Programming Wireless Sensor Networks Applications using SMEPP: A case study

Javier Barbarán, José A. Dianes, Manuel Díaz, Daniel Garrido, Luis Llopis, Ana Reyna, Bartolomé Rubio

Department of Languages and Computing Science
University of Málaga, Spain
{barbaran,jdianes,mdr,dgarrido,luisll,reyna,tolo}@lcc.uma.es

Abstract. The SMEPP middleware is a secure and generic middleware, based on a new network centric abstract model for embedded peer-to-peer (EP2P) systems based on services with security as a main design issue. It has an adaptable architecture which can be used in different devices ranging from standard PCs to motes. This paper presents a case study that uses SMEPP for programming standard and small constrained devices such as motes. The middleware is implemented over a runtime infrastructure using a specific designed component model that controls the interactions between the different elements of the middleware. The development of SMEPP applications is carried out through a neutral-language API which can be used in different languages such as Java or nesC. The paper presents a real use of the middleware in an environmental monitoring application for nuclear power plants with wireless sensor networks showing the suitability of the middleware to develop this kind of applications.

Keywords. Middleware, Security, Services, P2P, Embedded

1 Introduction

Traditional middleware architectures have focused on achieving interoperability across heterogeneous platforms and software languages. Although the platforms have evolved from their creation incorporating new specific services and profiles (real time, embedded systems, telecommunications), but their architectures have remained to a great extent and stable.

The Secure Middleware for Embedded Peer-to-Peer systems (SMEPP) project [7] is a European project financed by the European Commission in the Sixth Framework Program. One of the main works carried out in this project has been the definition of a suitable architecture for secure EP2P systems. In the case of EP2P it is necessary to take into account other clearly different aspects such as the surveillance of application behaviour, the processing infrastructure and the underlying communication networks, in order to dynamically separate both the middleware and the applications, and to achieve the appropriate quality of service.

SMEPP supports a high-level, service-oriented model to program the interaction among peers, thus hiding low-level details that concern the supporting infrastructure.

Three key features of the model are the notion of group of peers, the notion of service offered by peers (or by groups), and the concern of security.

A key factor for the success of the middleware is reusability and adaptability [2]. The SMEPP architecture has been adapted to be used in motes. The result of this adaptation is SMEPP Light [3], the version of the middleware for constrained devices. SMEPP Light can run in these devices and it can fully interoperate with the rest of devices of the middleware such as PDAs or standard PCs.

The need for adaptation to different devices and domains makes it necessary to establish a component based software architecture as well as tailored software that achieves these requirements. SMCOM is a component model specially designed to be used in the implementation of the components of SMEPP that gives a modular architecture. This component model is supported by a runtime framework adapted to the different platforms of SMEPP.

Because of the special characteristics of SMEPP, there is a huge range of applications that can be easily solved using this middleware. But a particular field that can take benefit from the use of SMEPP and wireless sensor networks (WSNs), is the related to WSNs for energy power plants, and particularly for nuclear power plants due to the special security needs. Both SMEPP and SMEPP Light configurations are being tested in a real application where sensors are deployed outside and inside the plant and can measure different environmental conditions.

The structure of the paper is as follows: Section 2 introduces the architecture of SMEPP Light and the special needs in order to achieve the interoperability with SMEPP. Following it is shown SMEPP design focusing on the component model used and some details of the runtime framework. In Section 4, the main part of this paper is presented with the case study of developing a real application using SMEPP/SMEPP Light for environmental monitoring application for nuclear power plant. The paper finishes with some conclusions.

2 SMEPP Light Architecture

A main goal of SMEPP is to provide an efficient and flexible architecture that, at the same time, must be scalable and adaptable to the resource constrained devices that will be used in the SMEPP application domains. As result, the architecture of SMEPP for WSNs is SMEPP Light. Customizing a middleware is an error-prone task and requires deep knowledge of platform and middleware design. To solve this, SMEPP has a configuration tool which allows selecting an adequate and possibly reduced set of components of SMEPP for a concrete platform such as SMEPP Light.

The SMEPP Light architecture must support the sensors requirements that are: small memory, small computing power and battery powered. The conceptual architecture for this version of SMEPP is quite similar to the full one, but since SMEPP Light offers a subset of functionalities of SMEPP, also its architecture exploits a subset of the SMEPP components. Figure 1 shows the architecture of SMEPP Light taking into account the special characteristics of motes and TinyOS [9].

The architecture is divided into non-hierarchical layers. The higher layer is composed just by Service Model Support that is the SMEPP Light API. The implementation of the API primitives will be developed following the SMEPP Runtime Component Framework philosophy but using also the Runtime Component Framework provided by TinyOS, which will be visible by all layers.

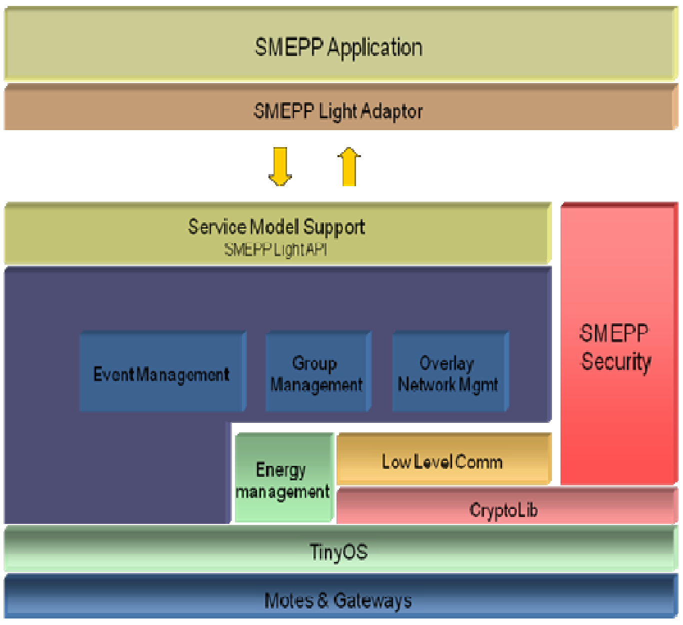

Fig. 1. SMEPP Light Architecture Overview

The SMEPP Common Services are composed by Event Management, Group Management and Overlay Network. The Group Management component manages the peers of each group and the topology of the group, and uses the group management and peer initialization primitives.

The Event Management component maps the event management primitives and it is in charge of subscription and event management. It provides commands to subscribe or unsubscribe to a certain event, generate events and set the reception mode for a certain event.

The Overlay/Network Management component implements the communication between peers. It relies on the Runtime Framework, which is based on TinyOS network/MAC component (CC24240ActiveMessage) that simply allows to address a single node, or to broadcast the message to the entire network.

The Energy Management component manages the duty cycles of the peer. In particular it manages the on/off periods of the radio interface.

The Low Level Communication offers the basic secure-communication primitives to higher layers, by using the CryptoLib library developed for motes using an efficient AES-128 algorithm suitable for wireless sensors [13, 14, 15]. In addition, it manages the keys for all the security issues related to the network and the group layers, and provides commands to encrypt/decrypt messages.

Due to the special communications system in sensors, such as radio protocol used typically, that is 802.15.4/Zigbee, the middleware has to provide some mechanism in order to overcome this problem. In order to provide communication between SMEPP and SMEPP Light (PCs or laptops and computers) it is necessary to use an adaptor application that is the called SMEPP Light Adaptor.

Security is a cross cutting aspect on all layers and is shown on the side so that its relation to other functionalities can be clearly shown in the architecture. The Cryptographic Services component provides the foundations for the higher-level security components by offering implementations of cryptographic primitives; in SMEPP Light by means of CryptoLib library, developed specially for sensors in nesC. Some devices might feature explicit support for security processing. Such device-specific support is captured in the Infrastructure Security component. Secure Topology Management includes the authentication and authorization of new peers joining the SMEPP or SMEPP Light network as well as to the protection of routing data. Finally, group security provides similar services for groups: secure joining of groups and protected communication within groups.

In [17] is shown a complete study of WSN challenges and a classification of the different solutions that exists. It is interesting to compare in some way the present architecture of SMEPP Light with other ones that are similar, and study its suitability for the case study that we are proposing. As said before, SMEPP (and SMEPP Light) belongs to Component-based methodology that proposes software construction by plugging software components [18]. The most remarkable case as a competitor in WSN for SMEPP Light is RUNES. The main differences between them are that RUNNES runs Contiki as Runtime Component Framework, and SMEPP Light runs TinyOS, and the other important difference is that SMEPP provides a secure middleware even for embedded devices, which is a characteristic less important in RUNNES.

3 SMEPP Component Model and Runtime Framework

The SMEPP applications cover several device types ranging from standard PCs to lower capabilities devices such as mobile phones or motes. This way, the election of the component model used in SMEPP is a crucial decision. Component models such as .NET [5] or JavaBeans [8] could be a good election because they are intensively used in standard workstations. However, they are not suitable for embedded systems, since they do not explicitly address memory, real-time or cost constraints. The project RUNES [1] presents an architecture for networked embedded systems based on components. However, the different underlying SMEPP paradigms (P2P and Service Orientation) and the status of the implementation make difficult the reusing of the implementation components and tools.

Middleware configurability and customization constitute an active research area. These topics are addressed from two approaches [6]:

- a dynamic approach, in which the middleware is capable of adapting to the dynamics of the system by reconfiguring itself during runtime, and

- a static approach, that focus on highly customizable middleware architectures capable of fulfilling the requirements of different distributed applications.

SMEPP follows the second approach, providing configuration, adaptation and analysis tools. One of the main contributions of our approach is that of providing analysis tools. The presentation of the tools is out of the scope of this paper.

3.1 The SMCOM Component Model

The component model used for the implementation of SMEPP components is called SMCOM (SMEPP Component Model). This component model derives from a previous work, UM-RTCOM [4] a component model specifically suitable for real-time and embedded systems. Some of its main features have prompted SMEPP to use it for the development of SMEPP components.

One of the main features of SMCOM is the using of the synchronization primitives to carry out the communication between components. This communication is performed through method interfaces and events:

- The `wait` primitive is used to wait for new invocations on services or the raising of consumed events.
- The `call` primitive is used to invoke services of SMEPP components (interconnected with the caller).
- The `raise` primitive is used to create events in an asynchronous way.

In the case of motes, SMEPP Light is implemented on top of TinyOS and nesC [10] which provide a runtime component framework. In this case, we use this component framework to implement SMCOM in such a way that it is possible to communicate components of SMEPP and SMEPP Light.

3.2 Runtime Framework

The Runtime Component Framework (RCF) is the basis for the execution of the different SMEPP and SMEPP Light components. It has to provide an Application Programming Interface (API) based on SMCOM that will be used by the different components of the SMEPP architecture. Basically, the RCF has to provide mechanisms for: connection to the RCF, component interconnection, methods invocation, events publishing, subscribing and creation.

These mechanisms are the only way that components of SMEPP can interact between them. In this sense, the runtime component framework works as a type of virtual machine or scheduler between all the layers of the SMEPP architecture where invocations of components are received, scheduled and executed.

4 Case Study: An Environmental Monitoring Application

In this section we describe a real application used to test the middleware in the field of sensor networks for nuclear power plants monitoring environmental conditions and sending it over the network to PDAs or PCs.

4.1 Application Scenario

The application is focused on the environmental monitoring and remote control of workers in industrial plants. Monitoring the effects of industrial plants on the environment is a key issue in different application domains and very especially in the field of the nuclear energy, where the security aspects of SMEPP could be validated. Moreover, in nuclear industry, exposition to ionizing radiation is a risk present in daily operation and maintenance activities for workers present in the plant. The departments of radiological protection of the nuclear facilities (e.g. power plants, industry, and healthcare) have the responsibility of controlling exposition of workers to radiation. Previous works has been carried out in the field of nuclear power plants, as it is RadMote framework [16] that has attracted the attention of the security departments of power plants. This work implements the study case that was presented in [16] but without using any middleware so, it is easy to see that it is very complex to do it as a commercial product.

The system is composed of one or more wireless sensor networks that will be deployed on some industrial environment. Particularly, the proposed system is focused on nuclear power plants. Sensors will be deployed outside and inside the plant and could measure different environmental conditions.

In the application, radiation sensors will be connected with a small device with compute capabilities (mobile devices, such as motes, PDAs or similar) that workers will wear. On the other hand, static radiation sensors (also known as sensor network's monitoring areas) and environmental monitors (measuring temperature, air quality, etc.) will be also used. It is important to provide a detailed control of some zones sensed by sensors; this will be controlled using wireless high precision cameras. These static sensors and all the personal devices will form a large ad-hoc network.

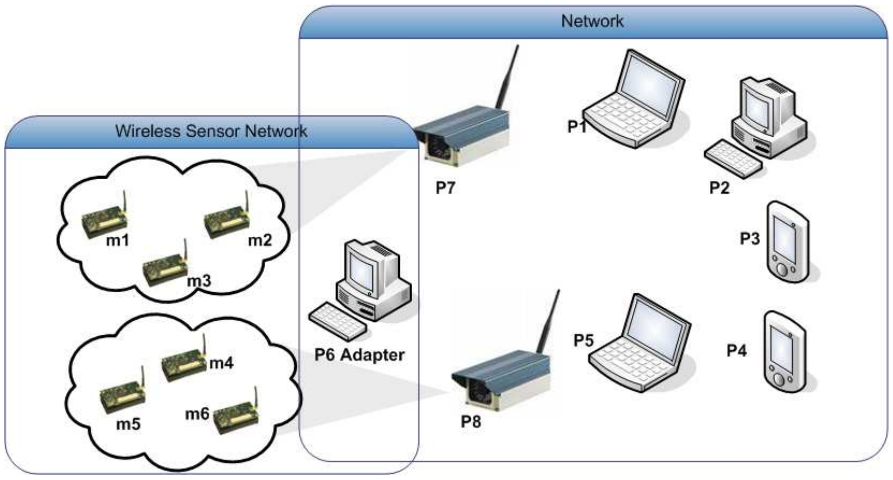

Fig. 2. Environmental Monitoring Application

In Figure 2, a deployment scheme of the proposed system is sketched, where a set of sensors are situated in fixed position into the plant, forming groups. Each sensor is running SMEPP Light, and a special peer will run SMEPP and a special adaptor, in order to enable communication between the two different SMEPP configurations, for sensors and for the rest of devices. Different critical zones are controlled by high performance cameras, that in case of alarm for high radiation values will broadcast images of the risk area to the specific peers. The SMEPP peers will form groups in order to share the environmental information and the alarms of the system.

In the following table we show the tasks for each device:

Table 1. Roles of Participants.

Devices	Description
m1, …,m6	Monitor radiologic measurements raising an event when the measure exceeds a predefined threshold.
P6 Adapter	Enables communication between SMEPPLight and SMEPP. Providing monitor service for the WSN.
P1	Responsible of supervising through video the sensor network when an event is raised.
P2, …,P5	Monitors events on the sensor network.
P7,P8	Provides video streaming service to monitor a critical situation

This application requires working with SMEPP and SMEPP Light, so the adapter takes a key role in this communication. The main efforts for programmers will be the implementation of the services, because the peer implementations using SMEPP API simplifies the code by abstracting the programmer of problems such as communication or security issues. Figure 4 shows the flow diagram of the *camera* peer and *supervisor* peers required for this application where each of the boxes of the diagrams represents an invocation of a primitive provided by SMEPP.

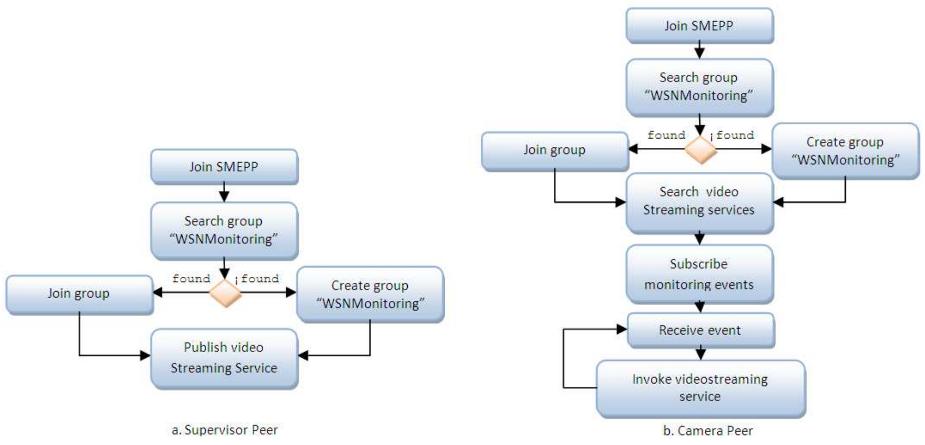
Join SMEPP
Search group "WSNMonitoring"
found
¡found
Join group
Create group "WSNMonitoring"
Publish video Streaming Service
a. Supervisor Peer
Join SMEPP
Search group "WSNMonitoring"
found
¡found
Join group
Create group "WSNMonitoring"
Search video Streaming services
Subscribe monitoring events
Receive event
Invoke videostreaming service
b. Camera Peer

Fig. 3. Camera and Supervisor Peers

In the rest of the section we are going to briefly describe some of the main points to develop a SMEPP-based application in the described scenario.

4.2 Creating Peers and the SMEPP API

The first steps related to the creation of a SMEPP application are usually the "creation" of the peer and the joining or creation of groups. Later, the peer will publish some service or will invoke services of other peers. In order to do these tasks, the SMEPP developer can use several primitives provided by an API. Of course, this API will be different depending on the target language used (e.g. Java or nesC). The following table shows some of the primitives provided by the SMEPP API [7].

Table 2. SMEPP API primitives.

NewPeer	Programs call the newPeer primitive to become peers
CreateGroup	SMEPP groups are logical associations of peers in which services can be published, and messages can be sent (either as service invocations/responses, or as raised events). Peers call the createGroup primitive to start new peer groups.
JoinGroup	Peers use the joinGroup primitive to enter a group (viz., to become members of a group)
GetServices	Returns the list of services which (optionally) matches a given pattern.
Invoke	The invoke primitive serves to call an (one-way or request-response) operation (identified by operationName) of a running service identified by id, which can be either groupServiceId, or peerServiceId, or sessionId.
Publish	Peers offer a service inside a groupId group through publish primitive.
Event	Event raises (viz., notifies to the middleware) an eventName event.
Subscribe	Entities (viz., peers or running services) use the subscribe primitive to

	register themselves as event listeners

First of all, the peer must be created, this is done though the newPeer primitive as the following code exposes (the developer provides security credentials):

```
        PeerManager peer = PeerManager.newPeer(myCredentials);
```

Once the peer is created, it is part of SMEPP (if the execution success) and it is able to invoke any other provided primitive. After this step it can be provider or consumer of services. In this scenario we have a video-streaming service and consumers of that service pushing video data obtained. To do this, the peer has to be included in a SMEPP group by creating a group or joining an existing group. The following code shows the process of creating a SMEPP group:

```
    //group creation
    GroupDescription groupDescr = new GroupDescription(
                          "WSNMonitoring",...);

    GroupId gid = peer.createGroup(groupDescr);
```

When the peer is joined to one group, it can publish services as the following section explains.

4.3 SMEPP Services and Video-streaming Service

SMEPP Services [11] can be: *state-less* or *state-full* services. The state less services do not keep track of their interactions with clients. Clients can invoke the operations of such services one or more times and in any order.

On the other hand, the state-full services keep track of their interactions with clients. We divide state-full services into *session-less* and *session-full* services. State-full session-less services are services that have only one *virtual communication channel*, which is shared by all clients, and which is active when the service is published.

In our application, we need *session-full* services to provide video-streaming services. In order to transmit video from the wireless cameras (p7 and p8) to the *supervisor* peer, SMEPP developer can carry out the following tasks:

- *Supervisor* peer offers a session-full service which repeatedly receives one-way messages (exposing an operation) carrying video data.
- Cameras start a session of the above service and flushes data by repeatedly invoking that operation. A timeout on each call allows the application to send data without necessarily waiting for an acknowledgement.

The percentage of calls which terminate without raising a timeout exception automatically identifies the amount of data which has surely been delivered. The throughput can be varied independently from the timeout, by concurrently sending data through different threads (each of them calling *Invoke* primitive on different bits of data).

In order to provide a service (e.g. video streaming service), the peer must provide an xml document describing the service, called “Contract” [12]. This contract will be used for *matching* (which will be in turn used within *discovery*) and also for *verification* and *analysis*. A contract must contain all the information that any client of the service may need to *discover*, to *instantiate*, and to *interact* with the service. The following code shows the publishing of a SMEPP service using a contract for the video-streaming service:

```
serviceId = peer.publish(gid,
                    ContractLoader.loadXMLFromFile(
                    "VideoContract.xml"),

                    new SMEPPServiceGrounding(VideoService.class),
                           myPeer, new HashMap());
```

The service must also provide its behavior. Services are programmed in Java or nesC (depending on the platform) and they must listen to the operations described in their contracts.

```
public void run() {
  ReceivedMessage message;
  Serializable[] data;
  while (true) {
    message = svc.receiveMessage("VideoServiceOperation",
                                  new Class[]{Byte[].class});
      // Store video information ...
  }
```

In the above code we can see the reception of SMEPP messages. These messages have been sent by another peer joined to the same group of the service publisher.

4.4 SMEPP Light sample application and adaptor

We have developed an application for WSN using SMEPP Light. This application has been done using nesC as the programming language and SMEPP Light as a nesC-component that makes easy the programmer's life. The use of SMEPP Light is really similar to SMEPP code that is shown previously, but the main difference is that services are very restricted here because of WSN restrictions, and services in SMEPP Light are very basic ones are implemented using nesC commands and avoiding XML in motes.

In the application, the motes create two groups, and depending on their positions each sensor will join to a particular group.

```
event void Boot.booted() {
     pID = call SensorSmepp.smepp_newPeer(netKey, netMAC, per);
     //Other actions needed when booting the mote
     call SensorSmepp.smepp_getgroups(group);
);

event void SensorSmepp.getGroups_result(uint16_t gID[]) {
     uint8_t i;

     for (i = 0; i < MAX_GROUPS_RECEIVED; i++) {
            if (gID[i] == group.ID) {
                   founded = TRUE;
         }
     }
     if (found) {
         call TimerJoin.startOneShot(10);
     }
     else {
         call TimerCreateGroup.startOneShot(10);
     }

}
```

In the code above is shown the code that is executed when the mote turns on, and first of all it creates itself as a peer inside SMEPP Light, and invokes the command asking for the getGroups primitive. This command will raise the getGroups_result as soon as getGroups primitives finishes. Following it will check if the group that the peer wants to join exists, if so it joins and otherwise it creates it.

The footprint is about 3008 bytes in RAM in the mote including the application and the middleware and it is not so big taking into account the set of functionalities that SMEPP Light offers.

SMEPP peers have to subscribe to WSN events in order to receive environmental information so they can make the correct actions depending on the value sensed. To do this, we need to use the so called SMEPP Light adaptor that is a Java-based application that faces with WSN connection; it is component-based as well, so depending on the type of base station (stargate device, MIB520 base station) the adaptor will provide the appropriate component in order to manage the connection with sensors. The adaptor basically manages the subscriptions between SMEPP and SMEPP Light, converting 802.11 b/g packets into 802.15.4/Zigbee packets for the motes, and receiving the events from WSN and sending it to SMEPP peers.

5 Conclusions

SMEPP is a middleware specifically designed for the development of EP2P systems. In this paper we have presented a case study for monitoring environmental conditions in power plants and remote control of workers inside the plant. In order to help the reader to understand how to develop this sample application we explained briefly the component model and the Runtime Component Framework that allows its execution.

The focus of the SMEPP proposal was on the integration of security services and network quality. From the security point of view, security was taken into an approach from the very beginning of the project and it is now integrated into the current version of the middleware. Quality of service has also being taken into account in the design. The specification can be achieved in terms of service contracts and the component model has been designed to be able to analyze and monitor the real-time behaviour of all the components, including those related to basic communication support.

Acknowledgments. This work is fully supported by EU funded project FP6 IST-5-033563

References

1. Costa, P. et al: The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems. 16th Annual IEEE Internacional Symposium on Personal Indoor and Mobile Radio Communications IMRC'05), Berlin, Germany. September 2005.
2. Tarvainen, P.:.An Approach to Evaluate the Adaptability of Software Architectures. Proceedings of the 5th Workshop on System Testing and Validation, (ICSSEA 2007), Paris, France, December 2007.
3. Vairo C.; Albano M.; Chessa S.: A Secure Middleware for Wireless Sensor Networks. Middleware for Mobile Embedded Peer-to-Peer Systems 1st International Workshop. July 2008.
4. Díaz, M.; Garrido, D.; Llopis, L.; Rus, F.; Troya, J.M.: UM-RTCOM: An Analyzable Component Model for Real Time Distributed Systems. J. Syst. Software (2007), doi:10.1016/j.jss.2007.07.010
5. Microsoft .Net Web site. http://msdn.microsoft.com/netframework/
6. RTZen. UCI-DOC Group. http://doc.ece.uci.edu/rtzen/
7. SMEPP Web site. http://www.smepp.org
8. Sun Microsystems. Enterprise JavaBeans Specification 2.1. 2005.
9. TinyOS Web site. http://www.tinyos.net
10. Gay D.; Levis, P.; Von Behren R.; Welsh, M.; Brewer, E.; Culler, D.: A Holistic Approach to Networked Embedded Systems. Proceedings of Programming Language Design and Implementation, 2003.
11. Brogi, A.; Popescu, R.; Gutiérrez, F.; López P., Pimentel, E.: A Service-Oriented Model for Embedded Peer-to-Peer Systems. In proceedings of the 6th International Workshop on the Foundations of Coordination Languages and Software Architectures, Lisbon, Portugal, September 8, 2007.
12. Brogi, A.; Popescu, R.: Workflow Semantics of Peer and Service Behaviour. Proceedings 2nd IEEE International Symposium on Theoretical Aspects of Software Engineering, June 2008, Nanjing, China.
13. Roman R.; Fernandez-Gago, M.C.; Lopez, J.:. Featuring Trust and Reputation Management Systems for Constrained Hardware Devices. Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems (Autonomics 2007), Rome (Italy), October 2007.
14. Baek, J.; Foo, E.; Tan, H.; Zhou, J.: Securing Wireless Sensor Networks - Threats and Countermeasures. Chapter 3 of "Security and Privacy in Wireless and Mobile Computing", ISBN 978-1905886-906, Troubador Publishing, 2008.
15. López, J. Zhou, J.: Wireless Sensor Network Security. Cryptology & Information Security Series, IOS Press, April 2008.
16. Barbarán, J.; Díaz, M.; Esteve, I.; Rubio, B.; RadMote: A mobile framework for radiation monitoring in nuclear power plants. In Proceedings of the XXI International CESSE. May 2007.
17. Rubio. B; Díaz, M.; Troya. J.M.; Programming approaches and challenges for wireless sensor network. In Second International Conference on Systems and Networks Communications. August 2007.
18. G. HeineMan and W. Council. Component-Based Software Engineering: Putting the pieces together. Addison-Wesley: Reading, MA, 2001.