-volatile memory. The *UDDI Manager* module is responsible for recovering these sheets and using them to compose SOAP messages for the communication with the UDDI server. Our proposal is based

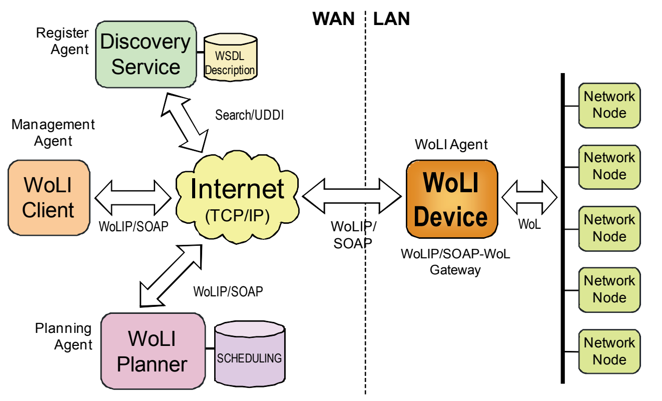

Fig. 5. Scenario for WoLI device performance.

on WS-I basic profile 1.0 with synchronous request-response interactions over HTTP transport protocol.

Whenever we want to register or update services in a UDDI server it's sufficient to configure the chip in *publication state* (see Table I) and restart it. After a satisfactory registration, this module situates the device status in *service state* and restarts it.

4) WS Container

The WS container's main function is to isolate the *WS Core* of the hardware implementation details, providing a familiar interface for the Web Services designer, so that with the present hardware design and development tools, its implementation will be similar to standard WS programming.

Since self-sufficiency is a device prerequisite in order to make clients able to discover and operate it, it is necessary to register its service before activating the WS container. This block is activated when the *init* block detects that the device status is 2 (see Table I). At this point, it will take over the chip and provide the WS interface through which the WS clients will be able to contact the device, not only to administrate it, but also to access the services it provides.

V. WoLI Web Service

In order to provide the proposal with enough functionality to facilitate comprehension of its functioning, and aiming to develop a functional prototype to test it, we will propose a specific network service offered as WS. This service is known as WoLI Service and its thorough description will be the goal of this section.

The WoLI service is a Web Service which enables boot control of the network nodes with WoL support via Internet standard protocols and Service-Oriented Architectures, which renders the service regardless the administrator's location or used platform.

The main convenience of the WoLI service is to facilitate system nodes remote management throughout a wide area network in general, and the Internet in particular, where the simple impracticality of start up, disconnect or restart distant nodes notably restricts the systems administrators' ability to act remotely.

Fig. 5 shows a diagram of the main elements and agents involved in the service, as well as the existing relationships between them. They will be, in a nutshell: WoLI clients, WoLI schedulers, WoLI device and the network nodes to which are the service destined. A more detailed explanation of each one of them will ensue on the next paragraphs.

The **WoLI client** provides the user, via service management agent, access to the service, both for schedule, and in order to generate WoL instructions via Internet. Orders are transmitted to the WoLI scheduler or to the WoLI device embedded in SOAP messages by means of the WoLIP application protocol (Wake on LAN over Internet Protocol) defined for this purpose. This agent will generally be external to the device.

The **WoLI Scheduler** usually acts as a control panel for all the possible WoLI devices distributed throughout the Internet. This control is implemented via the scheduler agent which carries out, executes and verifies all the previously established tasks on the WoLI devices (switches on an individual or a group of nodes, verifies its status, updates the firmware and even plans the work of the WoLI devices). This agent may either reside in an external node outside the LAN, generally at the location of the service or communications provider, or be integrated in a WoLI device. From a WoLI client point of view, the scheduler agent behaves like a Web Service which can be located thanks to the UDDI register.

The **WoLI Device** is the cornerstone of the service. This is an embedded network device designed with a miniaturized WSoC chip conceived to act as a WoLIP-WoL pathway between the broad area network and the local area network in which it is located.

In the specific case of an instruction to remotely boot a network node, the agent undertakes the task to translate the request to a WoL datagram.

The **WoLI agent** always acts from within the LAN and behaves differently, depending on whether it acts in active or passive mode. In passive mode, the agent awaits reception of WoLIP requests from a management agent, generally external, in order to execute them on demand (Fig. 6). However, in

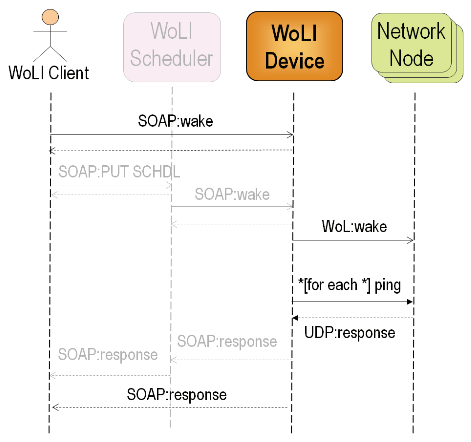

Fig. 6. WoLI passive mode service sample sequence diagram.

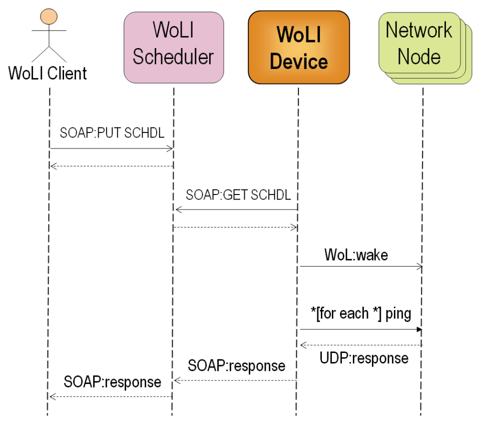

Fig. 7. WoLI active mode service sample sequence diagram.

active mode, it is up to the agent to make the first move and request a work plan from a scheduler agent (Fig. 7). Its operation in active mode is fundamental, given that it enables the independent work of all the possible security and protection intranet policies implemented, since, for this purpose, it would use standard HTTP requests as a base for the WoLIP protocol.

This agent acts as a Web Service, receiving instructions defined in the WoLIP protocol, encapsulated in SOAP messages and defined by means of WSDL pages. In our proposal, this agent's functionality is, in fact, the same that the one we will implement as WS Core of the WSoC Chip.

Network nodes are the object of the administration and this term comprises all those devices connected to the system with WoL support in their adapting cards. We are talking about PCs, network servers, networking devices or any other piece of equipment which fulfils the conventional requirements.

The **Discovery Service** comprises a standard UDDI registry service. It is responsible for maintaining the pages that describe the WoLI services in WSDL format, as well as make available that information to service access-seeking clients.

VI. WoLIP Service Protocol

The WoLIP service protocol defines a series of instructions used by users and by the system's manifold components in order to communicate with each other. This protocol is based on messages and is supported by SOAP, which acts as an information exchange mechanism allowing remote procedure calls.

Each WoLIP protocol command is embedded in the body of a SOAP message which contains the name of a remote procedure, which, in turn, implements the functionality of the command and the arguments required for its execution.

Table II.
Main commands supported by WoLIP Protocol.

CMD	ARG	FUNCTION
SET	MODE	Reports the current operation mode.
	MODE PASSIVE [puerto]	Sets the passive mode and, optionally, the listening port number.
	MODE ACTIVE <ip> [:Puerto]	Sets the active mode, specifying the server's IP address and port number.
	RUN	Reports the current WoL service state.
	RUN <STARTS | STOP>	Starts or stops the WoL service.
UPDT	FIRM <file>	Updates the device's firmware, from the specified file.
GET	SCHDL	Returns the list of scheduled tasks in the device.
PUT	SCHDL	Adds a task or a set of tasks to the scheduling.
VALIDATE	<user> <pass>	User identification and authentication.
WAKE	<host>	Boots a network node through WoL.
PING	<host>	Checks if the specified network node is running.

```
SOAP Envelope
    SOAP Msg Header
    SOAPMsg Body
        WoLIP Command
            Action level 1
            Action level 2
            Arguments
```

It is possible to distinguish the following elements:

- *Command* defines the service actions in terms of request. It corresponds to the name of the remote procedure which implements the functionality of the WoLIP command.
- *Action level 1* and *Action level 2* are special parameters which profile the functionality of the request.
- *Argument* represents the necessary information for executing the application.

For each WoLIP request message there will be a corresponding SOAP response, the body of which will be formed by the request made or by an error message should any execution problem arises.

The requests defined in the protocol (see Table II) may be grouped into three major types: configuration orders, basic orders and control orders.

The configuration of the internal variables of the device determines their function mode. These variables are managed using the SET command. The basic service provided by the WoLI is invoked using the WAKE command. We present an example below of a SOAP request which invokes the WAKE command.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap= \
        "http://schemas.xmlsoap.org/soap/envelope/"
  Soap:encodingStyle=       \
       "http://schemas.xmlsoap.org/soap/">
  <soap:Body>
    <wp:wake xmlns:wp=      \
              "http://www.dtic.ua.es/wolip">
      <host>192.168.1.23</host>
    </wp:wake></soap:Body>
</soap:Envelope>
```

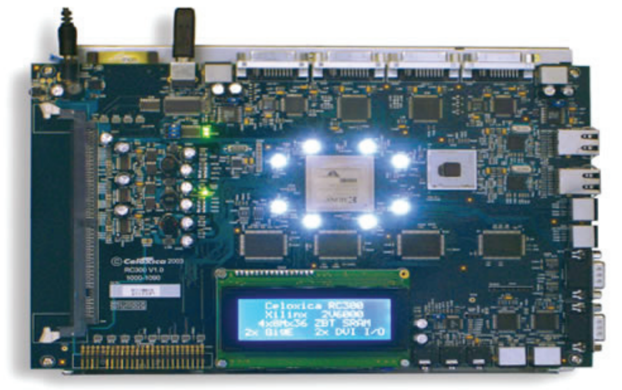
Fig. 8. Celoxica RC203E board.

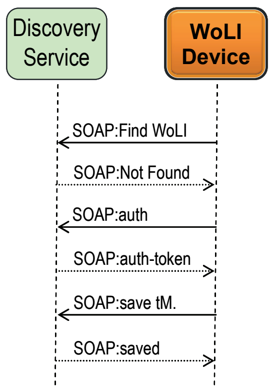

Fig. 9. Publication service test sequence diagram.

The PUT and GET commands, combined with the SCHDL argument for programming and obtaining the programming of a device, enable the connection process of one or several sets of equipment to be planned in an autonomous manner.

The service permits access control by means of a user-password type list. These variables are managed via the VALIDATE command.

All the WoLIP protocol orders provide a response: OK, if the order is properly executed or, conversely, ERROR if it is not, except GET SCHDL which returns the list of tasks programmed for the device and the PING command which will asynchronously return the status of the network node it has attempted to switch on.

Fig. 7 shows a simple sequence diagram in which a WoLI client or a previously programmed Scheduler requests remote boot of network device through a WoLI device, which translates the request so that its LAN nodes understand it. In addition the WoLI device can verify that the node did, in fact, start up, communicating this to the requester.

VII. Implementation and Evaluation

A. *Prototype implementation*

FPGA technology has been chosen as a rapid and appropriate design tool for the implementation of the prototype. The Celoxica RC203E board was used for the implementation of WoLI SoC. This board has one Xilinx XC2V3000 FPGA, two ZBT 4MB memory banks and a SmartMedia Flash memory for permanent storage. The network interface included in this board is the MicroSystems LAN91C111 which includes a MAC+PHY full-duplex controller with a 16KB buffer.

In order to simplify the process we declined to implement the DHCP client given that, for all validation tests effects, the chip is able to operate using the FS stored configuration. The WoLI planner has not been implemented because the active mode of this service has not been implemented in the prototype.

The whole WSoC design uses 8,936 slices of the FPGA (63%), one block RAM (1%) and works at 40 MHz. Only 10KB of external SRAM are used for temporal storage of auxiliary data. Usual estimates agree that, once the DHCP client is added, only about 70% of FPGA slices remain occupied, leaving the remaining 30% free for WS-Core implementation.

Although this margin may sound a little bit narrow to specialists, it is important to point out that this platform is oriented to undertake only simple network management applications, meaning that most of the necessary functions will be supplied by the already wired inferior layers. The fact that the WS-Core developed for our tests only made use of a meager 3% of the available FPGA is evidence of that.

Handel-C high level description language was used for the system coding, allowing a fast and incremental development by means of Celoxica DK4 Design Suite development environment. The DK validates, compiles and generates an EDIF file used by the Xilinx ISE 7.1 tool to synthesize, implement the Place & Route and generate the bitstream necessary for FPGA reprogramming. Some libraries included in PDK v4.1 (Platform Development Kit) were used as drivers for the external devices. The SmartMedia is used to store the bitstream of the whole design. The design uses nor microprocessor neither software. It draws on the architecture shown in section 3.

B. *Prototype Evaluation*

In order to evaluate the proposal, a scenario has been recreated such as that indicated in Fig 5. This scenario has incorporated the prototype implemented based on the Celoxica RC2003E plaque, a UDDI registry service, a WS client for accessing the system and a PC network compatible with WoL.

1) *Publication Evaluation*

An Apache jUDDI v0.9rc4 server able to support UDDI v.2.0 has been used to check the auto-registration module. It allows the service registration in a standard way. Besides, we could check the correct communication by connecting the prototype to a network and sniffing the network traffic between the device and the server. The device seeks the service in the jUDDI server as shown in Fig.9. If it doesn't find it, it will publish the service by doing the authentication to deal with jUDDI server private functions.

The TCPDump tool installed in the server has been used to sniff network traffic during the evaluation and the generated information has been analyzed using *The Wireshark Network Protocol Analizer*. With these tools it was possible to verify the correct device-server communication. The tools have also allowed us to capture network traffic similar to the one shown next. In this traffic fragment, the network device sends a SOAP message with the protocol UDDI command *save_tModel*.

```
POST /juddi/publish HTTP/1.1
Content-length:260
Content-Type: text/xml; charset=utf-8
Host: 172.19.32.39

<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas
  .xmlsoap.org/soap/envelope/">
  <soapenv:Body>
   <save_tModel generic="2.0" xmlns="urn:uddi-
  org:api_v2">
   <authInfo>
authToken:E98101F0-4656-11DC-81F0-FA34E9DBA31E
  </authInfo>
  <tModel tModelKey="">
    <name>wolifpgadtic</name>
    <description>WoLI Service</description>
    <overviewDoc>
      <description>
        WoLI service by FPGA
      </description>
      <overviewURL>
        http://172.019.032.151/index.wsdl
      </overviewURL>
     </overviewDoc>
    </tModel>
   </save_tModel>
  </soapenv:Body>
</soapenv:Envelope>
```

2) *Service Evaluation*

The service has been successfully tested using, on the one hand, a PHP client using a NuSOAP library and, on the other, a command line standard client developed by the Apache group known as WSIF. This standard command line client is able to use services through an invoker by writing the function, the parameters and the address where the WSDL sheet is stored; after which it builds a correct call to the prototype service and shows the obtained response.

The procedure carried out by PHP and WSIF clients when the auto-registration process has finished is as follows:

- Firstly, the client seeks the service in the jUDDI server to obtain the address of the WSDL sheet that, in this case, is stored in the FPGA for testing the device web server capabilities.
- Secondly, the client requests the WSDL description sheet through a HTTP GET request. After that the FPGA replies with the WSDL sheet.
- Once the WSDL description is obtained, the client builds valid SOAP requests that will be sent through HTTP POST messages to the FPGA Web Services server. And it will reply with the corresponding SOAP messages, according to the described WoLIP commands in Table II.

In order to validate the proposal, a small selection of commands from the WoLIP protocol was implemented, including the protocol's WAKE command, and a variation of the VALIDATE command which does not require a password and which we have termed LOGIN <user>. A small portion of the source code of the service logic is specified bellow. As you could see, in order to execute the WAKE command correctly it is necessary for the invoking host to have previously executed the LOGIN command. The session will be valid until the host invokes the LOGOUT command.

```
switch(command) {
 case 1: // LOGIN
  if(user==0x64746963 && logedHost==0)
  {
    //Test If Correct user and send Response
    SendTCPData(&IpInfoPtr,&TcpInfoPtr,LOGIDIR,1);
    logedHost=IpInfoPtr.SourceAddress;
  }
  else
    SendTCPData(&IpInfoPtr,&TcpInfoPtr,LERRDIR,1);
 break;
 case 2: // LOGOUT command
  // Test previous login and send WoLIP response
  if(IpInfoPtr.SourceAddress==logedHost)
  {
    SendTCPData(&IpInfoPtr,&TcpInfoPtr,LOGODIR,1);
    logedHost=0;
  }
  else
    SendTCPData(&IpInfoPtr,&TcpInfoPtr,LOERDIR,1);
 break;
 case 3: // WAKE command
  if(IpInfoPtr.SourceAddress==logedHost)
  {//Send Wake Packet if previous LOGIN
   SendUDPWol(&IpInfoPtr,TcpInfoPtr.IpsCheck,mac);
   //Send WoLIP Response
   SendTCPData(&IpInfoPtr,&TcpInfoPtr,WAKEDIR,0);}
  else
   SendTCPData(&IpInfoPtr,&TcpInfoPtr,WAKERDIR,1);
  break;
 }
```

As in the registry process evaluation, a network monitor has been used to capture the network traffic between the client and the FPGA. As a result, we have been able to verify that the communication between both is carried out in a correct way, as can be seen by means of the message sent from the client that is shown next.

```
POST /woli.wsdl HTTP/1.0
Host: 172.19.32.151
User-Agent: NuSOAP/0.7.2 (1.94)
Content-Type: text/xml; charset=ISO-8859-1
SOAPAction: "urn:woliwsdl#wake"
Content-Length: 507
<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle
=”http://schemas.xmlsoap.org/soap/encoding/”
xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"xml
ns:xsd="http://www.w3.org/2001/XMLSchema"xmlns:xsi
="http://www.w3.org/2001/XMLSchema-
instance"xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns="urn:woliwsdl">
<SOAP-ENV:Body>
```

```
    <tns:wake xmlns:tns="urn:woliwsdl">
      <host xsi:type="xsd:string">
        000D88270A36
      </host>
    </tns:wake>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Obviously, having sent the SOAP response to this message, it was observed that the *magic packet WoL frame,* generated by the FPGA and sent across the network, caused the boot up of the specified host.

VIII. Conclusions

In this paper we have presented a SoC to design cost effective and zero-maintenance embedded network devices that run Web Services in a SOA style. Due to the characteristics of the proposal, the offered services should be simple and with minimum resource requirements.

The main advantage of this approach is that it can provide services, which are common in today's network environments without any need for specialized system administrators, neither for the initial deploy and setting up tasks nor for the subsequent maintenance tasks.

A WoL over Internet FPGA-based prototype has been also developed in order to test the functionalities and to assess the proposal. The device is able to auto-register and to work independently or integrated with other services from existing networks, regardless of whether they are provided with other similar devices or not, or as conventional network services.

From a user's perspective, each service represents a small network device which simply requires connection to the network of their organization or home to make the required service available.

We are currently working in the implementation of the TLS/SSL security protocols in order to provide support to HTTPS, as well as developing other WS cores that implement network services and integrate them all into a model based on Semantic Web Services, so that in the future they not only will be compatible with existing services, but also with new services or configurations which were not considered in its initial design as well.

References

[1] J. Roy and A. Ramanujan. "Understanding Web Services." *IEEE Internet Computing*, vol. 3, no. 6, pp. 69–73, Nov.-Dec. 2001.

[2] L. Jóźwiak and S. Ong, "Quality-driven model-based architecture synthesis for real-time embedded SoCs," *Journal of Systems Architecture*, vol. 54, pp.349-368, March-April 2008.

[3] J.J. Rodriguez-Andina, M.J. Moure, and M.D. Valdes, "Features, Design Tools, and Application Domains of FPGAs," *Trans. on Industrial Electronics*, vol. 54, no. 4, pp. 1810-1823, Aug. 2007.

[4] V. Gilart, F. Maciá, J.A. Gil, and D. Marcos, "Services and networks management through embedded devices and SOA," in *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference EDOC 2006*, Hong Kong, Oct. 2006, pp395-398.

[5] J.A. Gil, D. Marcos, F. Maciá, and V. Gilart, "Wake on LAN over Internet as Web Service," in *Proc. of the 11th IEEE International Conference on Emerging Technologies and Factory Automation* ETFA 2006, Prague, September 2006, pp1261-1268.

[6] F. Maciá, D. Marcos, and V. Gilart, "Industrial TCP/IP Services Monitoring through Embedded Web Service**,"** *Journal on Embedded Systems*, Vol. 2008, pp 1-10, 2008.

[7] M.S. Jeong, K.H. Kim, J.H. Kwon, and J.T. Park, "CORBS/CMIP: Gateway Service Scheme for CORBA/TMN Integration." *Knom Review*, Vol.2, No. 1, pp. 55-62, 1999.

[8] G. Aschemann, T. Mohr, and M. Ruppert, "Integration of SNMP into a CORBA- and Web-Based Management Environment," in *Proc. Kommunikation in Verteilten Systemen*, Heidelberg, 1999, pp. 210-221.

[9] T.C. Du, E.Y. Li, and A.P. Chang, "Mobile Agents in Distributed Network Management," *Communications at the ACM*, vol. 46, no. 7, pp. 127-132, 2003.

[10] J. Guo, Y. Liao, and B. Parviz. "An Agent-Based Network management system" in *Proc. of the 2005 Internet and Multimedia Applications*, Honolulu (Hawai), Aug. 2005.

[11] European Co-ordination Action for Agent-based Computing [Online]. Available: http://eprints.agentlink.org/. [Accessed: March 20, 2009].

[12] R. Boutaba and I. Aib, "Policy-based Management: A historical perspective" *Journal of Network and Systems Management*, vol. 15, Issue 4, pp. 447-480, December 2007.

[13] J.E. López. V.A. Villagrá, and J.I. Asensio, "Ontologies: Giving Semantics to Network Management Models," *IEEE Network*, vol. 17, no. 3, pp. 15-21, May-June 2003.

[14] J. Peer, "A POP-Based Replanning Agent for Automatic Web Service Composition," in *proc. of the 2005 Second European Semantic Web Confenrece*, LNCS, pp 47-61, May 2005.

[15] J. Sloten, A. Pras, and M. Van Sinderen, "On the standardisation of Web Service management operations," in *proc. of the 2004 X EUNICE Summer School and IFIP WG 6.3 Workshop, Tampere*, Finland, June 2004, pp. 143-150.

[16] T. Klie, and F. Straub, "Integrating SNMP agents with XML-based management systems," *IEEE Communications Magazine*, vol. 42 Issue 7, pp. 76-83, July 2004.

[17] L.S. Indrusiak, M. Glesner, and R. Reis, "On the Evolution of Remote Laboratories for Prototyping Digital Electronic Systems," *Trans. on Industrial Electronics*, vol. 54, no. 4, pp. 3069-3077, Dic. 2007.

[18] R. Marau, P. Leite, P., M. Velasco, P. Marti, L. Almeida, P. Pedreiras, and J.M. Fuertes, "Performing Flexible Control on Low-Cost Microcontrollers Using a Minimal Real-Time Kernel," *Trans. on Industrial Informatics*, vol. 4, pp. 125-133, 2008

[19] O. Laouamri and C. Aktouf, "Remote Testing and diagnosis of System-on Chips Using Network Management Frameworks," In *Proc. of Desing, Automation & Test in Europe Conference & Exhibition*, Nice, France, April 2007, pp. 1-6.

[20] S. Cuenca, A. Grediaga, H. Llorens, and M. Albero "Performance Evaluation of FPGA-Embedded Web Servers," in *IEEE International Conference on Electronics Circuits and Systems*, Marrakech, Marocco December, 2007, pp. 1187-1190.

[21] R. Koch, T. Pionteck, C. Albrecht, and E. Maehle, "An Adaptive System-on-Chip for Network Applications," in *Proc. 20th Parallel and Distributed Processing Symposium*, April 2006, pp. 8.

[22] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation," *IEEE Trans. on Industrial Informatics*, vol. 1, pp. 62-70, February 2005.

[23] F. Jammes, A. Mensch, and H. Smit, "Service-Oriented device communications using the Device Profile for Web Services," *IEEE 21st International Conference on Advanced Information Networking and Aplications*, Canada, May. 2007, pp. 956-963.

[24] K.C. Thamboulidis, G.V. Koumoutsos, and G.S. Doukas, "Semantic Web Services in the Development of Distributed control and Automation Systems," *IEEE International Conference on Robotics and Automation*, Rome, Italy, April 2007, pp. 10-14.

[25] L. Ribeiro, J. Barata, A. Colombo, and J. Jammes, "A generic Communication Interface for DPWS-based Web Services," *IEEE International Conference on Industrial Informatics*, Daejeon, Korea, July, 2008, pp. 13-16.

[26] S. Cuenca, H. Ramos, H. Llorens, and F. Maciá, "Reconfigurable Architecture for Embedding Web Services," *IEEE. 4th Southern Conference on Programmable Logic*, San Carlos de Bariloche, March 2008, pp. 119-124.

Francisco Maciá Pérez (M'08) was born in Spain in 1968. He received his engineering degree and the Ph.D. degree in Computer Science from the University of Alicante in 1994 and 2001 respectively.

He worked as System's Administrator at the University of Alicante form 1996 to 2001. He was an Associate Professor from 1997 to 2001. Since 2001, he is an Assistant Professor and currently he is the Director of the Department of Computer Science and Technology at the University of Alicante. His research interests are in the area of network management, computer networks, smart sensor networks and distributed systems, which are applied to industrial problems.

Juan Antonio Gil Martínez-Abarca was born in Spain in 1970. He received his engineering degree in Computer Science from the University of Alicante in 1994.

Since 1998, he is System's Administrator at the University of Alicante and, since 1999, he has been an Associate Professor at the Department of Computer Science and Technology at the University of Alicante. His research interests are in the area of network management, computer networks and distributed systems.

Héctor Ramos-Morillo was born in Alicante, Spain, in 1978. He received the engineering degree in computer science from the University of Alicante in 2004, where he has been working toward the Ph.D. degree in the Department of Computer Science and Technology since 2005.

He is currently a System's Administrator at the Department of Computer Science and Technology, University of Alicante. His research interests are in the area of network management, computer networks, embedded systems and smart sensor networks.

Francisco J. Mora-Gimeno (M'08) was born in Spain in 1967. He received the M.Sc. degree in computer science from the Polytechnic University of Valencia, Valencia, Spain, in 1995. Since 2004, he has been working toward the Ph.D. degree in the Department of Computer Science and Technology, University of Alicante, Alicante, Spain.

Since 2002, he has been an Assistant Professor with the Department of Computer Science and Technology, University of Alicante. His main topics of interest include intrusion detection systems, network security, computer networks and distributed systems.

Diego Marcos Jorquera (M'08) was born in Spain in 1974. He received his engineering degree in Computer Science from the University of Alicante in 1999, where he has been working toward the Ph. D. degree in the Department of Computer Science and Technology since 2004.

He is currently an Assistant Professor with the University of Alicante. His research interests are in the area of network management, computer networks, and distributed systems

Virgilio Gilart-Iglesias (M'09) was born in Spain in 1976. He received his engineering degree in computer science from the University of Alicante in 2001, where he has been working toward the Ph.D. degree in the Department of Computer Science and Technology since 2004.

He is currently an Assistant Professor with the Department of Computer Science and Technology, University of Alicante. His research interests are in the area of e-Business, Business Processes Management Systems, Service Oriented Architectures and distributed systems, which are applied to industrial domain.