A Reflective Middleware for Controlling Smart Objects from Mobile Devices

Diego López de Ipiña, Iñaki Vázquez, Daniel García, Javier Fernández and Iván García [(1)]

[(1)] 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

Mobile devices are mainly used for communication, entertainment, and as electronic assistants. However, their increasing computational, storage, communicational and multimedia capabilities make them suitable for previously unexpected scenarios such as Ambient Intelligence (AmI). Thus, mobile devices may be used as intermediaries between us and the smart objects (everyday objects augmented with computational services) in our surroundings. This paper describes the design and implementation of a middleware to transform mobile devices into universal remote controllers of smart objects.

1. Introduction

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 facilities. Moreover, they are far more easily extensible (Compact.NET [2], J2ME [3] or Symbian [4]) than ever before.

Mobile devices equipped with Bluetooth, built-in cameras, barcode or RFID readers transform into sentient devices [5], i.e. they are aware of what smart objects are in their whereabouts. A smart object is an everyday object or device (door, classroom, parking booth) augmented with some accessible computational service. Once a mobile device discovers a nearby smart object, it can induce changes on its behaviour.

Bearing in mind the technical progress and sentient features of last generation mobile devices, it is natural to think that they will play a starring role in the context of Ambient Intelligence (AmI) [6]. An obvious application will be their use as facilitators or intermediaries between us and a smart environment. In other words, mobile devices can behave 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 EMI^{2}lets (Environment to Mobile Intelligent Interaction applets), a software framework to facilitate the development and deployment of mobile context-aware applications for AmI environments.

The structure of the paper is as follows. Section 2 describes EMI^2, a software architecture modelling AmI. Section 3 introduces the EMI^{2}lets platform, a partial materialisation of the EMI^2 architecture, which simplifies both the creation of software representatives for everyday objects and their controlling proxies deployed in mobile devices. Section 4 proposes a novel mechanism to discover smart objects based on visual tags used in EMI^{2}lets. Section 5 lists some interesting applications produced with the EMI^{2}lets platform. Section 6 mentions some related work. Finally, section 7 offers some conclusions and suggests further work.

2. EMI^2: an AmI architecture

Regardless of the continuous progress in the research topics which contribute to the AmI vision, namely Ubiquitous Computing [7], context-awareness [8] or intelligent user interfaces [9], we are still far away from its materialisation. However, the definition of suitable software architectures and frameworks specially catered for AmI may be a good starting point. The EMI^2 (Environment to Mobile Intelligent Interaction) architecture is our proposed solution.

EMI^2 defines a multi-agent software architecture, where agents of different types, modelling the different roles played by entities in AmI, communicate and cooperate to fulfil a common goal, i.e. to *enhance and facilitate the user interactions with her smart environment*.

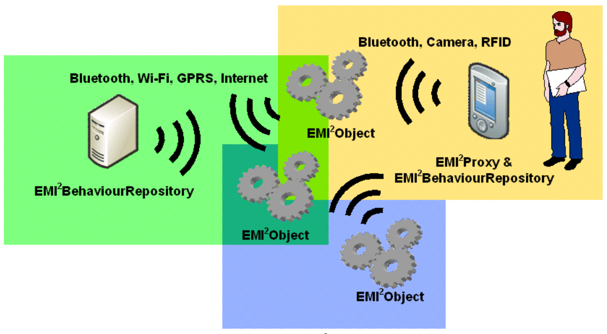

Bluetooth, Camera, RFID
Bluetooth, Wi-Fi, GPRS, Internet
EMI^{2}Object
EMI^{2}BehaviourRepository
EMI^{2}Proxy & EMI^{2}BehaviourRepository
EMI^{2}Object
EMI^{2}Object

Figure 1: *The EMI^2 Architecture.*

We understand by smart environment a location, either indoors or outdoors, 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. Similarly, the door of our office may be augmented with an access control service which demands a user passing by to enter a PIN in her mobile to be given access.

Figure 1 depicts the main components of the EMI^2 architecture. We distinguish three main types of agents:

- *EMI^{2}Proxy*: 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.

- *EMI^{2}Object*: is an agent representing any device or physical object (e.g. vending machine, door, ticket box) within a smart environment augmented with computational services, i.e. the capacity to adapt its behaviour in response to ambient conditions or user commands. An EMI^{2}Object cooperates to achieve its goals with other EMI^2 agents.
- *EMI^{2}BehaviourRepository*: is an agent where knowledge and intelligence are combined to support sensible adaptation. EMI^{2}Objects may require the assistance of an external EMI^{2}BehaviourRepository to coordinate their own adaptation according to the user's preferences, behaviour patterns or even the explicit commands received from an EMI^{2}Proxy. The user's mobile device can also be powered with an internal EMI^{2}BehaviourRepository loaded with personal information and profiles in order to minimize the interaction with the owner, i.e. adopting implicit adaptation.

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 in front of his office door his mobile phone commands the electric lock to open. Active methods can be implemented with any well-known distributed computing technology capable of issuing commands. These commands will be transported in a local context by bearers such as Bluetooth or Wi-Fi and in a global context by GPRS/UMTS.

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/broadcasts 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 this paper we concentrate on the design and implementation of a middleware to provide universal active influence capabilities to our mobile devices over the surrounding smart objects in our environment. We have tackled the issue of passive influence over smart objects in previous work [10].

2.2. The Need for an Active Influence Middleware

The minimum requirements a middleware for active influence must address are: (1) a mechanism to discover through ad-hoc or wireless networking the computing services made available by surrounding smart objects, and (2) a mechanism to interact with those discovered services, so that the objects they represent adapt to the user's preferences and commands.

The current state of the art in discovery and interaction platforms falls into three categories [12]. Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini [13]. 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. A good example of this is Universal Plug and Play (UPnP) [14]. Finally, there are interaction independent discovery protocols such as Service Location Protocol [15].

Once a service is discovered one of the following communication mechanisms is normally used: 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.

3. The EMI^{2}lets platform

EMI^{2}lets is the result of mapping the EMI^2 architecture into a software development platform devised to enable AmI scenarios. This platform is specially suited for active interaction mechanisms. However, it has been designed so that passive mechanisms may be incorporated in the future.

EMI^{2}lets is a development platform for AmI which addresses the intelligent discovery and interaction among EMI^{2}Objects and EMI^{2}Proxies. EMI^{2}lets follows a Jini-like mechanism by which once a service is discovered, a proxy of it (an EMI^{2}let) is downloaded into the user's device (EMI^{2}Proxy). An EMI^{2}let is a mobile component transferred from a smart object (EMI^{2}Object) to a nearby handheld device, which usually offers a graphical interface for the user to interact with its associated smart object.

The EMI^{2}lets 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. In other words, the EMI^{2}let platform ensures that an EMI^{2}Proxy is always using 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 EMI^{2}lets, i.e. the software components downloaded from EMI^{2}Objects to EMI^{2}Proxies, 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 EMI^{2}lets. This platform provides the infrastructure and software tools required to develop and deploy smart objects and their controlling proxies.

The objectives set for the design and implementation of the EMI^{2}lets platform are:

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

- Develop middleware independent of a particular discovery or interaction mechanism. Abstract the programmer from the several available discovery (Bluetooth SDP or wireless UPnP discovery) and interaction mechanisms (RPC or publish/subscribe). Likewise, allow this middleware to easily adapt to newly emerging discovery (RFID) and interactions means.
- Utilise commonly available hardware and software features in mobile devices, without demanding the creation of proprietary hardware, or software protocols. Essentially, reuse current infrastructure and integrate it for its application to the AmI domain.
- Generate software representatives (proxies) of smart objects which can be run in any platform, following a "write once run in any device type" philosophy. The same EMI^{2}let should run in a mobile, a PDA or a PC.

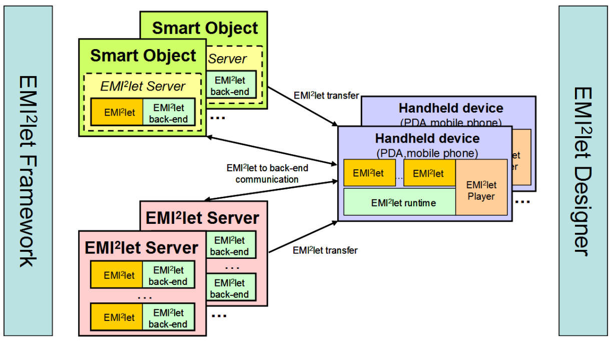


Figure 2: EMI^{2}lets Platform.

3.1. The EMI^{2}lets vision

Figure 2 shows a possible deployment of an EMI^{2}let-aware environment. A group of handheld devices running the EMI^{2}let Player and hosting the EMI^{2}let runtime can discover and interact with the software representatives (EMI^{2}lets) of surrounding smart objects. A smart object may be equipped with enough hardware resources to host an EMI^{2}let Server, or alternatively a group of EMI^{2}lets associated to different smart objects may all be hosted within a standalone EMI^{2}let Server.

The EMI^{2}let Server acts as a repository of smart objects. It publishes the services offered by the hosted objects, transfers them on demand to the requesting EMI^{2}let Players, and, optionally, acts as running environment for the EMI^{2}let server-side facets.

Some EMI^{2}lets may directly communicate with their associated smart objects in order to issue adaptation commands. However, often specialised 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 computing tasks to the server-side (back-end) counterpart of an EMI^{2}let. The EMI^{2}let on the hand-held device will communicate with its server-side counterpart in the EMI^{2}let Server by means of the EMI^{2}Protocol. For example, a light-controlling EMI^{2}let could communicate with its EMI^{2}let server-side, which would issue X10 commands over the power line to switch on the associated lamps (smart objects).

3.2. Internal architecture

The EMI^{2}lets platform consists of the following elements:

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

In order to achieve the EMI^{2}lets design objectives, we have created the layered software architecture shown in Figure 3. Programmers only deal with the first layer, the *EMI^{2}let 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 an EMI^{2}let:

1. *Discovery* interface to undertake the search for available EMI^{2}lets independently of the discovery mechanisms used underneath.
2. *Interaction* interface to issue commands over the services discovered, independently of the available communication mechanisms.
3. *Presentation* interface to specify the graphical controls and events that represent the look and feel of an EMI^{2}let.
4. *Persistency* interface to store EMI^{2}let-related data in the target device.

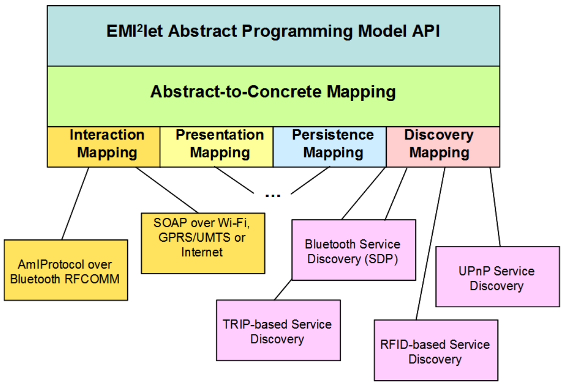

EMI²let Abstract Programming Model API
Abstract-to-Concrete Mapping
Interaction Mapping
Presentation Mapping
Persistence Mapping
Discovery Mapping
...
AmIProtocol over Bluetooth RFCOMM
SOAP over Wi-Fi, GPRS/UMTS or Internet
Bluetooth Service Discovery (SDP)
UPnP Service Discovery
TRIP-based Service Discovery
RFID-based Service Discovery

Figure 3: EMI^{2}lets Internal Architecture.

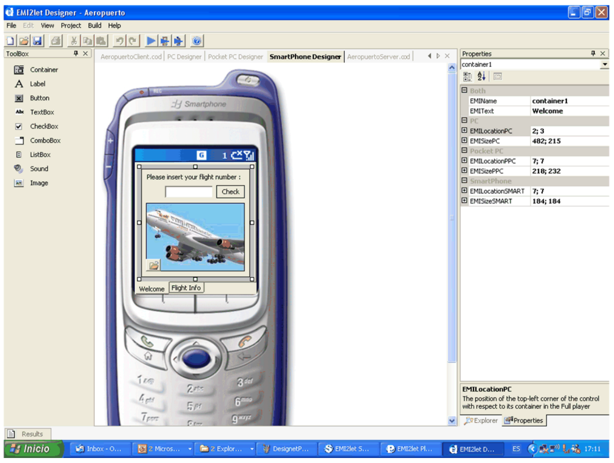
The *EMI^{2}let Abstract-to-Concrete Mapping* layer translates the invocations over the generic interfaces to the appropriate mechanisms available both in the mobile device and the smart objects in the environment. The discovery, interaction, presentation and persistency abstractions encapsulate the corresponding concrete 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 EMI^{2}let runtime in the target device.

The architecture of the EMI^2 framework is very flexible and extensible because it is based on the concept of plug-in. A plug-in is simply an implementation of one of the available abstractions or functional mappings. In the process of associating a generic invocation to an actual one, the *EMI^{2}let Abstract-to-Concrete Mapping* will be responsible of selecting the actual plug-in (or group of plug-ins) which best matches the invocation type. For example, if a downloaded EMI^{2}let 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 EMI^{2}let server-side, then it will translate the invocation into an EMI^{2}Protocol message transported over Bluetooth RFCOMM. Otherwise, it will invoke via GPRS the generic web service (with methods corresponding to the EMI^{2}Protocol commands) implemented by every EMI^{2}let server-side. Similarly, if a mobile device is Bluetooth and Wi-Fi capable, it will use both Bluetooth SDP and UPnP service discovery to concurrently search for smart objects.

The plug-in selection is made according to an XML configuration file which states whether a plug-in may be run concurrently with others of the same type or in isolation. In the latter case, a priority is assigned to each plug-in which will determine which one to select when several are available. We plan to establish a more sophisticated plug-in configuration model in future work.

With regards to the presentation abstraction, we have defined a minimum set of graphical controls with which we can generate the graphical interface of an EMI^{2}let. Some examples of the classes defined are: `EMI2Panel`, `EMI2Button` or `EMI2TextBox`. This enables us to create EMI^{2}let graphical interfaces which are agnostic of the target mobile device. For instance, 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. Still, with the help of the EMI^{2}let Designer (see Figure 4) we can rearrange the layout of the graphical controls of an EMI^{2}let for each of the three target device types supported: PC, PDA and mobile phone. The EMI^{2}let Designer also generates the source code templates for an EMI^{2}let and its server-side counterpart, which can then be edited and compiled to generate the EMI^{2}let binaries ready to be discovered and downloaded.


EMI2let Designer - Aeropuerto
File Edit View Project Build Help
ToolBox
Container
Label
Button
TextBox
CheckBox
ComboBox
ListBox
Sound
Image
AeropuertoClient.cod | PC Designer | Pocket PC Designer | SmartPhone Designer | AeropuertoServer.cod
Smartphone
Please insert your flight number :
Check
Welcome
Flight Info
Properties
Explorer
Properties
Results
Inicio

container1

Both	
EMIName	container1
EMIText	Welcome
PC	
EMILocationPC	2; 3
EMISizePC	482; 215
Pocket PC	
EMILocationPPC	7; 7
EMISizePPC	218; 232
SmartPhone	
EMILocationSMART	7; 7
EMISizeSMART	184; 184

EMILocationPC
The position of the top-left corner of the control with respect to its container in the Full player

Figure 4: EMI^{2}let Designer.

3.3. Implementation details

The use of reflection is paramount in the EMI^{2}lets platform. It enables an EMI^{2}let Player to verify that the code arriving as part of an EMI^{2}let complies with the EMI^{2}lets framework, and most importantly, is a piece of code which can be trusted. Every EMI^{2}let downloaded is signed with an MD5 checksum encrypted by a private key only shared by the EMI^{2}let designer and player.

After verification, the player can start the EMI^{2}let by invoking the methods defined in the `EMI2let` base class, inherited by every EMI^{2}let. The methods defined by this class closely resemble to the ones provided by a J2ME [3] `MIDlet` class:

- `start`, starts or resumes the execution of a downloaded EMI^{2}let.
- `pause`, pauses its execution.
- `destroy`, destroys it.

In addition, the `EMI2let` class includes some EMI^{2}lets-specific methods such as:

- `getUUID`, returns the unique identifier of an EMI^{2}let.
- `setProperty/getProperty`, sets or gets the properties associated to a EMI^{2}let. For instance, the `EMI2let.Durable` property is set to `true` when an EMI^{2}let has to be cached in the player after its execution. Thus, it can be executed again in the future. Otherwise, an EMI^{2}let is wiped out from the Player either when its execution is completed or it is out of range, cannot access, the EMI^{2}Object it represents.
- `notifyDisconnected`, informs an EMI^{2}let when the EMI^{2}Object that it controls cannot be accessed any longer.
- `getAddresses`, enables the EMI^{2}let Player to retrieve the addresses where an EMI^{2}let server-side is available. For instance, it may be accessible both through a Bluetooth address or a URL pointing to a web service.

The first reference implementation of EMI^{2}lets has used Microsoft .NET, a framework which fully supports reflection through the `System.Reflection` namespace. Moreover, the .NET platform addresses software development for all the client hardware platforms considered in EMI^{2}lets, i.e. PC, PDA and mobile phone. The EMI^{2}lets presentation controls devised have been based on the ones provided by Compact.NET, the least common multiple.

The most noticeable part of our implementation is the assembly fusion undertaken at the player side merging the arriving EMI^{2}let assembly with the EMI^{2}let library installed in each target device. This library represents the player's runtime, i.e. the abstract-to-concrete layer and the interaction, discovery, presentation and persistency mappings implementation with their corresponding plug-in modules. In other words, the code downloaded is linked dynamically (late bound) with the runtime installed in the target device. The .NET class `System.Reflection.Assembly` is heavily used in this process.

4. An EMI^{2}let discovery plug-in

A good example of an EMI^{2}let plug-in is the service discovery mechanism based on the TRIP [16] tag-based visual identification system which we have developed.

A factor that limits the use of Bluetooth as an underlying networking technology for publicly accessible mobile services is that its device discovery process takes a significant (sometimes unbearable) time. The discovery process in Bluetooth is divided into two main phases: (1) device discovery, i.e. what other devices are accessible via Bluetooth, and (2) service discovery, i.e. what services are offered by the discovered devices. 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 speed up service discovery, we have devised a tag-based content/service selection mechanism, which bypasses the slow Bluetooth device discovery process. Our approach is inspired by the work of [17].

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 5. The ringcode is divided into: (1) one *sync-sector* used to specify the beginning of the data encoded in a tag, (2) two *checksum-sectors* used to encode an 8-bit checksum, which detects decoding errors and corrects three bit errors, and (3) 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. 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 $\approx 2^{66}$).

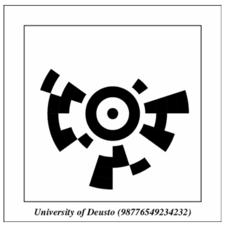


Figure 5: A tag encoding 66 bits of data.

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 very well suited for the low-quality built-in cameras of mobile devices, as we suggested in [5]. In fact, our experience shows that the 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 5-30 cms from the tags which label the smart objects in an environment. We have implemented the TRIP tag reading software for Compact.NET devices. It achieves 2 fps in a TSM 500 Pocket PC.

4.1. Encoding EMI^{2}lets' addresses

We have used TRIP tags to encode the Bluetooth address of an EMI^{2}let Server and an identifier to select a smart object in that server. Likewise, we have also used those tags to encode tiny urls (see http://tinyurl.com) which point to a smart object in an EMI^{2}let Server. The tiny url server is currently generating 6 character-long identifiers, whilst we can encode up to 8 characters. For example, the identifier `8ggaj` maps to the url http://wap.deusto.es. Two bits of a TRIP ringcode are used to encode an EMI^{2}let address type, `00` for Bluetooth and `01` for an Internet tiny url. For Bluetooth, 48 bits are dedicated to encode the BD_ADDRESS of an EMI^{2}let Server, and the remaining 16 bits to encode a unique identifier for a specific EMI^{2}let. For Internet, 64 bits are available to encode a tiny url, containing the tiny url identifier of an EMI^{2}let server-side.

Noticeably, the TRIP visual tags do not only improve service discovery but they also serve to make the user more aware of the smart objects available in the environment.

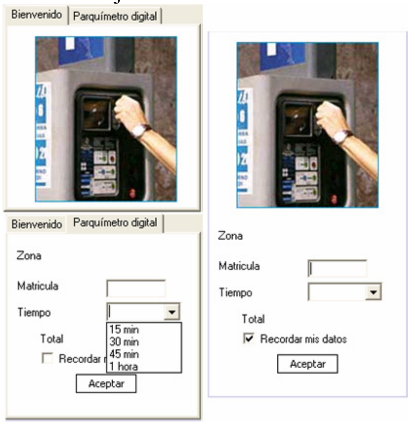


Figure 6: Parking EMI^{2}let for PDA (left) and PC (right).

5. EMI^{2}lets applications

The Parking EMI^{2}let, see Figure 6, is a concept application developed with EMI^{2}lets. It shows how a physical object in an outdoors space can be augment with AmI features. It is meant to be deployed in any street parking booth, where we can purchase tickets to park our car for a limited period of time. Often, we have to keep returning to the parking place to renew the ticket so that the local police force does not issue a fine for parking time expiration. Thanks to the EMI^{2}lets platform a user could discover, download (from the ticket booth) and install a parking EMI^{2}let which would help him solve this situation. With the downloaded EMI^{2}let the user could purchase parking tickets via Bluetooth every time the user is in the parking place, and remotely via GPRS when the EMI^{2}let warns her (at her office) that its parking ticket is about to expire. This scenario shows the EMI^{2}lets platform capability to enact an action over a smart object both locally, while in the environment, or remotely, far away from the environment. This application is an example of a durable EMI^{2}let.

Other EMI^{2}lets developed have allowed us to perform as diverse tasks as ordering a meal in a busy restaurant, controlling the electronic devices and lights of a room, offering a spoken bus arrival notification for blind people or providing subtitles on mobile phones for deaf people attending a conference.

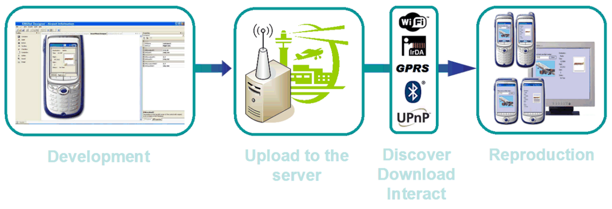


Figure 7: EMI^{2}let Development and Deployment.

All the EMI^{2}lets developed have followed the development and deployment cycle shown in Figure 7. As we can see the EMI^{2}lets platform provides tools to assist the programmer in the whole development (EMI^{2}let Designer and EMI^2 framework) and deployment (EMI^{2}let Player and Server) of smart objects, turning the creation of smart spaces into a much simpler task.

6. Related work

The EMI^{2}lets platform presents some resemblance to the Smoblets software framework proposed by [18]. Both frameworks allow the download of software representatives of objects located in a smart space into a mobile device. However, Smoblets are thought to operate when they are only within range of the smart object they represent. On the contrary, EMI^{2}lets 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, Wi-Fi) and global (GPRS/UMTS) 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 EMI^{2}lets are mainly thought to transform mobile devices into hosts of smart object proxies, which simplify their remote control.

The EMI^{2}lets framework's layered software architecture has been inspired by the ReMMoC framework [12]. However, EMI^{2}lets 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 a main innovation, the code generated for an EMI^{2}let 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 [17] have also used tags (based on our 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 higher encoding capacity (from 63 to 66 bits).

7. Conclusion and further work

This work has described the design and implementation of a novel reflective middleware which provides universal active influence capabilities to mobile devices over smart objects, independently of the objects location. This framework presents the following features:

- Transforms mobile devices into universal remote controllers of smart objects.
- Enables both local and global access to 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 AmI using conventional readily-available hardware and software tools.
- Follows a "write once run in any device type" development philosophy.

In future work we want to add more sophisticated service discovery and context negotiation features between EMI^{2}let Players and Servers, following the WebProfiles model described in [11]. In addition, we want to enable the cooperation of smart objects, for instance, through the creation of a distributed shared tuple space. Finally, we intend to incorporate Semantic Web features to our framework, which may move the user "out of the loop" in the EMI^{2}lets discovery and execution process, as suggested in [19].

References

[1] (2005) *Bluetooth Specification version 1.1*, http://www.bluetooth.com.
[2] (2005) *Mobile Developer Center*, http://msdn.microsoft.com/mobility/, Microsoft Coorporation.
[3] (2005) *Java 2 Platform, Micro Edition (J2ME)*, http://java.sun.com/j2me/, Sun Microsystems, Inc.
[4] (2005) *Symbian OS – the mobile operating System*, http://www.symbian.com/, Symbian Ltd.
[5] López de Ipiña D., Vázquez I. and Sainz D. (2005) *Interacting with our Environment through Sentient Mobile Phones*, in *Proceedings of* IWUC-2005, ICEIS 2005, Miami, pp. 19-28, ISBN 972-8865-24-4.
[6] Shadbolt N. (2003) *Ambient Intelligence,* in *IEEE Intelligent Systems*, vol. 2, no.3.
[7] Weiser M. (1991) *The computer for the twenty-first century*, in *Scientific American*, vol. 265, no. 3, pp. 94-104.
[8] Hopper, A. (2000) *The Clifford Paterson Lecture, Sentient Computing*, in *Philosophical Transactions of the Royal Society London*, vol. 358, no. 1773, pp. 2349-2358.
[9] Chorianopoulos K. et al. (2003) *Intelligent user interfaces in the living room*, in *Proceedings of Iinternational conference on Intelligent user interfaces*, pp. 230–232.
[10] Vázquez, J.I., López de Ipiña, D. (2004) *An Interaction Model for Passively Influencing the Environment*, in *Adjunct Proceedings of EUSAI*, Eindhoven, The Netherlands.
[11] Vázquez, J.I. and López de Ipiña D. (2005) An HTTP-based Context Negotiation Model for Realizing the User-Aware Web, in IWI 2005, Chiba, Japan. May 2005.
[12] Grace P., Blair G. S. and Samuel S. A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments, in Mobile Computing and Communications Review, vol. 9, no. 1, pp. 2-14.
[13] (1999) Arnold K., O'Sullivan B. et al. *The Jini Specification*, Addison Wesley.
[14] (2005) *The Universal Plug and Play Forum*, http://www.upnp.org/.
[15] Czerwinski S., Zhao B. et al. *An architecture for a Secure Service Discovery Service*, in *MobiCom'99*, August 1999.
[16] López de Ipiña, D., Mendonça P. and Hopper A. (2002) *TRIP: a Low-cost Vision-based Location System for Ubiquitous Computing*, in *Personal and Ubiquitous Computing*, vol. 6, no. 3, pp. 206-219.
[17] Scott D. et al. (2005) *Using Visual Tags to Bypass Bluetooth Device Discovery*, in *ACM Mobile Computing and Communications Review*, vol.9, no.1, pp 41-52.
[18] Siegemund, F. and Krauer T. (2004) Integrating Handhelds into Environments of Cooperating Smart Everyday Objects, in *Proceedings of the 2nd European Symposium on Ambient Intelligence. Eindhoven*, The Netherlands.
[19] Lassila O. and Adler M. (2003) Semantic Gadgets: Device and Information Interoperability