A Middleware for the Deployment of AmI Spaces

Diego López de Ipiña[1], Iñaki Vázquez[1], Daniel Garcia[1], Javier Fernández[1], Iván García[1], David Sáinz[1] and Aitor Almeida[1]

[1] University of Deusto, Faculty of Engineering, Avda. Universidades 28, 48007 Bilbao, Spain
`{dipina, ivazquez}@eside.deusto.es,`
`{dsainz, aalmeida}@tecnologico.deusto.es`

Abstract. The latest mobile devices are offering more multimedia features, better communication capabilities (Bluetooth, Wi-Fi, GPRS/UMTS) and are more easily programmable than ever before. So far, those devices have been used mainly for communication, entertainment, and as electronic assistants. On the other hand, Ambient Intelligence (AmI) is emerging as a new research discipline merging the fields of Ubiquitous Computing and Communications, Context Awareness and Intelligent User Interfaces. The ultimate goal of AmI is to surround our working and living environments with context-aware, cooperative and invisible devices that will assist and help us in our everyday activities. Current mobile devices, which accompany us anywhere and at anytime, are the most convenient tools to help us benefit from AmI-enhanced environments. In other words, mobile devices are the best candidates to intermediate between us and our surroundings. In consequence, this paper proposes a middleware which aims to make this vision reality following a two-fold objective: (1) to simplify the creation and deployment of physical spaces hosting smart objects and (2) to transform mobile devices into universal remote controllers of those objects.

1 Introduction

Ambient Intelligence (AmI) [13] defines an interaction model between us and a context-aware environment, which adapts its behaviour intelligently to our preferences and habits, so that our life is facilitated and enhanced.

Current PDAs and mobile phones are equipped with significant processing and storage capabilities, varied communications mechanisms (Bluetooth [1], Wi-Fi, GPRS/UMTS) and increasingly capable multimedia capture and playback facilities. Moreover, they are far more easily programmable (Compact.NET [9], J2ME [15] or Symbian [17]), i.e. extensible, than ever before.

Mobile devices equipped with Bluetooth, built-in cameras, GPS receivers, barcode or RFID readers can be considered as sentient devices [8][11], since they are aware of what smart objects are within an AmI space. We understand by Smart Space (or AmI-enhanced environment), a location, either indoors or outdoors, where the objects present within (smart objects) are augmented with computing services. A smart object is an everyday object (door, classroom, parking booth) or a device augmented with some accessible computational service [2]. Once a mobile device discovers a nearby smart object, it may operate over it.

We deem that mobile devices will play a key role within AmI, since they are always with us and 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 middleware to facilitate the development and deployment of mobile context-aware applications for AmI spaces. This software provides the software infrastructure to (1) convert physical environments into AmI spaces and (2) transform mobile devices into remote controllers of the smart objects in those spaces.

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 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 deployable in mobile devices. Section 4 illustrates the life cycle of an EMI^{2}let from its development to its deployment in a mobile device and lists some interesting applications produced with the EMI^{2}lets platform. Section 5 shows some performance results achieved by the current implementation of EMI^{2}lets. Section 6 overviews some related work. Finally, section 7 offers some conclusions and suggests further work.

2 EMI^2: an AmI Architecture

Regardless of the continuous progress achieved in all the related research topics which contribute to the AmI vision, we are still far away from its materialisation. A good starting point to solve this may be the definition of suitable software architectures and frameworks specially catered for AmI. 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 *AmI Space*. For instance, a cinema may be enhanced with a Bluetooth mobile phone accessible 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 the user to enter a PIN in her mobile to be given access.

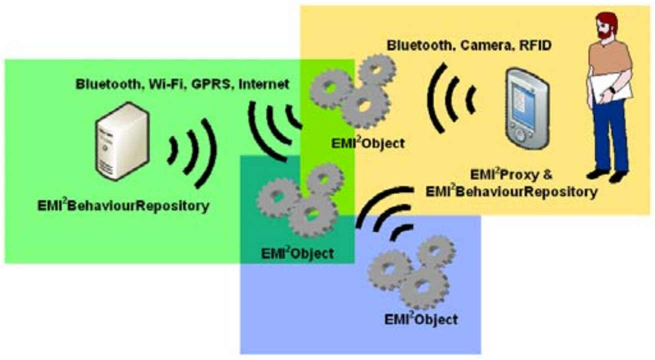


Fig. 1. The EMI^2 Architecture

Fig. 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.
- *EMI^{2}Object*: is an agent representing any device or physical object (vending machine, door, ticket box) within a smart environment augmented with computational services, i.e. the capacity to adapt its behaviour based on ambient conditions or user commands. An EMI^{2}Object cooperates 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.

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. Active methods can be implemented with any distributed computing technology capable of issuing commands, which 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 [19] 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.

2.2 Active Influence over EMI^{2}Objects

In this paper we want to 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.

The minimum features such a middleware has to provide 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 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 [5][21]. Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini [16]. 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) [18]. Finally, there are interaction independent discovery protocols such as the Service Location Protocol [3].

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 EMI^{2}Objects.

3 The EMI^{2}lets Platform

EMI^{2}lets is the result of mapping the EMI^2 architecture into a software development platform 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 to a nearby handheld device, which offers a graphical interface for the user to interact over that 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.

- *Interoperability*, the EMI^{2}lets 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 ease the development and deployment of AmI scenarios.

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

- Transform mobile devices (mobile phones and PDAs) into remote universal controllers of the smart objects located within an AmI space.
- Enable both local (Bluetooth, Wi-Fi) and global access (GPRS/UMTS) to the smart objects in an AmI space, 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). Allow this middleware to easily adapt to newly emerging discovery (e.g. RFID identification) and interactions means.
- Make use of commonly available hardware and software features in mobile devices, without demanding the creation of proprietary hardware, or software.
- 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 could be run in a mobile, a PDA or a PC.

3.1 The EMI^{2}lets Vision

Fig. 2 shows a possible deployment of an EMI^{2}let-aware environment. A group of 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 a 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 is normally equipped. 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.

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-aware devices for executing downloaded code.
4. An EMI^{2}let Player to discover, download, verify and control the execution life of a downloaded EMI^{2}let. A version of the player is available for each device type which may act as host of EMI^{2}lets, e.g. PDA, mobile phone or PC.
5. An EMI^{2}let Server which acts as repository of EMI^{2}lets and as running environment of EMI^{2}lets server-sides.

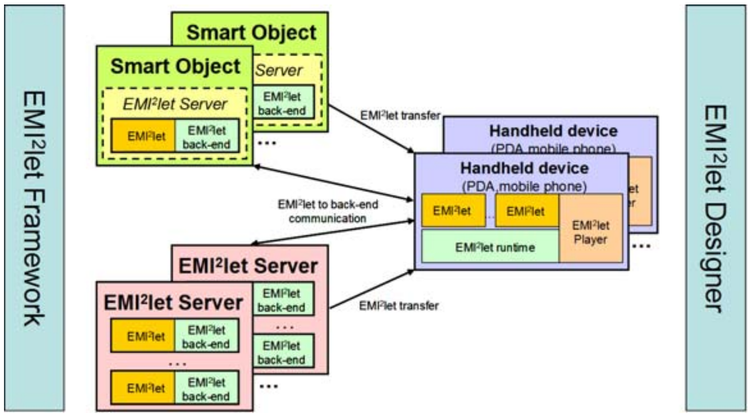


Fig. 2. The EMI^{2}lets in action.

In order to achieve the design objectives previously listed, we have created the layered software architecture shown in Fig. 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 a mobile sentient application:

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.
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.

The *EMI^{2}let Abstract-to-Concrete Mapping* layer translates the invocations over the generic interfaces to the appropriate available mechanisms both in the mobile device and the EMI^{2}Objects in the environment. The discovery, interaction, presentation and persistency abstractions encapsulate the concrete discovery, interaction, presentation or persistency models used. They implement an API for performing service discovery and interaction, graphical interface generation and data persistence 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 abstractions which are part of the EMI^{2}let runtime in the target device.

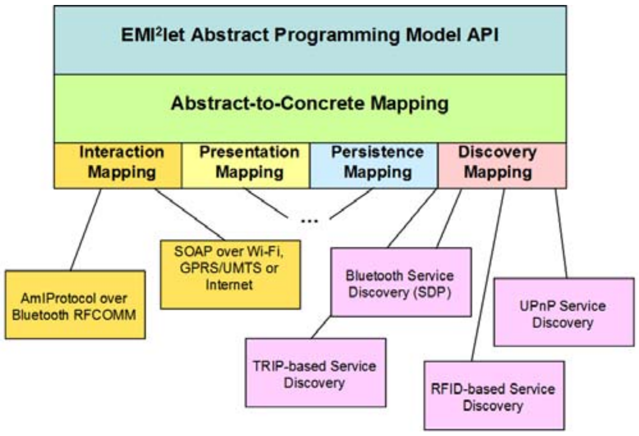

EMI²let Abstract Programming Model API
Abstract-to-Concrete Mapping
Interaction Mapping
Presentation Mapping
Persistence Mapping
Discovery Mapping
...
AmIProtocol over Bluetooth RFCOMM
SOAP over Wi-Fi, GPRS/UMTS or Internet
Bluetooth Service Discovery (SDP)
UPnP Service Discovery
TRIP-based Service Discovery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

Fig. 3. EMI^{2}lets Internal Architecture

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 mapping (or group of mappings) 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 in its surroundings.

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. Currently we support the following controls: panel, label, button, textbox, checkbox, combobox, listbox, sound and image. Some examples of the graphical control classes defined are: `EMI2Panel`, `EMI2Button` or `EMI2TextBox`. This set of controls enables us to create graphical interfaces for EMI^{2}lets which are agnostic to 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. Still, in order to guarantee a proper layout of the graphical controls according to the three target device types (PC, PDA and mobile phone) supported, specific layout hints can be given, with the help of the EMI^{2}let Designer, for each device type. An example showing this fact can be seen in Fig. 4. Lately, we have added support for a new target device type, namely web-enabled devices. We have enhanced the functionality of the EMI^{2}let Server so that it can export EMI^{2}lets as web pages accessible from non EMI^{2}let-compliant web-enabled devices.

The modus operandi of the plug-ins associated to any of the four available functional mapping 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. We plan to establish a more sophisticated and flexible plug-in configuration model in due time.

Both the *Abstract-to-Concrete Mappings* and the *Functional Mapping* (plug-ins) layers compose the runtime installed in each target device. The code of the downloaded EMI^{2}let is linked on arrival by the EMI^{2}let Player with the runtime.

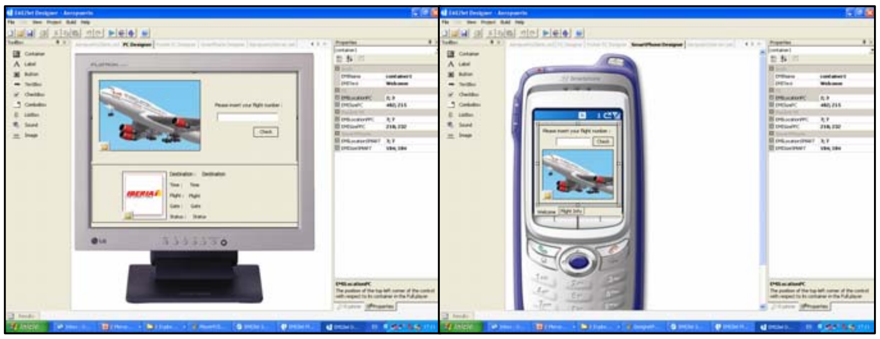


Fig. 4. EMI^{2}lets Designer

3.3 The EMI^{2}Protocol

The EMI^{2}Protocol defines a set of basic commands or messages which are exchanged among EMI^{2}let Players and Servers and EMI^{2}let client- and server-sides. For those messages to be exchanged, a connection between the client and server peers must have previously been established. The discovery plug-ins in a Player are in charge of discovering surrounding Servers and opening connections between Players and Servers.

The plug-in implementations which use the Bluetooth bearer actually exchange the commands as specified below, whereas plug-ins using other bearers such as Wi-Fi or GPRS invoke a generic web service with methods corresponding to those commands. The most important commands offered are:

- `HELLO`, through this message an EMI^{2}let Player provides metadata to an EMI^{2}let Server. The metadata provided is the type of device the Player is running on and the set of communication and discovery mechanisms (i.e. plug-ins installed and running) available at the device.

- `HELLO_RESPONSE`, the EMI^{2}let Server informs the connected EMI^{2}let Player about the communication channels it supports (i.e. the server's own plug-ins installed and running).
- `SERVICE_QUERY`, message issued by an EMI^{2}let Player to an EMI^{2}let Server in order to retrieve information about the EMI^{2}lets it hosts.
- `SERVICE_QUERY_RESPONSE`, message issued by an EMI^{2}let Server to provide a requesting EMI^{2}let Player with the metadata of the services it hosts.
- `DOWNLOAD_SERVICE`, message issued by an EMI^{2}let Player to an EMI^{2}let Server in order to retrieve the code of a selected EMI^{2}let.
- `DOWNLOAD_SERVICE_RESPONSE`, message issued by an EMI^{2}let Server to provide a requesting EMI^{2}let Player with the code of a selected service.
- `COMMAND_SEND`, message encapsulating a packet of data sent between an EMI^{2}let executing in a Player and its server-side hosted on an EMI^{2}let Server.
- `COMMAND_RESPONSE`, message encapsulating a packet of data sent between an EMI^{2}let server-side and the EMI^{2}let running in the Player.

3.4 Implementation

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 can be trusted. Every EMI^{2}let downloaded is encrypted with a private key only shared by the EMI^{2}let designer and the player. After downloading an EMI^{2}let, the Player unencrypts its code and verifies that the class downloaded follows the EMI^{2}let framework rules.

After verification, the player can start the EMI^{2}let by invoking the methods defined in the `EMI2let` base class, from which every EMI^{2}let must inherit. The methods defined by this class closely resemble the ones provided by a J2ME 9 `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}let-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 of the EMI^{2}Object it represents.
- `NotifyDisconnected`, offers an EMI^{2}let the possibility of being aware of when the EMI^{2}Object that it controls cannot be accessed any longer.
- `GetAddresses`, enables the EMI^{2}let-hosting player to retrieve the addresses at which the EMI^{2}let server-side is available. For instance, an EMI2let server-side may be accessible both through a Bluetooth address or a url pointing to a web service.

Our first reference implementation has used Microsoft .NET, a platform that 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, namely PC, PDA and mobile phone. As a least common multiple for the definition of the presentation controls of an EMI^{2}let, we have chosen most of the Compact.NET framework graphical controls, which represent a superset of the ones in the SmartPhone framework and a subset of the standard .NET desktop-oriented ones.

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 assembly code downloaded is linked dynamically (late bound) with the runtime installed in the target device. This assembly process would not have been possible without the use of reflection.

3.5 An EMI^{2}let Plug-in Example

The plug-in based mechanism adopted in EMI^{2}lets guarantees its extensibility. If we want to add support to EMI^{2}lets for any newly emerging discovery or communication technology we simply need to implement a plug-in for the corresponding abstraction. In order to prove this point we have implemented a service discovery plug-in based on the TRIP [7] tag-based visual identification system.

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 [12].

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 Fig. 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 $\approx 2^{66}$).

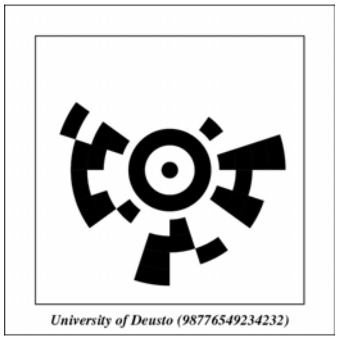


Fig. 5. A tag encoding 66 bits of data.

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 [8]. 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, achieving 2 fps in a TSM 500 Pocket PC.

3.6 EMI2lets State Management

The EMI^{2}lets platform incorporates a simple state management mechanism. In order to prevent the user from continuously entering the same input details in the execution of a previously run (durable) EMI^{2}let, the player stores in EMI^2 Cookie objects the last values input. The EMI^2 Cookies, contrary to the well-known HTTP cookies, keep the state in the player (client-side). The UDDI associated to an EMI2let is employed to establish associations between EMI2lets and EMI2Cookies. Currently, we are working on extending state management in EMI2lets by adopting the WebProfiles model proposed at [20].

4 EMI^{2}lets Applications

In this section we will first describe the lifecycle of an EMI^{2}let from its development to its deployment and secondly we will mention some of the applications developed with EMI^{2}lets.

4.1 The Life Cycle of an EMI^{2}let

Fig. 6 shows the life cycle of an EMI^{2}let from its development with the EMI^{2}let Designer (see Fig. 4) until its deployment at the target mobile device and EMI^{2}let server. In our approach, active .NET code developed on a PC with the help of the EMI^{2}let Designer is uploaded into an EMI^{2}let Server, from where it is later discovered, downloaded, verified and executed in the context of an EMI^{2}let Player. After its execution and depending on its durability properties, the EMI^{2}let is cached or removed from the Player.

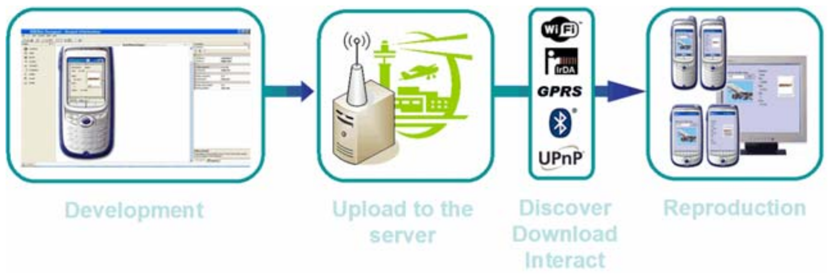


Fig. 6. EMI^{2}let Life Cycle.

4.2 Examples of EMI^{2}lets

We have developed EMI^{2}lets targeted to the following application domains: accessibility, home/office automation, industry and public spaces.

In the domain of accessibility we have developed EMI^{2}lets 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 EMI^{2}lets platform.

In the home and office automation domain we have created EMI^{2}lets that enable us to control from our mobile devices the lights, music system (in fact the Windows Media Player in a PC) and a Pan/Tilt/Zoom security camera at a home or office.

As far as the industry domain is concerned we have developed an EMI^{2}let 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 EMI^{2}let uses the Bluetooth communication channel. When we are far away from the location of the robot, the EMI^{2}let uses the GPRS channel to communicate with the robot. The communication channel choice is undertaken by the EMI^{2}lets runtime autonomously.

Finally, on what we call the “public space” domain, we have created EMI^{2}lets 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 EMI^{2}lets show how a physical object in an outdoors space can be augmented with AmI features. For example, the Parking EMI^{2}let 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 while in the parking, 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 one of the biggest virtues of EMI^{2}lets, its capability to enact an action over an EMI^{2}Object both locally, while in the environment, or remotely, far away from the environment.

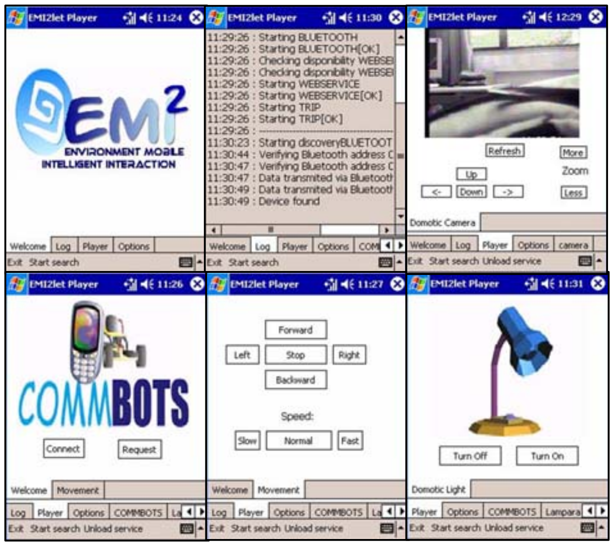
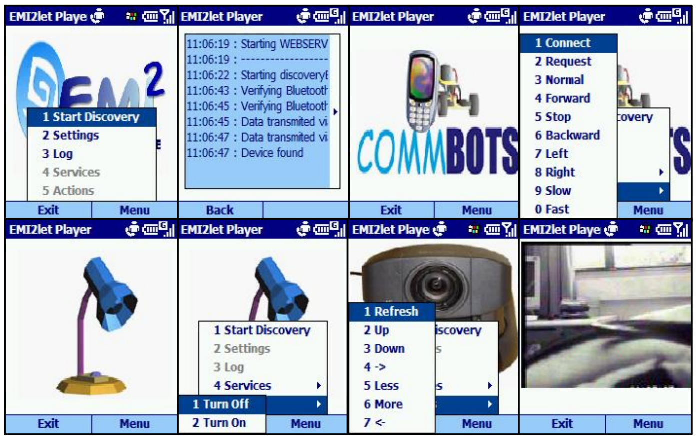

```
11:29:26 : Starting BLUETOOTH
11:29:26 : Starting BLUETOOTH[OK]
11:29:26 : Checking disponibility WEBSE
11:29:26 : Checking disponibility WEBSE
11:29:26 : Starting WEBSERVICE
11:29:26 : Starting WEBSERVICE[OK]
11:29:26 : Starting TRIP
11:29:26 : Starting TRIP[OK]
11:29:26 : ------------------------------
11:30:23 : Starting discoveryBLUETOOT
11:30:44 : Verifying Bluetooth address C
11:30:47 : Verifying Bluetooth address C
11:30:47 : Data transmited via Bluetooth
11:30:49 : Data transmited via Bluetooth
11:30:49 : Device found
```

Fig. 7. EMI^{2}lets running on a PDA.

Fig. 7 and Fig. 8 show three of the previously described EMI^{2}lets in action running in a PDA and a mobile phone, respectively. The EMI^{2}lets shown allow a user to control from his mobile device a robot, a lamp or a PTZ security camera. Something remarkable about the EMI^{2}lets platform is that in the development of those EMI^{2}lets 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.



```
11:06:19 : Starting WEBSERV
11:06:19 : -------------------
11:06:22 : Starting discoveryE
11:06:43 : Verifying Bluetooth
11:06:45 : Verifying Bluetooth
11:06:45 : Data transmited vi
11:06:47 : Data transmited vi
11:06:47 : Device found
```

Fig. 8. EMI^{2}lets running on a mobile phone.

5 EMI^{2}lets Performance Results

In order to asses the performance of our current implementation of the EMI^{2}lets platform we have carried out two tests:

1. A comparative measurement illustrating the different latencies experienced during an EMI^{2}let 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 EMI^{2}let and its server-side.

Fig. 9 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 EMI^{2}let Server to enquire about its installed EMI^{2}lets over GPRS or undertaking Bluetooth discovery. However, once the Bluetooth discovery has concluded the download of an EMI^{2}let code and the exchange of information between an EMI^{2}let and its server-side is much better than through GPRS and only worse to Wi-Fi which has a much better transfer rate.

Fig. 10 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.

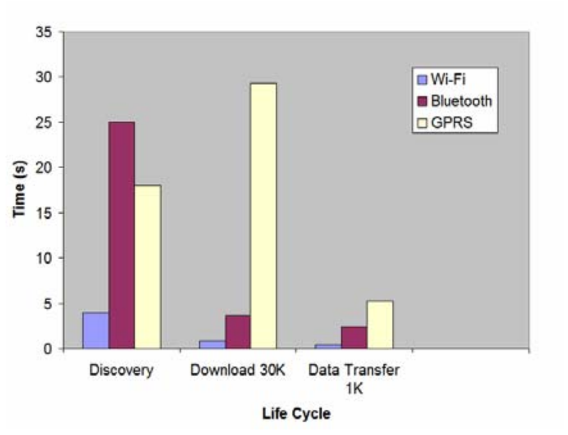


Fig. 9. EMI^{2}lets communication costs.

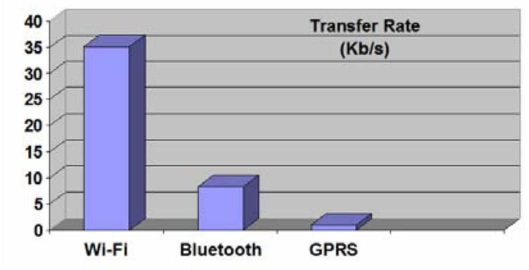


Fig. 10. Effective data transfer rate in EMI^{2}lets.

6 Related Work

The EMI^{2}lets platform presents some resemblance to the Smoblets software framework proposed by [14]. Both frameworks download into a mobile device the software representatives of objects located in a smart space. However, Smoblets only operate when they are within range of the smart object they represent. On the contrary, 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 [5]. 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, the EMI^{2}let code generated is independent of the target platform where it will be run (PC, PDA or mobile).

The Pebbles project [10] 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 [4] 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.

7 Conclusion and Future 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 [20]. 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 [6].

Acknowledgements

This work has been financed by a SAIOTEK 2004-05 grant from the Basque Government and the Cátedra de Telefónica Móviles at the University of Deusto. This work was granted the Image Cup 2005 prize by Microsoft Spain.

References

1. Bluetooth Specification version 1.1, http://www.bluetooth.com, (2005)
2. Beigl, M., Gellersen H.W., and Schmidt, A.: MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects. Computer Networks, Special Issue on Pervasive Computing, Vol. 25, No. 4, (2004) 401–409.
3. Czerwinski S., Zhao B. et al.: An architecture for a Secure Service Discovery Service. Proceedings of MobiCom’99, (1999)
4. Edwards W.K., Newman M. W., Sedivy J.Z. and Smith T.F: Bringing Network Effects to Pervasive Spaces. IEEE Pervasive Computing – Mobile and Ubiquitous Systems, Vol. 4, No. 1, (2005) 15-17

5. Grace P., Blair G. S. and Samuel S.: A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments. Mobile Computing and Communications Review, ACM SIGMOBILE, Vol. 9, No. 1, (2005) 2-14
6. Lassila O. and Adler M.: Semantic Gadgets: Device and Information Interoperability in Kalle Lyytinen & Yongjin Yoo (eds.): "Ubiquitous Computing Environment", Case Western Reserve University, (2003)
7. López de Ipiña, D., Mendonça P. and Hopper A.: TRIP: a Low-cost Vision-based Location System for Ubiquitous Computing, in Personal and Ubiquitous Computing, Vol. 6, No. 3, (2002) 206-219
8. López de Ipiña D., Vázquez I. and Sainz D.: 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, (2005) 19-28
9. Microsoft Corporation.: Mobile Developer Center, http://msdn.microsoft.com/mobility/, (2005)
10. Myers B.A.: Using Hand-Held Devices and PCs Together. Communications of the ACM, Vol. 44, No. 11, (2001) 34 - 41.
11. Rohs M., Zweifel P.: 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)
12. Scott D. et al.: Using Visual Tags to Bypass Bluetooth Device Discovery. ACM Mobile Computing and Communications Review, Vol.9, No.1, (2005) 41-52.
13. Shadbolt N.: Ambient Intelligence. IEEE Intelligent Systems, Vol. 2, No.3, (2003)
14. Siegemund, F. and Krauer T.: Integrating Handhelds into Environments of Cooperating Smart Everyday Objects. Proceedings of the 2nd European Symposium on Ambient Intelligence. Eindhoven, The Netherlands, (2004)
15. Sun Microsystems, Inc.: Java 2 Platform, Micro Edition (J2ME), http://java.sun.com/j2me/ (2005)
16. Sun Microsystems, Inc.: Jini Specifications Archive - v2.1, http://java.sun.com/products/jini/2_1index.html, (2005)
17. Symbian Ltd.: Symbian OS – the mobile operating System, http://www.symbian.com/, (2005)
18. The Universal Plug and Play Forum: http://www.upnp.org/, (2005)
19. Vázquez, J.I., López de Ipiña, D.: An Interaction Model for Passively Influencing the Environment. Adjunct Proceedings of the 2nd European Symposium on Ambient Intelligence, Eindhoven, The Netherlands, (2004)
20. Vázquez, J.I. and López de Ipiña D.: An HTTP-based Context Negotiation Model for Realizing the User-Aware Web. 1st International Workshop on Innovations In Web Infrastructure (IWI 2005), Chiba, Japan (2005)
21. Zhu F., Mutka M.W., L.M. Ni.: Service Discovery in Pervasive Computing Environments. IEEE Pervasive Computing, Vol. 4, No. 4, (2005) 81-90