Abstract

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

1. Introduction

Current PDAs and mobile phones are equipped with continuously increasing processing and storage capabilities, better and more varied communications mechanisms (Bluetooth [1], Wi-Fi, GPRS/UMTS) and increasingly capable multimedia facilities. Moreover, they are far more easily extensible (Compact.NET [2], J2ME [3] or Symbian [4]) than ever before.

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

Bearing in mind the technical progress and sentient features of last generation mobile devices, it is natural to think that they will play a starring role in the context of Ambient Intelligence (AmI) [6]. An obvious application will be their use as facilitators or intermediaries between us and a smart environment. In other words, mobile devices can behave as our personal electronic butlers, facilitating and enhancing our daily activities, and even acting on our behalf based on our profiles or preferences.

In this paper, we describe the design and implementation of EMI²lets (Environment to Mobile Intelligent Interaction applets), a software framework to facilitate the development and deployment of mobile context-aware applications for AmI environments.

The structure of the paper is as follows. Section 2 describes EMI², a software architecture modelling AmI. Section 3 introduces the EMI²lets platform, a partial materialisation of the EMI² architecture, which simplifies both the creation of software representatives for everyday objects and their controlling proxies deployed in mobile devices. Section 4 proposes a novel mechanism to discover smart objects based on visual tags used in EMI²lets. Section 5 lists some interesting applications produced with the EMI²lets platform. Section 6 mentions some related work. Finally, section 7 offers some conclusions and suggests further work.

2. EMI²: an AmI architecture

Regardless of the continuous progress in the research topics which contribute to the AmI vision, namely Ubiquitous Computing [7], context-awareness [8] or intelligent user interfaces [9], we are still far away from its materialisation. However, the definition of suitable software architectures and frameworks specially catered for AmI may be a good starting point. The EMI² (Environment to Mobile Intelligent Interaction) architecture is our proposed solution.

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

We understand by smart environment a location, either indoors or outdoors, where the objects present within (smart objects) are augmented with computing services. For instance, a cinema may be enhanced with a mobile phone locally accessible (Bluetooth) ticket booking service, so preventing the user from long queuing to purchase tickets. Similarly, the door of our office may be augmented with an access control service which demands a user passing by to enter a PIN in her mobile to be given access.

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

- **EMI²Proxy**: is an agent representing the user, which runs on the user’s mobile device (PDA or mobile phone). It acts on behalf of the user, adapting/controlling the environment for him, both explicitly, under the user’s
control, or implicitly, on its own judgement based on the profiles, preferences and previous interactions of the user with the environment.

- **EMI2Object**: is an agent representing any device or physical object (e.g. vending machine, door, ticket box) within a smart environment augmented with computational services, i.e. the capacity to adapt its behaviour in response to ambient conditions or user commands. An EMI2Object cooperates to achieve its goals with other EMI2 agents.

- **EMI2BehaviourRepository**: is an agent where knowledge and intelligence are combined to support sensible adaptation. EMI2Objects may require the assistance of an external EMI2BehaviourRepository to coordinate their own adaptation according to the user’s preferences, behaviour patterns or even the explicit commands received from an EMI2Proxy. The user’s mobile device can also be powered with an internal EMI2BehaviourRepository loaded with personal information and profiles in order to minimize the interaction with the owner, i.e. adopting implicit adaptation.

### 2.1. Active and passive mechanisms

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

Active methods are those in which the agent explicitly commands other agents to change their state or perform an action. For example, when a user enters a building, a sensor identifies him and commands the lift to be ready at the ground floor. When the user stands in front of his office door his mobile phone commands the electric lock to open. Active methods can be implemented with any well-known distributed computing technology capable of issuing commands. These commands will be transported in a local context by bearers such as Bluetooth or Wi-Fi and in a global context by GPRS/UMTS.

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

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

#### 2.2. The Need for an Active Influence Middleware

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

The current state of the art in discovery and interaction platforms falls into three categories [12]. Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini [13]. After discovery, the service (either a proxy or the full service) is downloaded onto the mobile device where it then operates. Secondly, solutions where the discovery protocols are integrated with specific interaction protocols, which are used to invoke the service after the service has been discovered. A good example of this is Universal Plug and Play (UPnP) [14]. Finally, there are interaction independent discovery protocols such as Service Location Protocol [15].

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

### 3. The EMI2lets platform

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

EMI2lets is a development platform for AmI which addresses the intelligent discovery and interaction among EMI2Objects and EMI2Proxies. EMI2lets follows a Jini-like mechanism by which once a service is discovered, a proxy of it (an EMI2let) is downloaded into the user’s device (EMI2Proxy). An EMI2let is a mobile component transferred from a smart object (EMI2Object) to a nearby handheld device, which usually offers a graphical interface for the user to interact with its associated 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. In other words, the EMI2let platform ensures that an EMI2Proxy is always using the communication means with best trade-off between performance and cost. For example, if Wi-Fi and Bluetooth are available, the former is chosen, however if GPRS/UMTS and Bluetooth are available, the latter is chosen.

- **Interoperability**, the EMI2lets, i.e. the software components downloaded from EMI2Objects to EMI2Proxies, are agnostic of the target device type, e.g. PC, a PDA or a mobile phone.

- **AmI** is the application domain that has driven the design of EMI2lets. This platform provides the infrastructure and software tools required to develop and deploy smart objects and their controlling proxies.

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

- Transform mobile devices (mobile phones and PDAs) into universal remote controllers of smart objects located in AmI environments.

- Enable both local (Bluetooth, Wi-Fi) and global access (GPRS/UMTS) to interact with the smart objects in AmI environments, seamlessly adapting to the most suitable underlying communication mechanisms.
• Develop middleware independent of a particular discovery or interaction mechanism. Abstract the programmer from the several available discovery (Bluetooth SDP or wireless UPnP discovery) and interaction mechanisms (RPC or publish/subscribe). Likewise, allow this middleware to easily adapt to newly emerging discovery (RFID) and interactions means.

• Utilise commonly available hardware and software features in mobile devices, without demanding the creation of proprietary hardware, or software protocols. Essentially, reuse current infrastructure and integrate it for its application to the AmI domain.

• Generate software representatives (proxies) of smart objects which can be run in any platform, following a “write once run in any device type” philosophy. The same EMI2let should run in a mobile, a PDA or a PC.

3.1. The EMI2lets vision

Figure 2: EMI2lets Platform.

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-compliant devices for executing the code downloaded.
4. An EMI2let Player to discover, download, verify and control the execution of a downloaded EMI2let. A version of the player is available for each device type which may act as a host of EMI2lets, e.g. mobile phone, PDA or PC.
5. An EMI2let Server which acts as a repository of EMI2lets and as a running environment of EMI2lets server-sides.

In order to achieve the EMI2lets 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 an EMI2let:
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, independently of the available communication mechanisms.
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.

3.2. Internal architecture

The EMI2lets internal architecture consists of the following elements:
1. The EMI2let Abstract Programming Model API
2. The EMI2let Abstract-to-Concrete Mapping
3. Service discovery and interaction
4. Service presentation
5. Data persistency

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

On deployment the code generated through these abstract interfaces is linked to the concrete implementations of the classes used which are part of the EMI2let runtime in the target device.
The architecture of the EMI² framework is very flexible and extensible because it is based on the concept of plug-in. A plug-in is simply an implementation of one of the available abstractions or functional mappings. In the process of associating a generic invocation to an actual one, the EMI²let Abstract-to-Concrete Mapping will be responsible of selecting the actual plug-in (or group of plug-ins) which best matches the invocation type. For example, if a downloaded EMI²let is installed on a device where both Bluetooth and GPRS communication are available, the abstract-to-concrete layer will have to choose one of those mechanisms to issue commands. Thus, if the mobile device is still within Bluetooth range of the EMI²let server-side, then it will translate the invocation into an EMI²Protocol message transported over Bluetooth RFCOMM. Otherwise, it will invoke via GPRS the generic web service (with methods corresponding to the EMI²Protocol commands) implemented by every EMI²let server-side. Similarly, if a mobile device is Bluetooth and Wi-Fi capable, it will use both Bluetooth SDP and UPnP service discovery to concurrently search for smart objects.

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

With regards to the presentation abstraction, we have designed a minimum set of graphical controls with which we can generate the graphical interface of an EMI²let. Some examples of the classes defined are: EMI²Panel, EMI²Button or EMI²TextBox. This enables us to create EMI²let graphical interfaces which are agnostic of the target mobile device. For instance, when a programmer creates an EMI²Button, it is translated into a button control in a PC or a PDA, but into a menu option in a mobile phone. Still, with the help of the EMI²let Designer (see Figure 4) we can rearrange the layout of the graphical controls of an EMI²let for each of the three target device types supported: PC, PDA and mobile phone. The EMI²let Designer also generates the source code templates for an EMI²let and its server-side counterpart, which can then be edited and compiled to generate the EMI²let binaries ready to be discovered and downloaded.

3.3. Implementation details

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 an MD5 checksum encrypted by a private key only shared by the EMI²let designer and player.

After verification, the player can start the EMI²let by invoking the methods defined in the EMI²let base class, inherited by every EMI²let. The methods defined by this class closely resemble the ones provided by 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.
- `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 after its execution. Thus, it can be executed again in the future. Otherwise, an EMI²let is wiped out from the Player either when its execution is completed or it is out of range, cannot access, the EMI²letObject it represents.
- `notifyDisconnected`, informs an EMI²let when the EMI²letObject that it controls cannot be accessed any longer.
- `getAddresses`, enables the EMI²let Player to retrieve the addresses where an EMI²let server-side is available. For instance, it may be accessible both through a Bluetooth address or a URL pointing to a web service.

The first reference implementation of EMI²lets has used Microsoft .NET, a framework which fully supports reflection through the System.Reflection namespace. Moreover, the .NET platform addresses software development for all the client hardware platforms considered in EMI²lets, i.e. PC, PDA and mobile phone. The EMI²lets presentation controls devised have been based on the ones provided by Compact.NET, the least common multiple.

The most noticeable part of our implementation is the assembly fusion undertaken at the player side merging the arriving 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 interaction, discovery, presentation and persistency mappings implementation with their corresponding plug-in modules. In other words, the code downloaded is linked dynamically (late bound) with the runtime installed in the target device. The .NET class System.Reflection.Assembly is heavily used in this process.

4. An EMI²let discovery plug-in

A good example of an EMI²let plug-in is the service discovery mechanism based on the TRIP [16] tag-based visual identification system which we have developed. A factor that limits the use of Bluetooth as an underlying networking technology for publicly accessible mobile services is that its device discovery process takes a significant
sometimes unbearable) time. The discovery process in Bluetooth is divided into two main phases: (1) device discovery, i.e. what other devices are accessible via Bluetooth, and (2) service discovery, i.e. what services are offered by the discovered devices. In an error-free environment, the device discovery phase must last for 10.24s if it is to discover all the devices [1].

In order to speed up service discovery, we have devised a tag-based content/service selection mechanism, which bypasses the slow Bluetooth device discovery process. Our approach is inspired by the work of [17].

The TRIP visual tags are circular barcodes (ringcodes) with 4 data-rings and 20 sectors. A visual tag, large enough to be detected by a mobile device tag reading software, is shown in Figure 5. The ringcode is divided into: (1) one sync-sector used to specify the beginning of the data encoded in a tag, (2) two checksum-sectors used to encode an 8-bit checksum, which detects decoding errors and corrects three bit errors, and (3) seventeen data-sectors which encode 66 bits of information.

The information in a TRIP tag is encoded in anti-clockwise fashion from the sync sector. Each sector encodes a hexadecimal digit comprising the values 0 to D. The E hexadecimal number is only permitted in the sync sector. Given the 17 data encoding sectors, the range of valid IDs is from 0 to 1577-1 (985261253569359375 ÷ 266).

The TRIP tags were designed to work well with the low-resolution fixed-focal-length cameras found on conventional CCTV systems. Consequently, they are also very well suited for the low-quality built-in cameras of mobile devices, as we suggested in [5]. In fact, our experience shows that the TRIP ringcodes are more reliably recognized than linear (UPC) barcodes, which demand far higher image resolutions. TRIP works reliably with 160x120 pixel images taken at a distance of 5-30 cms from the tags which label the smart objects in an environment. We have implemented the TRIP tag reading software for Compact.NET devices. It achieves 2 fps in a TSM 500 Pocket PC.

4.1. Encoding EMI2lets' addresses

We have used TRIP tags to encode the Bluetooth address of an EMI2let Server and an identifier to select a smart object in that server. Likewise, we have also used those tags to encode tiny urls (see http://tinyurl.com) which point to a smart object in an EMI2let Server. The tiny url server is currently generating 6 character-long identifiers, whilst we can encode up to 8 characters. For example, the identifier 8ggaj maps to the url http://wap.deusto.es. Two bits of a TRIP ringcode are used to encode an EMI2let address type, 00 for Bluetooth and 01 for an Internet tiny url. For Bluetooth, 48 bits are dedicated to encode the BD_ADDRESS of an EMI2let Server, and the remaining 16 bits to encode a unique identifier for a specific EMI2let. For Internet, 64 bits are available to encode a tiny url, containing the tiny url identifier of an EMI2let server-side.

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

Figure 6: Parking EMI2let for PDA (left) and PC (right).

5. EMI2lets applications

The Parking EMI2let, see Figure 6, is a concept application developed with EMI2lets. It shows how a physical object in an outdoors space can be augment with AmI features. It is meant to be deployed in any street parking booth, where we can purchase tickets to park our car for a limited period of time. Often, we have to keep returning to the parking place to renew the ticket so that the local police force does not issue a fine for parking time expiration. Thanks to the 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 every time the user is in the parking place, and remotely via GPRS when the EMI2let warns her (at her office) that its parking ticket is about to expire. This scenario shows the EMI2lets platform capability to enact an action over a smart object both locally, while in the environment, or remotely, far away from the environment. This application is an example of a durable EMI2let.

Other EMI2lets developed have allowed us to perform as diverse tasks as ordering a meal in a busy restaurant, controlling the electronic devices and lights of a room, offering a spoken bus arrival notification for blind people or providing subtitles on mobile phones for deaf people attending a conference.

Figure 7: EMI2let Development and Deployment.
All the EMI2lets developed have followed the development and deployment cycle shown in Figure 7. As we can see the EMI2lets platform provides tools to assist the programmer in the whole development (EMI2let Designer and EMI2 framework) and deployment (EMI2let Player and Server) of smart objects, turning the creation of smart spaces into a much simpler task.

6. Related work

The EMI2lets platform presents some resemblance to the Smoblets software framework proposed by [18]. Both frameworks allow the download of software representatives of objects located in a smart space into a mobile device. However, Smoblets are thought to operate when they are only within range of the smart object they represent. On the contrary, EMI2lets remain at the user’s terminal, even when he is far away from the smart object. This allows the user to control that smart object anytime and anywhere, both using local (Bluetooth, Wi-Fi) and global (GPRS/UMTS) communication mechanisms. Furthermore, the main application of Smoblets is to transform mobile devices into execution platforms for code downloaded from smart items with limited processing resources, whereas 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 a main innovation, the code generated for an EMI2let is independent of the target platform type where it will be run (PC, PDA or mobile phone). This is due to the fact that our layered software architecture follows a “write once run in any device type” philosophy.

Other authors [17] have also used tags (based on our TRIP tags) to encode addresses of smart objects. Our data encoding strategy, using the same number of rings as them, achieves better error correction capabilities (from 2 to 3 bits) and has a higher encoding capacity (from 63 to 66 bits).

7. Conclusion and further work

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

- Transforms mobile devices into universal remote controllers of smart objects.
- Enables both local and global access to those smart objects, i.e. anywhere and at anytime.
- Independent and extensible to the underlying service discovery and interaction, graphical representation and persistency 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 EMI2let Players and Servers, following the WebProfiles model described in [11]. In addition, we want to enable the cooperation of smart objects, for instance, through the creation of a distributed shared tuple space. Finally, we intend to incorporate Semantic Web features to our framework, which may move the user “out of the loop” in the EMI2lets discovery and execution process, as suggested in [19].

References