A PLATFORM TO BUILD SMART SPACES CONTROLLABLE FROM MOBILE DEVICES

D López-de-Ipiña, J I Vazquez, D García, J Fernández, I García, D Sainz and A Almeida
University of Deusto, SPAIN
{dipina, ivazquez}@eside.deusto.es, {dsainz, aalmeida}@tecnologico.deusto.es

The current multimedia, processing and communication capabilities of mobile devices make them most suitable to act as our intermediaries with the surrounding environment. They are capable of sensing, processing, storing and communicating with the artefacts augmented with computing services (i.e. smart objects) deployed in a smart space. This paper describes a device-type, user-location and communication-means agnostic platform, namely EMI^{2}lets, which fulfils a two-fold purpose: a) it transforms our mobile devices into universal remote controllers of smart objects and b) it helps us constructing smart object ecosystems, i.e. smart spaces.

1. INTRODUCCTION

Current PDAs and mobile phones are equipped with continuously increasing processing and storage capabilities, better and more varied communications mechanisms (Bluetooth, Wi-Fi, GPRS/UMTS) and increasingly capable multimedia capture and playback facilities. Moreover, they are far more easily programmable, Symbian (1), Microsoft (2), Sun (3), Brew (4) or Opera (5), 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 et al. (6), Rohs and Zweifel (7) – since they can sense (discover) what smart objects are in their whereabouts. A smart object, Beigl et al. (8), is an everyday artefact, physical (e.g. door, classroom) or virtual (e.g. a specific location within a house) augmented with some accessible computational service. Once a mobile device discovers a nearby smart object, it can control it.

Taken into account all the above considerations, it is logical to think that mobile devices will play a key role in Ambient Intelligence (AmI), Shabolt (9). AmI envisions sentient computing-service enriched environments which explicitly or implicitly react to the user's current context, in order to enhance his daily activities. In consequence, 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, preferences and current context.

In this paper, we describe the design and implementation of EMI^{2}lets (Environment to Mobile Intelligent Interaction), López-de-Ipiña et al. (10), a middleware platform to facilitate the development and deployment of AmI-aware environments, such as a home or an office.

The structure of the paper is as follows. Section 2 describes the requirements to be fulfilled by a smart space enabling middleware. Section 3 details the EMI^{2}lets platform, which simplifies both the creation of software representatives for everyday objects and their controlling proxies deployable in mobile devices. Section 4 illustrates the smart object development and deployment process allowed by the EMI^{2}lets platform and its plug-in based extensibility features. Section 5 gives some example applications developed with the EMI^{2}lets platform. Section 6 shows some performance results. Section 7 overviews some related work. Finally, section 8 offers some conclusions and suggests further work.

2. SMART SPACE-ENABLING MIDDLEWARE

In order to make the AmI vision reality significant progress must still be achieved in diverse areas such as context-awareness (biometry, indoor location systems), ubiquitous computing and communication, intelligent interfaces, artificial intelligence (reasoning and learning) and so forth. Despite all this, given the current technology state of the art, the first AmI environments or smart spaces, can start being deployed if suitable AmI-specific middleware and tools are defined.

For us, a smart space is a physical location where the artefacts (physical and virtual objects) present within are augmented with computing services, i.e. they are populated with smart objects ecologies. Two examples of such smart objects would be: a) a mobile phone locally accessible (Bluetooth) ticket booking service in a cinema, preventing the user from long queuing to purchase tickets; or b) a virtual post-it service assigned to the fridge door where a user may see the notes left by other family members in his mobile phone when he passes by.

2.1. Active and Passive Mechanisms

An AmI agent (user or device) can influence the environment, and thus, its constituent smart objects' ecology state, via active (explicit interaction) or passive (implicit interaction) methods.

Active methods are those in which the agent explicitly commands the smart objects 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 or smart object 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 or smart objects to do anything specific, 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 be disseminated 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 and context negotiation issues, Vazquez and López-de-Ipiña (10).

2.2. Requirements for Active Influence over Smart Objects

The two minimum requirements a platform enabling active influence over smart objects must address are: a) a mechanism to discover through ad-hoc or wireless networking the computing services exported by surrounding smart objects, and b) 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, according to Grace et al. (12) and Zhu et al. (13). Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini, Sun (14). 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 (15)). Finally, interaction independent discovery protocols such as SLP, Czerwinski et al. (16).

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, the design and implementation of an AmI-enabling platform is described which addresses the service discovery and interaction aspects required for active influence (explicit invocation) over smart object ecologies.

3. THE EMI²LETS PLATFORM

EMI²lets is 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, or even different implementation platforms. EMI²lets addresses the intelligent discovery and interaction among mobile devices and smart objects. 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. 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 smart objects to EMI²Proxies, 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 EMI²lets. This platform provides the infrastructure and software tools required to ease the development and deployment of the smart objects that populate smart environments

.

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 concept

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 smart objects. A smart object may be equipped with enough hardware resources to host an EMI²let Server, or alternatively a group of EMI²lets associated to different smart objects may all be hosted within an autonomous version of an EMI²let Server.

The EMI²let Server acts as a repository of smart object representatives. It publishes the services offered by the hosted EMI²lets by means of the communication mechanism supported at the server (e.g. Bluetooth, UPnP), 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 smart object 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.

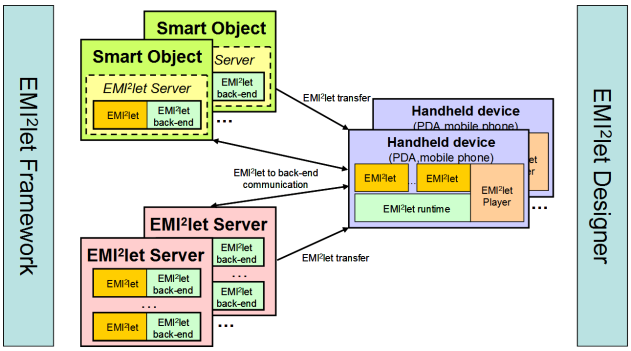

Figure 2: EMI²lets possible configuration

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 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 smart object representatives (EMI²lets) 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.

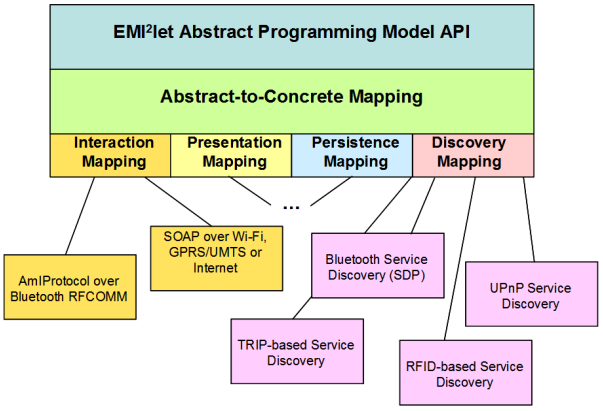

Figure 3: EMI²lets internal architecture

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. In the case that a communication link fails, the EMI²lets runtime will use other available communication links in a transparent way to the user.

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 Pocket PC and Web platforms

3.3. Importance of Reflection

The use of reflection is very important 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 (3) `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 smart object it represents.
- `NotifyDisconnected`, offers an EMI²let the possibility of being aware when the controlled smart object 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.

4. BUILDING AND DEPLOYING EMI²LETS

The main goal of the EMI²lets framework is to simplify the development and deployment of smart objects which can later be discovered and consumed by client devices in a communication and discovery protocol agnostic manner.

The development of smart objects is eased by: a) providing a framework, i.e. an API with a set of rules that every developer must follow, and b) an IDE which simplifies the interface design of an EMI²let for its three supported platforms: mobile phone, PDA and PC.

The deployment of smart objects is simplified by the EMI²let Server component, in charge of hosting and publishing EMI²lets.

The EMI²lets platform supports several discovery protocols and communication mechanisms in the form of plug-ins. Each discovery plug-in implements the discovery interface, whilst the interaction plug-ins do the same with the interaction interface. Currently, we can discover smart objects by means of UPnP, Bluetooth SDP, or looking up a simple web service registry. Invocations can be transmitted to smart objects following the EMI²let protocol by means of sockets through Wi-Fi and GPRS/UMTS and Bluetooth RFCOMM.

4.1. An EMI²let Life Cycle

Figure 5 illustrates the life cycle of an EMI²let from its development to its deployment:

1. .NET code following the EMI²let framework's API is developed on a PC through the EMI²let Designer. This tool offers a drag & drop based interface designer (see Figure 6) and simplifies the overall EMI²let coding and compilation process. Once the EMI²let implementation is completed it is uploaded into an EMI²let Server.
2. The EMI²let Server publishes the available EMI²lets deployed within it. This server is in charge of publishing the registered EMI²lets using all the available communication mechanisms and discovery protocols available at the hosting machine.
3. An EMI²let Player, running on either a PC, a PDA, a mobile phone or a web browser, discovers the services available in nearby EMI²let Servers by means of the available communication mechanisms on the device.
4. An EMI²let Player downloads from an EMI²let Server a user selected EMI²let which is then verified, installed and executed on the player. After

iterating with the EMI²let, the user may stop it and choose whether to wipe it out from the device or save it in the player EMI²let repository for later use.

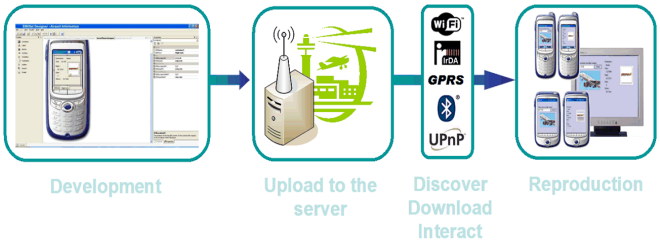

Figure 5: EMI²let lifecycle.

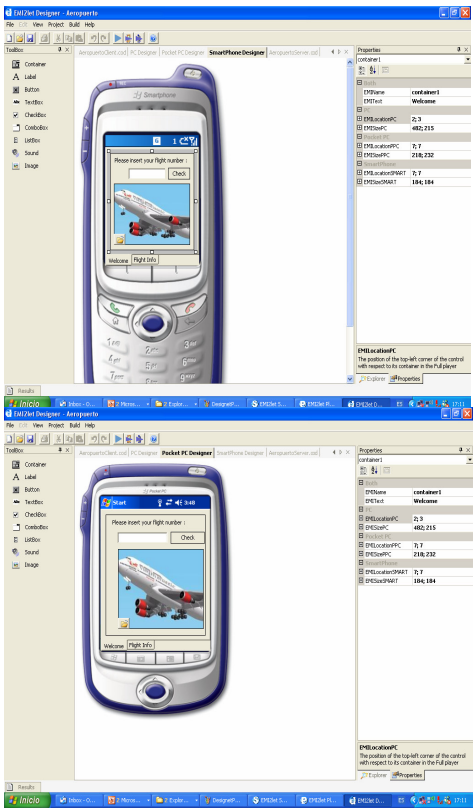

Figure 6: EMI²let Designer.

4.2. An EMI²let Discovery Plug-in

In order to prove the extensibility features of the EMI²lets platform, an interesting example of an EMI²lets plug-in is described. This plug-in accommodates to the discovery abstraction of EMI²lets and it is based on the TRIP, López-de-Ipiña (17), 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: a) device discovery and b) 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 (18).

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 et al. (19).

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 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 mobile phone cameras (6).

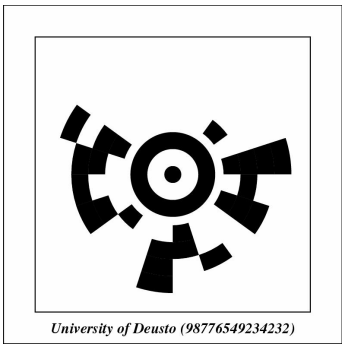

Figure 7: A tag encoding 66 bits of data.

We have applied TRIP tags to encode the Bluetooth address of an EMI²let Server and an identifier to select a smart object representative in the server. Likewise, we have also used those tags to encode tiny urls (see http://tinyurl.com) which point to an EMI²let 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, including the virtual ones, in his surroundings.

5. EMI²LETS EXAMPLES

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 deaf 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 a 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 a smart object both locally, while in the environment, or remotely, far away from the environment.

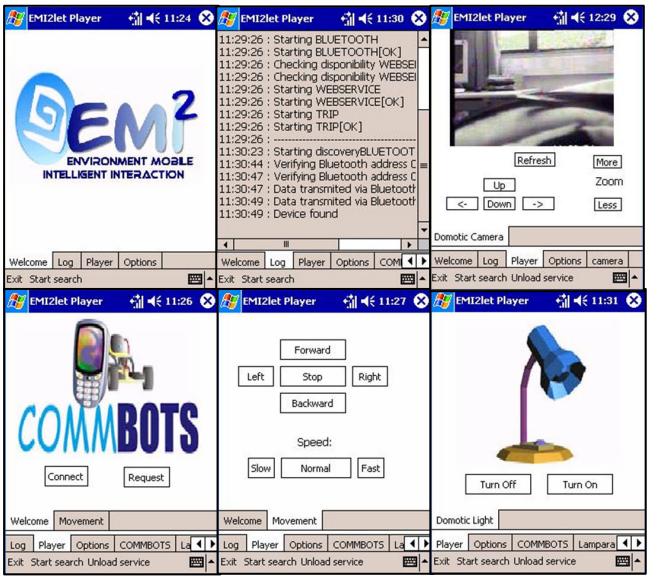

Figure 8: EMI²lets running on a PDA.

Figure 9: EMI²lets running on a mobile phone.

Figure 8 and Figure 9 show three of the previously described EMI²lets 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.

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.

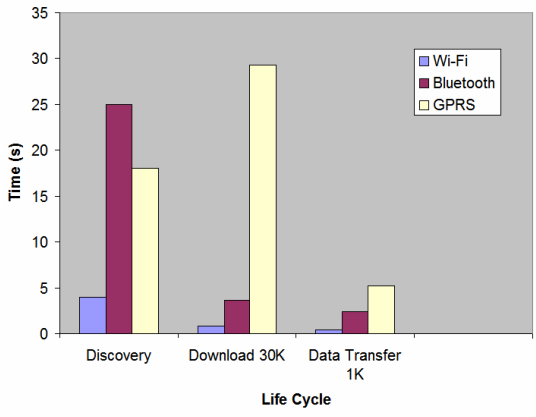

Figure 10:EMI²lets communication costs.

Figure 11 shows the effective data transfer rates obtained over the three wireless communication mechanisms we have used in EMI²lets. Obviously, the data transfer rate obtained through Wi-Fi is the best, whereas Bluetooth offers the second best behaviour.

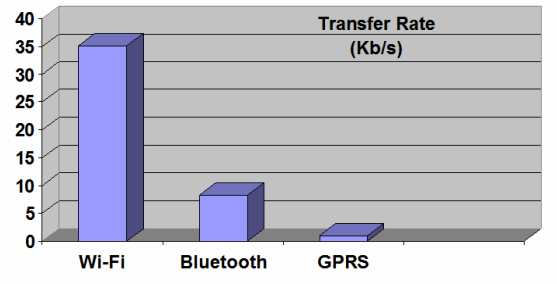

Figure 11: Effective data transfer rate in EMI²lets.

7. RELATED WORK

The EMI²lets platform presents some resemblance to the Smoblets software framework proposed by Siegemund and Krauer (20). 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 (12). 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.

The Pebbles project, Myers (21), 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 specify 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 et al. (22), 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 (19) 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 the EMI^{2}lets middleware platform which simplifies the development and deployment of smart objects and their discovery and control from mobile devices. EMI^{2}lets is aimed at enhancing our mobile devices and everyday objects in the environment with universal active influence capabilities and computer accessible computing services, respectively. The main features of this platform are:

- 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.
- Is 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” software development philosophy.

We are currently working in an implementation of EMI^{2}lets for the Java platform based on the OSGi standard (OSGi Alliance (23)).

In future work we want to add more sophisticated service discovery and context negotiation features between EMI^{2}let Players and Servers. For example, we want to make a user representing EMI^{2}let Player to only “see” (discover) those services that adjust to his profile, preferences and current context, rather than all the available services at that area. Moreover, we would like to enhance the EMI^{2}let Players with smart object

orchestration capabilities, so that the services offered by several smart objects can be composed. Finally, we would like to introduce mechanisms to enable the cooperation of smart objects, for instance, through the incorporation of distribution shared tuple space or the adoption of semantic web technologies.

ACKNOWLEDGEMENTS

This work has been financed by a 2005-06 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. Symbian, 2006, "Symbian OS – the mobile operating System2, http://www.symbian.com/
2. Microsoft, 2006, "Mobile Developer Center", http://msdn.microsoft.com/mobility/
3. Sun, 2006, "Java 2 Platform, Micro Edition (J2ME)", http://java.sun.com/j2me/
4. Brew, 2006, "Qualcomm Brew Home", http://brew.qualcomm.com/brew
5. Opera, 2006, "Opera Platform™ Enabling AJAX on phones", http://brew.qualcomm.com/brew/
6. López-de-Ipiña D., Vázquez J.I. and Sainz D., 2005, "Interacting with our Environment through Sentient Mobile Phones", IWUC-2005, ICEIS 2005, ISBN 972-8865-24-4, 19-28
7. Rohs M., Zweifel P., 2005, "A Conceptual Framework for Camera Phone-based Interaction Techniques, Pervasive Computing", PERVASIVE 2005, LNCS no. 3468, Springer-Verlag
8. Beigl M., Gellersen H.W., Schmidt A., 2001, "MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects", Computer Networks, Special Issue on Pervasive Computing, vol. 25, no. 4, 401–409
9. Shadbolt N., 2003, "Ambient Intelligence", IEEE Intelligent Systems, vol. 2, no.3
10. López-de-Ipiña D., Vazquez J. I., et al., 2005, "A Reflective Middleware for Controlling Smart Objects from Mobile Devices"., Smart Objects & Ambient Intelligence, Grenoble, France
11. Vazquez J.I., López-de-Ipiña D., 2005, "An HTTP-based Context Negotiation Model for Realizing the User-Aware Web", IWI 2005, Chiba, Japan,
12. Grace P., Blair G.S., Samuel S., 2005, "A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments", ACM SIGMOBILE, vol. 9, no. 1
13. Zhu F., Mutka M.W., Ni. L.M., 2005, "Service Discovery in Pervasive Computing Environments". IEEE Pervasive Computing, vol. 4, no. 4, October, 81-90
14. Sun, 2006, Sun Microsystems, Inc., "Jini Specifications Archive", http://java.sun.com/products/jini/2_1index.html
15. UPnP, 2005, "The Universal Plug and Play Forum", http://www.upnp.org/
16. Czerwinski S., Zhao B. et al., 1999, "An architecture for a Secure Service Discovery Service", MobiCom'99
17. López-de-Ipiña, D., Mendonça P., Hopper A., 2002, "TRIP: a Low-cost Vision-based Location System for Ubiquitous Computing", Personal and Ubiquitous Computing, vol. 6, no. 3, 206-219
18. Bluetooth, 2005, "Bluetooth Specification version 1.1", http://www.bluetooth.com
19. Scott D., et al., 2005, "Using Visual Tags to Bypass Bluetooth Device Discovery", ACM Mobile Computing and Communications Review, vol.9, no.1, 41-52
20. Siegemund F., Krauer T., 2004, "Integrating Handhelds into Environments of Cooperating Smart Everyday Objects", EUSAI, The Netherlands
21. Myers, B.A., 2001, "Using Hand-Held Devices and PCs Together". Communications of the ACM, vol. 44, no. 11, 34 – 41
22. Edwards W.K., Newman M. W. et al., 2005, "Bringing Network Effects to Pervasive Spaces", IEEE Pervasive Computing, vol. 4, no. 1, 15-17
23. OSGi Alliance, 2006, "The OSGi Service Platform – Dynamic services for networked devices", http://www.osgi.org/