

EMI²lets: A Reflective Framework for Enabling AmI

Diego López de Ipiña, Iñaki Vázquez, Daniel García, Javier Fernández and Iván García

Faculty of Engineering
University of Deusto
Avda. de las Universidades, 24
48007 Bilbao, SPAIN

{dipina, ivazquez}@eside.deusto.es, {dgarcia, jafernan, ivgarcia}@ctme.deusto.es

Abstract

An interesting new application domain for handheld devices may be represented by Ambient Intelligence (AmI), where they can be used as intermediaries between us and our surrounding environment. Thus, the devices will behave as electronic butlers who assist us in our daily tasks, by interacting with the smart objects (everyday objects augmented with computational services) surrounding us. This paper proposes a framework aimed to transform mobile devices into universal remote controllers of smart objects.

1. Introduction

Ambient Intelligence (AmI) [10] defines an interaction model between us and a context-aware environment, which adapts its behaviour *intelligently* to our preferences and habits, so that our daily life is facilitated and enhanced.

Current PDAs and mobile phones are equipped with continuously increasing processing and storage capabilities, better and more varied communications mechanisms (Bluetooth [1], Wi-Fi, GPRS/UMTS) and increasingly capable multimedia capture and playback facilities. Moreover, they are far more easily programmable [8][5] [12] than ever before.

Mobile devices equipped with Bluetooth, built-in cameras, barcode or RFID readers can be considered as *sentient* devices [7], since they are aware of what *smart objects* are in their whereabouts. A smart object is an everyday object (e.g. door, classroom) or a device augmented with some accessible computational service. Once a mobile device discovers a nearby smart object, it can induce changes on its behaviour.

We deem that mobile devices will play a key role in AmI, since they can act as facilitators or intermediaries between us and the environment. In other words, mobile devices can act as our

personal electronic butlers, facilitating and enhancing our daily activities, and even acting on our behalf based on our profiles or preferences.

In this paper, we describe the design and implementation of EMI²lets, a software framework to facilitate the development and deployment of AmI scenarios.

2. EMI²: an AmI architecture

In order to make the AmI vision reality, a good starting point should be the definition of suitable software architectures and frameworks specially catered for it. The EMI² (Environment to Mobile Intelligent Interaction) architecture is our proposed solution.

EMI² defines a multi-agent software architecture, where agents modelling the different roles played by entities in AmI, communicate and cooperate to enhance and facilitate the user interactions with her *smart environment*.

We understand by smart environment a location where the objects present within (smart objects) are augmented with computing services. For instance, a cinema may be enhanced with a mobile phone locally accessible (Bluetooth) ticket booking service, so preventing the user from long queuing to purchase tickets.

Figure 1 portrays the main components of the EMI² architecture. We distinguish three main types of agents:

- *EMI²Proxy*: is an agent representing the user, which runs on the user's mobile device (PDA or mobile phone). It acts on behalf of the user, adapting/controlling the environment for him, both *explicitly*, under the user's control, or *implicitly*, on its own judgement based on the profiles, preferences and previous interactions of the user with the environment.
- *EMI²Object*: is an agent representing any device or physical object (e.g. vending machine, door) within a smart environment

augmented with computational services, i.e. the capacity to adapt its behaviour based on ambient conditions or user commands.

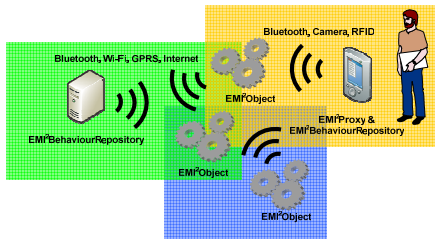


Figure 1. The EMI² Architecture

- *EMI²BehaviourRepository*: is an agent where knowledge and intelligence are combined to support sensible adaptation. EMI²Objects may require the assistance of an external EMI²BehaviourRepository to coordinate their own adaptation according to the user's preferences, behaviour patterns or even the explicit commands received from an *EMI²Proxy*. The user's mobile device can also be powered with an internal EMI²BehaviourRepository.

2.1. Active and passive mechanisms

A concrete agent can influence the environment, and thus, its constituent agents' state, via *active* (explicit interaction) or *passive* (implicit interaction) methods.

Active methods are those in which the agent explicitly commands other agents to change their state or perform an action. For example, when a user enters a building, a sensor identifies him and commands the lift to be ready at the ground floor. When the user stands by his office door his mobile phone commands the electric lock to open.

Passive methods are those in which an agent disseminates certain information (profiles, preferences), expecting that other agents change their state or perform an action at their discretion to create a more adapted environment. Using passive methods an agent does not command the target agents to do anything concrete, it simply publishes information preferences expecting the others to react changing their state in a positive way. Passive mechanisms are less intrusive than active methods, but they are less predictable and significantly more complex to implement.

In passive methods, the particular set of information to disseminate by the agent is dependant on the configuration of the environment in which is going to be published. Therefore, a discovery and negotiation process must take place among the entities in an environment in order to achieve an adapted behaviour for the users present within. In previous work, we have tackled these passive influence [14] and context negotiation [15] issues.

2.2. Active influence over EMI²Objects

The purpose of this paper is to design and implement a middleware to provide universal active influence capabilities to our mobile devices over the surrounding smart objects.

The two minimum features such middleware addresses are: (1) a mechanism to discover through ad-hoc or wireless networking the computing services exported by surrounding smart objects, and (2) a mechanism to interact with those discovered services, so that the represented objects adapt to the user's commands.

The current state of the art in discovery and interaction platforms falls into three categories [4]. Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini [1]. After discovery, the service (either a proxy or the full service) is downloaded onto the mobile device where it then operates. Secondly, solutions where the discovery protocols are integrated with specific interaction protocols, which are used to invoke the service after the service has been discovered, e.g. Universal Plug and Play (UPnP) [13]. Finally, interaction independent discovery protocols such as the SLP [3].

One of the following communication mechanisms is normally used to interact with a discovered service: remote method invocation, publish-subscribe or asynchronous messaging. For the purpose of this work we will concentrate on the remote method invocation paradigm, since it accommodates to the most popular mechanisms for distributed computing such as CORBA or Web Services.

In what follows we explain the design and implementation of an Aml-enabling middleware which addresses the service discovery and interaction aspects required for active influence (explicit invocation) on EMI²Objects.

3. The EMI²lets platform

EMI²lets is the result of mapping the EMI² architecture into a software development platform to enable AmI scenarios. It is specially suited for active interaction mechanisms. However, it has been designed so that passive mechanisms may be incorporated in the future.

EMI²lets is a development platform for AmI which addresses the intelligent discovery and interaction among EMI²Objects and EMI²Proxies. EMI²lets follows a Jini-like mechanism by which once a service is discovered, a proxy of it (an EMI²let) is downloaded into the user's device (EMI²Proxy). An EMI²let is a mobile component transferred from a smart object to a nearby handheld device, which normally offers a graphical interface to interact or influence the behaviour of a surrounding smart object.

The EMI²lets platform addresses three main aspects:

- *Mobility*, seamlessly to the user it encounters all the services available as he moves and selects the best possible mechanism to communicate with them. The EMI²let platform selects the communication means with best trade-off between performance and cost. For example, if Wi-Fi and Bluetooth are available, the former is chosen, however if GPRS/UMTS and Bluetooth are available, the latter is chosen.
- *Interoperability*, the EMI²lets, i.e. the software components downloaded from EMI²Objects to EMI²Proxies, are agnostic of the target device type, e.g. PC, a PDA or a mobile phone.
- *AmI* is the application domain that has driven the design of EMI²lets. This platform provides the infrastructure and software tools required to ease the development and deployment of mobile context-aware applications.

The objectives established for the design and implementation of the EMI²lets platform are:

- Transform mobile devices into remote universal controllers of the smart objects in an AmI environment.
- Enable both local (Bluetooth, Wi-Fi) and global access (GPRS/UMTS) to the smart objects in an AmI environment, seamlessly adapting to the most suitable underlying communication mechanisms

- Develop extensible middleware independent of a particular discovery or interaction mechanism. Abstract the programmer from the several available or emerging discovery (Bluetooth SDP or wireless UPnP discovery) and interaction mechanisms (RPC or publish/subscribe).
- Make use of commonly available hardware and software in mobile devices, without demanding the creation of proprietary hardware, or software protocols.
- Generate software representatives (proxies) of smart objects which can be run in any platform, following a "write once run in any device type" philosophy. For instance, the same EMI²let should be able to run in a mobile phone, a PDA or a PC.

3.1. The EMI²lets vision

Figure 2 shows a possible deployment of an EMI²lets-powered environment. A group of handheld devices running the EMI²let Player and hosting the EMI²let runtime can discover and interact with the software representatives (EMI²lets) of surrounding EMI²Objects. An EMI²Object may be equipped with enough hardware resources to host an EMI²let Server, or alternatively a group of EMI²lets associated to different EMI²Objects may all be hosted within an autonomous version of an EMI²let Server.

The EMI²let Server acts as a repository of EMI²Objects. It publishes the services offered by the hosted EMI²Objects, transfers them on demand to the requesting EMI²let Players, and, optionally acts as running environment for the EMI²let server-side facets.

Some EMI²lets may directly communicate with their associated EMI²Objects in order to issue adaptation commands. However, often a specialised piece of software may need to be developed which is far too complex to be implemented in the embedded hardware with which a smart object may be augmented. For those cases, it will be more convenient to delegate those cumbersome and heavy computing tasks to the server-side (back-end) counterpart of an EMI²let. The EMI²let on the hand-held device will communicate with its server-side counterpart in the EMI²let Server by means of the EMI²Protocol. For example, a light-controlling EMI²let could

communicate with its EMI²let server-side, which would issue X10 commands over the power line.

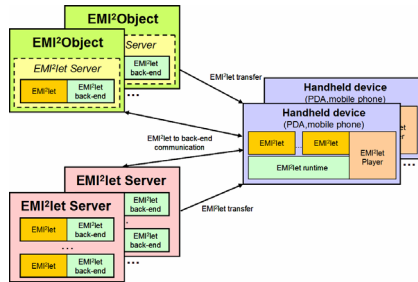


Figure 2. EMI²lets possible configuration.

3.2. Internal architecture

The EMI²lets platform consists of the following elements:

1. A programming framework defining a set of classes and rules that every EMI²let component must follow.
2. An integrated development environment, named EMI²let Designer, which simplifies the development of EMI²lets, both its client- and (optional) server-side.
3. A runtime environment installed on EMI²let-aware devices for executing downloaded code.
4. An EMI²let Player to discover, download, verify and control the execution of a EMI²let. A version of the player is available for each device type which can host EMI²lets, e.g. PDA, mobile phone or PC.
5. An EMI²let Server which acts as repository of EMI²lets and as running environment of EMI²lets server-sides.

In order to achieve the previously mentioned design objectives, we have created the layered software architecture shown in Figure 3. Programmers only deal with the first layer, the *EMI²let Abstract Programming Model API*, to develop the software counterparts of smart objects. This layer offers a set of generic interfaces (abstract classes) covering the main functional blocks of a mobile sentient application:

1. *Discovery* interface to undertake the search for available EMI²lets independently of the discovery mechanisms used underneath.
2. *Interaction* interface to issue commands over the services discovered.

3. *Presentation* interface to specify the graphical controls and events that represent the look and feel of an EMI²let.
4. *Persistency* interface to store EMI²let-related data in the target device.

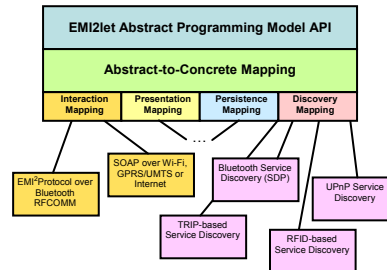


Figure 3. EMI²lets Internal Architecture

The *EMI²let Abstract-to-Concrete Mapping* layer translates the invocations over the generic interfaces to the appropriate available mechanisms both in the mobile device and the EMI²Objects in the environment. The discovery, interaction, presentation and persistency abstractions encapsulate the concrete discovery, interaction, presentation or persistency models used. They implement an API for performing service discovery and interaction, graphical interface generation and data persistency independent of the actual implementation in the target device.

On deployment the code generated through these abstract interfaces is linked to the concrete implementations of the classes used which are part of the EMI²let runtime in the target device.

In the process of associating a generic invocation to an actual one, the *EMI²let Abstract-to-Concrete Mapping* will be responsible of selecting the actual mapping (or group of mappings) which best matches the invocation type. For example, if a downloaded EMI²let is installed on a device where both Bluetooth and GPRS communication are available, the abstract-to-concrete layer will have to choose one of those mechanisms to issue commands. Thus, if the mobile device is still within Bluetooth range of the EMI²let server-side, then it will translate the invocation into an EMI²Protocol message transported over Bluetooth RFCOMM. Otherwise, it will invoke via GPRS the generic web service (with methods corresponding to the EMI²Protocol commands) implemented by an EMI²let back-end.

With regards to the presentation abstraction, we have defined a minimum set of graphical controls with which generate the graphical interface of an EMI²let. Some examples are: EMI²Panel, EMI²Button or EMI²TextBox. This enables us to create EMI²let graphical interfaces agnostic of the target mobile device. Thus, when a programmer creates an EMI²Button, it is translated into a button control in a PC or a PDA, but into a menu option in a mobile phone.

The operation of the functional mapping plug-ins is ruled by an XML configuration file, which states whether a plug-in may be run concurrently with other plug-ins of the same type or in isolation. In the latter case, a priority is assigned to each plug-in which will determine which of the plug-ins to select when several of them are available. Both the *Abstract-to-Concrete Mappings* and the *Functional Mapping* layers and plug-ins will be linked to the arriving EMI²let in an EMI²let Player (see Figure 4).



Figure 4. EMI²let Player.

3.3. Reflection

The use of Reflection is paramount in the EMI²lets platform. It enables an EMI²let Player to verify that the code arriving as part of an EMI²let complies with the EMI²lets framework, and most importantly, is a piece of code which can be trusted. Every EMI²let downloaded is signed with a private key only shared by the EMI²let designer and the player.

After verification, the player can start the EMI²let by invoking the methods defined in the EMI²let base class, extended by every EMI²let. The methods defined by this class follow similar signatures to a J2ME [5] MIDlet class:

- `start`, starts or resumes the execution of a downloaded EMI²let.
- `pause`, pauses its execution.
- `destroy`, destroys it.

In addition, the EMI²let class includes some EMI²lets-specific methods such as:

- `getUUID`, returns the unique identifier of an EMI²let, under which state related to an EMI²let can be persisted.
- `setProperty/getProperty`, sets or gets the properties associated to a EMI²let. For instance, the EMI²let.Durable property is set to `true` when an EMI²let has to be cached in the player, so that it can be executed again in the future. Otherwise, an EMI²let is wiped out from the player either when its execution is completed or it is out of range, cannot access, the EMI²Object it represents.
- `notifyDisconnected`, offers an EMI²let the possibility of being aware when the controlled EMI²Object cannot be accessed.
- `getAddresses`, enables the EMI²let-hosting player to retrieve the EMI²let server-side addresses. For instance, an EMI²let backend may be accessed both through a Bluetooth address or a url pointing to a web service.

3.4. The EMI²lets implementation

The most noticeable part of our implementation is the assembly fusion undertaken at the player side merging the arriving EMI²let assembly with EMI²let library installed in each target device. This library represents the player's runtime, i.e. the abstract-to-concrete layer and the four mappings implementation with their corresponding plug-in modules. In other words, the assembly code downloaded is linked dynamically (late bound) with the runtime installed in the target device. The .NET's System.Reflection namespace has provided us the support to enable this.



Figure 5. EMI²let lifecycle.

Figure 5 illustrates the life cycle of an EMI²let from its development to its deployment. In our approach active .NET code developed on a PC

through the EMI²let Designer (see Figure 6) is uploaded into an EMI²let Server, from where it is later downloaded and executed in the context of an EMI²let Player. After its execution an EMI²let is cached or removed from the Player.

4. An EMI²let discovery plug-in

As an example of an EMI²lets plug-in, a novel service discovery mechanism based on the TRIP [6] tag-based visual system is described.

A factor that limits the use of Bluetooth as an underlying networking technology for publicly accessible mobile services is that its device discovery model takes a significant (sometimes unbearable) time. The discovery process in Bluetooth is divided into two main phases: (1) device discovery and (2) service discovery in the devices discovered. In an error-free environment, the device discovery phase must last for 10.24s if it is to discover all the devices [1].

In order to improve the delay in service discovery, we propose a tag-based service selection, which bypasses the slow Bluetooth Device Discovery process, similar to [9].

The TRIP visual tags are circular barcodes (*ringcodes*) with 4 data-rings and 20 sectors. A visual tag, large enough to be detected by a mobile device tag reading software, is shown in Figure 7. The ringcode is divided into:

- One *sync-sector* used to specify the beginning of the data encoded in a tag.
- Two *checksum-sectors* used to encode a 8-bit checksum, which detects decoding errors and corrects three bit errors, and
- Seventeen *data-sectors* which encode 66 bits of information.

The information in a TRIP tag is encoded in anti-clockwise fashion from the sync sector. The sync-sector differs from the rest by presenting black in its four data rings sections. Each sector encodes a hexadecimal digit comprising the values 0 to D. The E hexadecimal number is only permitted in the sync sector. Given the 17 data encoding sectors, the range of valid IDs is from 0 to $15^{17}-1$ ($98526125335693359375 \approx 2^{66}$).

The TRIP tags were designed to work well with the low-resolution fixed-focal-length cameras found on conventional CCTV systems. Consequently, they are also suitable for the low-quality built-in cameras of mobile devices [7]. In

fact, TRIP ringcodes are more reliably recognized than linear (UPC) barcodes, which demand far higher image resolutions. TRIP works reliably with 160x120 pixel images taken at a distance of 5-30cm from the tags which label the EMI²Objects in an environment.

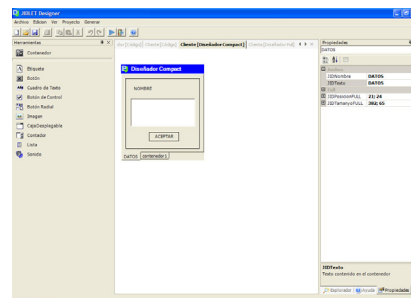


Figure 6. EMI²let Designer.

We have implemented the TRIP tag reading software both for Java J2ME and Compact.NET mobile devices. Both implementations work reliably, although further work on their performance is required. Currently, our J2ME implementation for a Nokia 6630 processes 1 fps and a Compact.NET implementation for a TSM 500 Pocket PC 2 fps.



Figure 7. A tag encoding 66 bits of data.

4.1. Encoding EMI²lets' addresses

We have applied TRIP tags to encode the Bluetooth address of an EMI²let Server and an identifier to select an EMI²Object in the server. Likewise, we have also used those tags to encode tiny urls (see <http://tinyurl.com>) which point to an EMI²Object in an EMI²let Server. The tiny url server is currently generating 6 character-long identifiers, whilst we can encode up to 8 characters. The scheme followed to encode an EMI²let address in a TRIP ringcode is:

- Two bits have been allocated to encode the address type, i.e. whether it is a Bluetooth (00) or an Internet tiny url (01) address.
- For Bluetooth, 48 bits are dedicated to encode the BD_ADDRESS of an EMI²let Server, and the remaining 16 bits to encode a unique identifier to select a specific EMI²let.
- For Internet, we have used the 66 bits available to encode a tiny url, containing the address of an EMI²let. For example, the tiny url identifier 8ggaj maps to the url <http://wap.deusto.es>.

Noticeably, the TRIP visual tags do not only improve service discovery but they also serve to call the user attention about the smart objects available in an environment.

5. EMI²lets applications

The Restaurant EMI²let, see Figure 8, can be used to make more efficient the food ordering process in a busy restaurant. While waiting in a restaurant, a mobile device running the EMI²let player could discover, download and execute that EMI²let. By means of it, the user could perform his order and automatically know the amount he will have to pay. This same application could be used the following day, when the user is at home, to request a take-away delivery from the same restaurant.

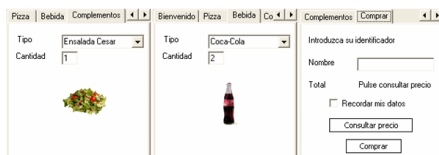


Figure 8. Restaurant example

Other EMI²lets developed have allowed us to perform as diverse tasks as buying a parking ticket while on the parking via Bluetooth and later renewing it via GPRS from our office, controlling the electronic devices and lights of a room, offering a voice synthesized bus arrival notification for blind people or providing subtitles on mobile phones for deaf people attending a conference.

In conclusion, the EMI²lets platform transforms our mobile phone or PDA into a universal remote controller which enables us to

obtain extra information and change the behaviour of everyday objects, independently of whether we are currently in an object's local environment or remotely from it.

6. Related work

The EMI²lets platform presents some resemblance to the Smoblets software framework proposed by [11]. Both frameworks offer the possibility to download into a mobile device the software representatives of objects located in a smart space. However, Smoblets are thought to operate when they are only within range of the smart object they represent, whereas EMI²lets can remain at the user's terminal, even when he is far away from the smart object. This allows the user to control that smart object anytime and anywhere, both using local (Bluetooth) and global (GPRS) communication mechanisms. Furthermore, the main application of Smoblets is to transform mobile devices into execution platforms for code downloaded from smart items with limited processing resources, whereas EMI²lets are mainly thought to transform mobile devices into hosts of smart object proxies, which simplify their remote control.

The EMI²lets framework's layered software architecture has been inspired by the ReMMoC framework [4]. However, EMI²lets does not only address the service discovery and interaction issues of mobile context-aware applications. It also tackles the graphical presentation and persistency aspects commonly used in those applications. Moreover, as main innovation, the code generated for an EMI²let is independent of the target platform type where it will be run (PC, PDA or mobile phone). This is due to the fact that our layered software architecture follows a "write once run in any device type" philosophy.

Other authors [9] have also used TRIP tags to encode addresses of smart objects. Our data encoding strategy, using the same number of rings as them, achieves better error correction capabilities (from 2 to 3 bits) and has a bigger encoding capacity (from 63 to 66 bits).

7. Conclusion and further work

This work has described the design and implementation of a novel reflective framework

which provides universal active influence capabilities to mobile devices over the smart objects in an environment. This framework presents the following features:

- Transforms mobile devices into universal remote controllers of smart objects.
- Enables both local and global access of those smart objects, i.e. anywhere and at anytime.
- Independent and extensible to the underlying service discovery and interaction, graphical representation and persistence mechanisms.
- Enables Aml using conventional readily-available hardware and software tools.
- EMI²lets are developed following a “write once run in any device type” philosophy.

In future work we want to add more sophisticated service discovery and context negotiation features between EMI²let Players and Servers, following the WebProfiles model described in [15]. In addition, we want to enable the cooperation of EMI²Objects, for instance, through the incorporation of distribution shared tuple space.

Acknowledgements

This work has been financed by a 2004-05 SAIOTEK grant from the Basque Government and the Cátedra de Telefónica Móviles España at the University of Deusto (<http://www.ctme.deusto.es>).

References

- [1] Arnold K., O’Sullivan B. et al. The Jini Specification, Addison Wesley, 1999.
- [2] Bluetooth Specification version 1.1, <http://www.bluetooth.com>
- [3] Czerwinski S., Zhao B. et al. An architecture for a Secure Service Discovery Service. In Proceedings of MobiCom’99, August 1999.
- [4] Grace P., Blair G. S. and Samuel S. A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments, Mobile Computing and Communications Review, vol. 9, no. 1, pp. 2-14, January 2005.
- [5] Java 2 Platform, Micro Edition (J2ME), <http://java.sun.com/j2me/>, Sun Microsystems, Inc, 2005
- [6] López de Ipiña, D., Mendonça P. and Hopper A. TRIP: a Low-cost Vision-based Location System for Ubiquitous Computing, Personal and Ubiquitous Computing, vol. 6, no. 3, pp. 206-219, May 2002.
- [7] López de Ipiña D., Vázquez I. and Sainz D. Interacting with our Environment through Sentient Mobile Phones. Proceedings of IWUC-2005, May 2005, pp. 19-28, ISBN 972-8865-24-4.
- [8] Mobile Developer Center, <http://msdn.microsoft.com/mobility/>, Microsoft Corporation, 2005
- [9] Scott D., Sharp R., Madhavapeddy A. and Upton E. Using Visual Tags to Bypass Bluetooth Device Discovery, ACM Mobile Computing and Communications Review, vol.9, no.1, pp 41-52, January 2005.
- [10] Shadbolt N. Ambient Intelligence. IEEE Intelligent Systems, vol. 2, no.3, July/August 2003.
- [11] Siegemund, F. and Krauer T. Integrating Handhelds into Environments of Cooperating Smart Everyday Objects
- [12] Symbian OS – the mobile operating System, <http://www.symbian.com/>, Symbian Ltd., 2005
- [13] The Universal Plug and Play Forum, <http://www.upnp.org/>, 2005.
- [14] Vázquez, J.I., López de Ipiña, D. An Interaction Model for Passively Influencing the Environment, Adjunct Proceedings of EUSAI. Eindhoven, The Netherlands. November 2004.
- [15] Vázquez, J.I. and López de Ipiña D. An HTTP-based Context Negotiation Model for Realizing the User-Aware Web. IWI 2005. Chiba, Japan. May 2005.