Journal of Universal Computer Science, vol. 12, no. 3 (2006), 297-314
submitted: 30/10/05, accepted: 15/1/06, appeared: 28/3/06 © J.UCS

EMI²lets: A Reflective Framework for Enabling AmI

Diego López de Ipiña, Juan Ignacio Vázquez, Daniel García
Javier Fernández, Iván García, David Sainz, Aitor Almeida
(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
{dsainz, aalmeida}@tecnologico.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, which always accompany us, will behave as electronic butlers who assist us in our daily tasks, by interacting with the smart objects (everyday objects augmented with computational services) in our whereabouts. In order to achieve such goal, this paper proposes an AmI-enabling framework providing two main functions: a) facilitate the development and deployment of smart objects and b) transform mobile devices into universal remote controllers of those smart objects.

Keywords: Ambient Intelligence, Pervasive Computing, Middleware, Smart Spaces
Categories: C.2.4, C.2.6, D.2.2, D.2.6, D.2.11, D.2.12

1 Introduction

Ambient Intelligence (AmI) [Shadbolt, 03] 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. AmI is a new research discipline aiming the creation of intelligent reactive spaces that serve the user. In order to achieve this, AmI combines the knowledge of several related research fields such as Ubiquitous Computing and Communication, Context-Aware Computing, Artificial Intelligence and intelligent interfaces.

Current PDAs and mobile phones are equipped with continuously increasing processing and storage capabilities, better and more varied communications mechanisms (Bluetooth [Bluetooth, 05], Wi-Fi, GPRS/UMTS) and increasingly capable multimedia capture and playback facilities. Moreover, they are far more easily programmable [Symbian, 05][Microsoft, 05] [Sun, 05a] than ever before.

Mobile devices equipped with Bluetooth, built-in cameras, GPS receivers, barcode or RFID readers can be considered as sentient devices [López de Ipiña, 05] [Rohs, 05], since they are aware of what smart objects are in their whereabouts. A smart object [Beigl, 01] 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 EMI^{2}lets, a software framework to facilitate the development and deployment of AmI scenarios.

The structure of the paper is as follows. Section 2 describes EMI^2, a software architecture modelling both passive and active interaction mechanisms for AmI. Section 3 details 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 deployable in mobile devices. Section 4 proves the extensibility features of the EMI^{2}lets platform with the description of an interesting discovery plug-in based on circular barcodes developed for EMI^{2}lets. Section 5 illustrates some example applications developed with the help of EMI^{2}lets framework. Section 6 shows some performance results achieved by the current implementation of EMI^{2}lets. Section 7 overviews some related work. Finally, section 8 offers some conclusions and suggests further work.

2 EMI^2: an AmI architecture

In order to make the AmI vision reality, a good starting point may be the definition of suitable software architectures and frameworks specially catered for it. The EMI^2 (Environment to Mobile Intelligent Interaction) architecture is our proposed solution.

EMI^2 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 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* or smart object: 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.
- *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.

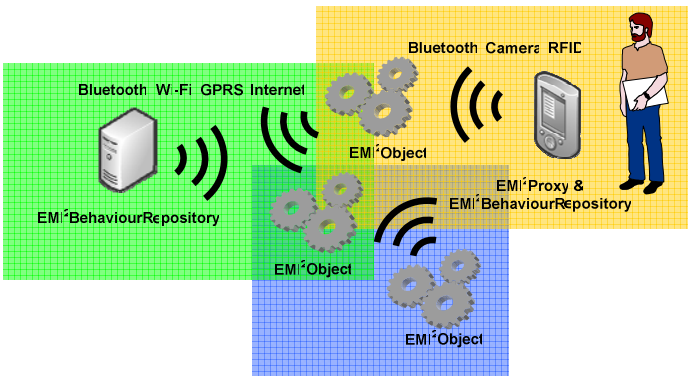

Figure 1: The EMI² architecture

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 [Vázquez, 04] and context negotiation [Vázquez, 05] issues.

2.2 Active influence over Smart Objects

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 requirements such middleware must address 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 [Grace, 05] [Zhu 05]. Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini [Sun, 05b]. 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) [UPnP, 05]. Finally, interaction independent discovery protocols such as the SLP [Czerwinski, 99].

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 AmI-enabling middleware which addresses the service discovery and interaction aspects required for active influence (explicit invocation) on smart objects.

3 The EMI^{2}lets platform

EMI^{2}lets is the result of mapping the EMI^2 architecture into a .NET-based 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.

EMI^{2}lets is a .NET-based 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 to a nearby handheld device, which normally offers a graphical interface to interact or influence the behaviour of a surrounding 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. The EMI^{2}let 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 EMI^{2}lets, i.e. the software components downloaded from smart objects to EMI^{2}Proxies, are agnostic to 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 ease the development and deployment of mobile context-aware applications.

The objectives established for the design and implementation of the EMI^{2}lets 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 EMI^{2}let should be able to run in a mobile phone, a PDA or a PC.

3.1 The EMI^{2}lets vision

Figure 2 shows a possible deployment of an EMI^{2}lets-powered 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 EMI^{2}Objects. An EMI^{2}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 EMI^{2}Objects may all be hosted within an autonomous version of an EMI^{2}let Server.

The EMI^{2}let Server acts as a repository of EMI^{2}Objects. It publishes the services offered by the hosted EMI^{2}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 EMI^{2}Objects 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 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.

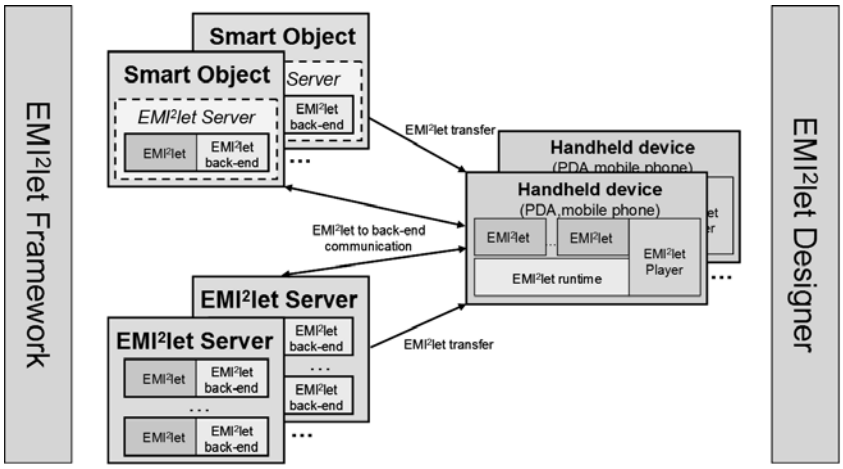

Figure 2: EMI²lets possible configuration

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 a 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.

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 provide an API for performing service discovery and interaction, graphical interface generation and data persistence independent of the actual implementation of that API in the target device.

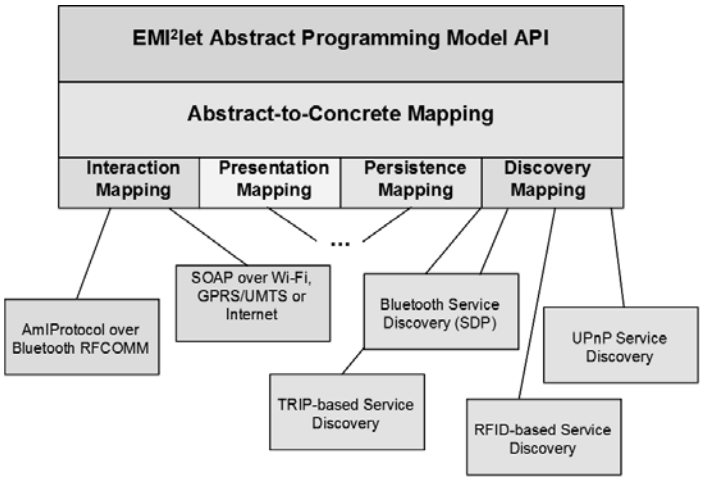

Figure 3: EMI²lets internal architecture

On deployment the code developed by means of the mentioned API abstract interfaces (Abstract Programming Model) is late bound to the concrete implementations of those interfaces (Concrete Mappings) 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 for 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 EMI²2let 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 the graphical interface of an EMI²let is generated. 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, running in any of the four supported device types (see Figure 4).

Figure 4: EMI²let Players in the PC, Windows Mobile and Web platforms

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 those found in a J2ME [Sun, 05a] `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 removed 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 back-end 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 the 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.

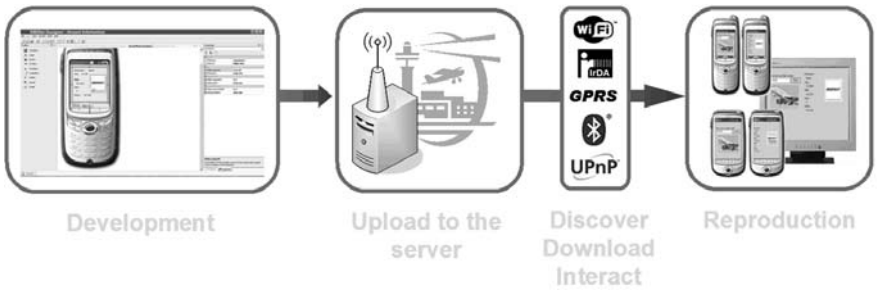

Figure 5: EMI²let lifecycle

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

In order to prove the extensibility features of the EMI²lets platform, an interesting example of an EMI²lets plug-in developed is described. This plug-in accommodates to the discovery abstraction of EMI²lets and it is based on the TRIP [López de Ipiña, 02] tag-based visual system.

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) amount of 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 [Bluetooth, 05].

In order to reduce the delay in service discovery, we propose a tag-based service selection, which bypasses the slow Bluetooth Device Discovery process, similar to [Scott, 05].

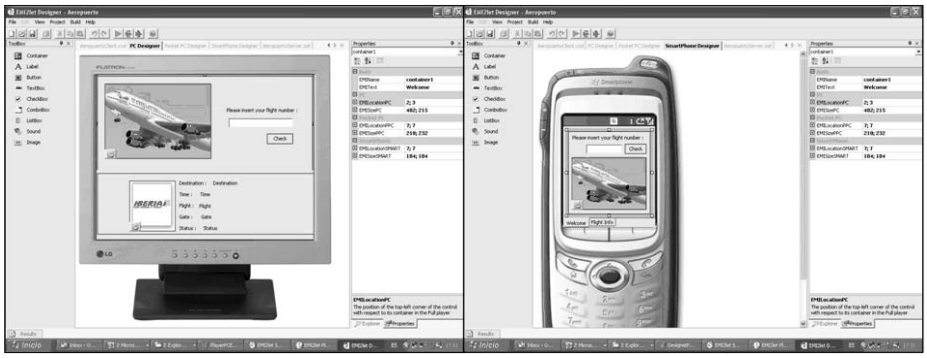

Figure 6: EMI²let Designer

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 [López de Ipiña, 05]. 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.

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 user's attention about the smart objects available in his surroundings.

5 EMI²lets applications

We have developed EMI²lets targeted to the following application domains: a) accessibility, b) home/office automation, c) industry, and d) public spaces.

In the domain of accessibility we have developed EMI²lets which associated to a bus stop offer a voice synthesized bus arrival notification for blind people or provide subtitles on the mobile phones of people attending to a conference. These applications demonstrated how simple it is to transform a physical space (bus stop or conference hall) into a more accessible environment thanks to the EMI²lets platform.

In the home and office automation domain some EMI²lets have been created that enable to control the lights, a music system (in fact the Windows Media Player in a PC) or a Pan/Tilt/Zoom security camera at a home or office, from mobile devices.

As far as the industry domain is concerned we have developed an EMI²let which allows us to control from our mobile device a robot equipped with a communications module supporting both Bluetooth and GPRS. When co-located with the robot our EMI²let uses the Bluetooth communication channel. When we are far away from the location of the robot, the EMI²let uses the GPRS channel to communicate with the robot. The communication channel choice is undertaken by the EMI²lets runtime autonomously.

Finally, on what we call the "public space" domain, we have created EMI²lets which allow us to control a parking booth, order food in a restaurant or review the departure time and gate of a plane in an airport. Those EMI²lets show how a physical object in an outdoors space can be augmented with AmI features. For example, the Parking EMI²let 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 EMI²lets platform a user could discover, download (from the ticket booth) and install a parking EMI²let which would help him solve this situation. With the downloaded EMI²let the user could purchase parking tickets via Bluetooth while in the parking, and remotely via GPRS when the EMI²let warns her (at her office) that its parking ticket is about to expire. This scenario shows one of the biggest virtues of EMI²lets, its capability to enact an action over an EMI²Object both locally, while in the environment, or remotely, far away from the environment.

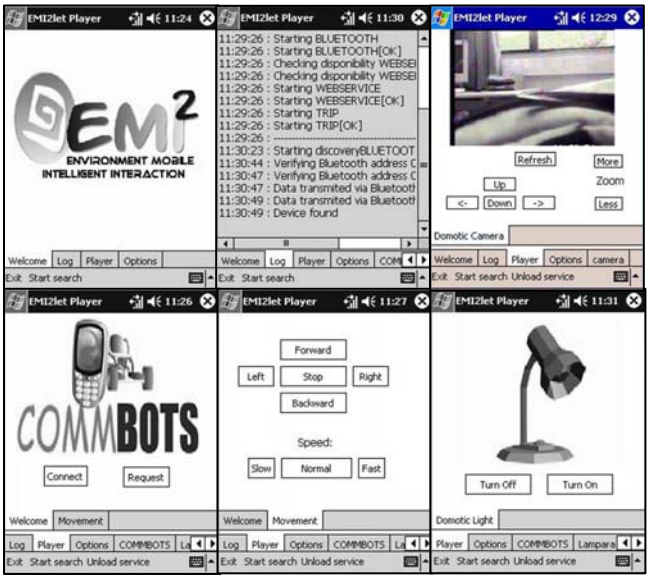

Figure 8: EMI²lets running on a PDA

Figure 8 and Figure 9 show three of the previously described EMI²lets in action running in a PDA and a mobile phone, respectively. The EMI²lets shown allow a user to control from his mobile device a robot, a lamp or a PTZ security camera. Something remarkable about the EMI²lets platform is that in the development of those EMI²lets we have written the code only once, independently of the target device where they will run. This is due to the "write once run in any device type" philosophy followed by our system.

Figure 9: EMI²lets running on a mobile phone

6 EMI²lets Performance Results

In order to asses the performance of our current implementation of the EMI²lets platform we have carried out two tests on a TSM 500 PDA with Bluetooth, Wi-Fi and GPRS support:

1. A comparative measurement illustrating the different latencies experienced during an EMI²let discovery, download and communication with its server-side, bearing in mind the nature of the communication channel used (Wi-Fi, Bluetooth or GPRS).
2. A comparative measurement to determine the average data rate achieved depending on whether we use Bluetooth, Wi-Fi or GPRS to transfer data between an EMI²let and its server-side.

Figure 10 shows that the discovery process based on UPnP over Wi-Fi is much faster than connecting directly to the IP address and port number of an EMI²let Server to enquire about its installed EMI²lets over GPRS or undertaking Bluetooth discovery. However, once the Bluetooth discovery has concluded the download of an EMI²let code and the exchange of information between an EMI²let and its server-side is much better than through GPRS and only worse to Wi-Fi which has a much better transfer rate.

Figure 11 shows the effective data transfer rates obtained over the three wireless communication mechanisms we have used in EMI^{2}lets. Obviously, the data transfer rate obtained through Wi-Fi is the best, whereas Bluetooth offers the second best behaviour.

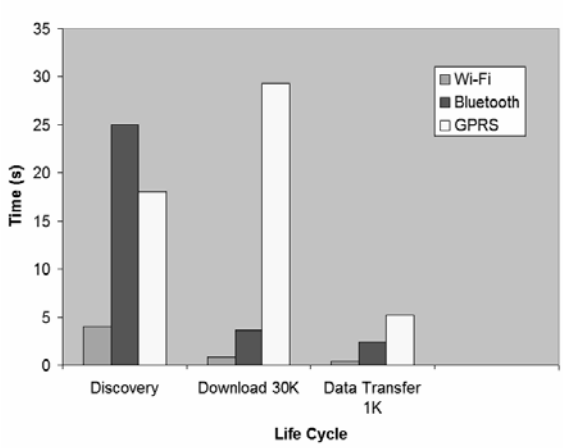

Figure 10:EMI^{2}lets communication costs

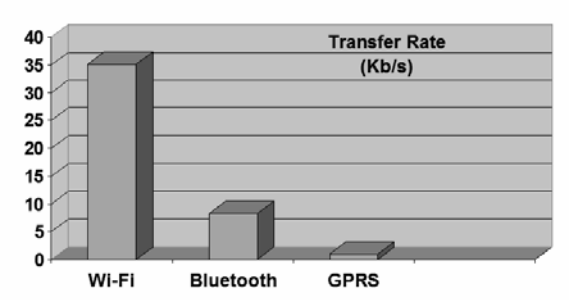

Figure 11: Effective data transfer rate in EMI^{2}lets

7 Related work

The EMI^{2}lets platform presents some resemblance to the Smoblets software framework proposed by [Siegemund, 04]. 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 EMI^{2}lets 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 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 [Grace, 05]. 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 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.

The Pebbles project [Myers, 01] is exploring how handheld devices, such as PDAs and mobile phones, can be used when they are communicating with a "regular" personal computer (PC), with other handhelds, and with computerized appliances such as telephones, radios, microwave ovens, automobiles, and factory equipment. Pebbles shares with EMI^{2}lets the goal of transforming handheld devices into universal remote controllers. Moreover, it adopts a similar architecture where a player in the handheld device communicates with server-side intermediaries to control the operation of the underlying smart objects. However, the main difference is that Pebbles defines a Personal Universal Controller (PUC) Specification Language through which the device parameters that can be controlled are specified. The PUC language does not only specifies these control parameters but also a protocol for transmitting changes to the state of these parameters between the appliance and the controller. Essentially, the player in Pebbles has to interpret the PUC specification published by a device in order to generate its interface, i.e. applies an XSLT-like transformation to obtain from the XML representation of the controlling parameters a set of graphical controls. Unfortunately, Pebbles focuses all its work on the presentation and interaction process and has not solved the important service discovery issues that EMI^{2}lets has addressed. Moreover, in EMI^{2}lets is the designer of a smart object the one who decides which will be the best look and feel of the graphical interface to control the smart object, whereas in Pebbles that decision is left to the player itself.

The Obje software architecture [Edwards, 05] is an interconnection technology that enables digital devices and services to interoperate over both wired and wireless networks – even when they know almost nothing about one another. Their goal is to be able of simply plug new device types into the network and all existing peers on the network will be able to use them. Similarly to EMI^{2}lets, Obje is agnostic to the underlying discovery and communication mechanisms. It also defines four simple abstractions that remain constant and all peers on the network understand: a) connect to another device, b) provide metadata about itself, c) be controlled, and d) provide references to other devices. In addition, it defines a messaging protocol over TCP/IP that every Obje-enabled device must implement. The main difference between EMI^{2}lets and Obje is that whereas in the former is the developer of a smart object the one who decides what the user interface presented to the end user will look like and what functionality it will have access to, in the Obje case the responsibility for

determining appropriate interactions shifts from the developer to the end user. In other words, the programming on each device in Obje only tells the device how to interact with peers using the abstract mechanisms previously mentioned. The authors of Obje argue such semantic ignorance is necessary for open-ended interoperability. However, this flexible approach implies that they will need to provide tools that let end-users compose and configure devices within a space. Our approach in EMI^{2}lets is much simpler and almost as flexible. The smart object developer decides the best and richest multiplatform (PC, PDA and mobile phone) user interface to control an object. Through EMI^{2}lets the end-user can directly operate with its surrounding objects. As a second drawback, Obje only runs on the PC platform and provides the capability for the end user to integrate different components within a smart space but does not make the smart objects embedded in AmI spaces readily available for the end-user to control as EMI^{2}lets does.

Other authors [Scott, 05] 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).

8 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 AmI using conventional readily-available hardware and software tools.
- EMI^{2}lets 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 EMI^{2}let Players and Servers, following the WebProfiles model described in [Lassila, 03]. In addition, we want to enable the cooperation of EMI^{2}Objects, 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

[Beigl, 01] M. Beigl, H.W. Gellersen, A. Schmidt, MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects., Computer Networks, Special Issue on Pervasive Computing, Vol. 25, No. 4, March 2001 401–409.

[Bluetooth, 05] Bluetooth Specification version 1.1, December 2005, http://www.bluetooth.com

[Czerwinski, 99] S. Czerwinski, B. Zhao et al., An architecture for a Secure Service Discovery Service. Proceedings of MobiCom'99, 1999.

[Edwards, 05] W.K. Edwards, M. W. Newman, J.Z. Sedivy, T.F Smith, Bringing Network Effects to Pervasive Spaces. IEEE Pervasive Computing – Mobile and Ubiquitous Systems, Vol. 4, No. 1, July-September 2005 15-17.

[Grace, 05] P. Grace, G.S. Blair and S. Samuel, A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments. Mobile Computing and Communications Review, ACM SIGMOBILE, Vol. 9, No. 1, January 2005 2-14.

[Lassila, 03] O. Lassila and M. Adler, Semantic Gadgets: Device and Information Interoperability in Kalle Lyytinen & Yongjin Yoo (eds.): "Ubiquitous Computing Environment", Case Western Reserve University, October 2003.

[López de Ipiña, 02] D. López de Ipiña, P. Mendonça and A. Hopper, TRIP: a Low-cost Vision-based Location System for Ubiquitous Computing, in Personal and Ubiquitous Computing, Vol. 6, No. 3, May 2002 206-219.

[López de Ipiña, 05] D. López de Ipiña, J.I. Vázquez and D. Sainz, Interacting with our Environment through Sentient Mobile Phones, Proceedings of 2nd International Workshop in Ubiquitous Computing (IWUC-2005), ICEIS 2005, ISBN 972-8865-24-4, May 2005 19-28.

[Microsoft, 05] Microsoft Corporation, Mobile Developer Center, http://msdn.microsoft.com/mobility/, December 2005.

[Myers, 01] B.A. Myers, Using Hand-Held Devices and PCs Together. Communications of the ACM, Vol. 44, No. 11, November 2001 34 – 41.

[Rohs, 05] M. Rohs, P. Zweifel, A Conceptual Framework for Camera Phone-based Interaction Techniques, Pervasive Computing: Third International Conference, PERVASIVE 2005, Lecture Notes in Computer Science (LNCS) No. 3468, Springer-Verlag, Munich, Germany, 2005.

[Scott, 05] D. Scott et al., Using Visual Tags to Bypass Bluetooth Device Discovery, ACM Mobile Computing and Communications Review, Vol.9, No.1, January 2005 41-52.

[Shadbolt , 03] N. Shadbolt, Ambient Intelligence, IEEE Intelligent Systems, Vol. 2, No.3, July/August 2003.

[Siegemund, 04] F. Siegemund, and T. Krauer, Integrating Handhelds into Environments of Cooperating Smart Everyday Objects, Proceedings of the 2nd European Symposium on Ambient Intelligence. Eindhoven, The Netherlands, November 2004.

[Sun, 05a] Sun Microsystems, Inc, Java 2 Platform, Micro Edition (J2ME), http://java.sun.com/j2me/, December 2005.

[Sun, 05b] Sun Microsystems, Inc., Jini Specifications Archive - v2.1, http://java.sun.com/products/jini/2_1index.html, December 2005.

[Symbian, 05] Symbian Ltd., Symbian OS – the mobile operating System, December 2005, http://www.symbian.com/

[UPnP, 05] The Universal Plug and Play Forum, December 2005, http://www.upnp.org/

[Vazquez, 04] J.I. Vázquez, D. López de Ipiña, An Interaction Model for Passively Influencing the Environment, Adjunct Proceedings of the 2nd European Symposium on Ambient Intelligence, Eindhoven, The Netherlands, November 2004.

[Vazquez, 05] J.I. Vázquez, and D. López de Ipiña, An HTTP-based Context Negotiation Model for Realizing the User-Aware Web, 1st International Workshop on Innovations In Web Infrastructure (IWI 2005), Chiba, Japan, May 2005.

[Zhu 05] F. Zhu, M.W. Mutka, L.M. Ni., Service Discovery in Pervasive Computing Environments. IEEE Pervasive Computing, Vol. 4, No. 4, October 2005 81-90.