A Middleware for the Deployment of AmI Spaces

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

[1] University of Deusto, Faculty of Engineering, Avda. Universidades 28, 48007 Bilbao, Spain
{dipina, ivazquez}@eside.deusto.es,
{dgarcia, jafernan, ivgarcia}@ctme.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. A radically different application domain for them may be represented by Ambient Intelligence (AmI), where mobile devices can be used as intermediaries between us and our surroundings. This paper proposes a middleware with 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) [1] 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 interesting processing and storage capabilities, varied communications mechanisms (Bluetooth [2], Wi-Fi, GPRS/UMTS) and increasingly capable multimedia capture and playback facilities. Moreover, they are far more easily programmable (Compact.NET [3], J2ME [4] or Symbian [5]), i.e. extensible, than ever before.

Mobile devices equipped with Bluetooth, built-in cameras, barcode or RFID readers can be considered as sentient devices [6], since they are aware of what smart objects are within an AmI space. We understand by AmI space or 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. Once a mobile device discovers a nearby smart object, it can 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 EMI²lets, 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.

2 EMI²: 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 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 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.

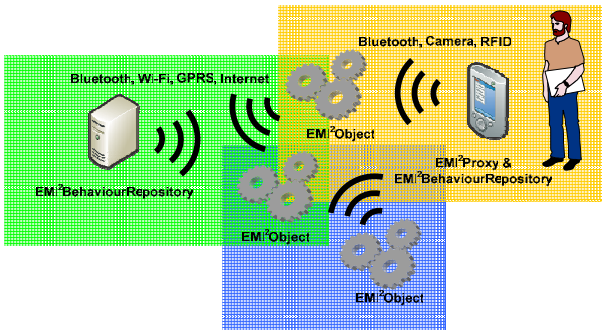

Fig. 1. The EMI² Architecture

Fig. 1 portrays 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.

- *EMI²Object*: 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 EMI²Object cooperates 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.

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 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 EMI²Objects

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 [9]. Firstly, solutions in which discovery protocols are supported by mobile code, e.g. Jini [10]. 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) [11]. Finally, there are interaction independent discovery protocols such as the Service Location Protocol [12].

In what follows we explain the design and implementation of 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 plaftorm 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 mobile context-aware application.

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 EMI²let should be able to run in a mobile, a PDA or a PC.

3.1 The EMI²lets Vision

Fig. 2 shows a possible deployment of an EMI²let-aware environment. A group of devices running the EMI²let Player and hosting the EMI²let runtime can discover and interact with the software representatives (EMI²lets) of surrounding EMI²Objects. An EMI²Object may be equipped with enough hardware resources to host an EMI²let Server, or alternatively a group of EMI²lets associated to different EMI²Objects may all be hosted within an autonomous version of an EMI²let Server. The EMI²let Server acts as a repository of EMI²Objects. It publishes the services offered by the hosted EMI²Objects, transfers them on demand to the requesting EMI²let Players, and, optionally acts as running environment for the EMI²let server-side facets.

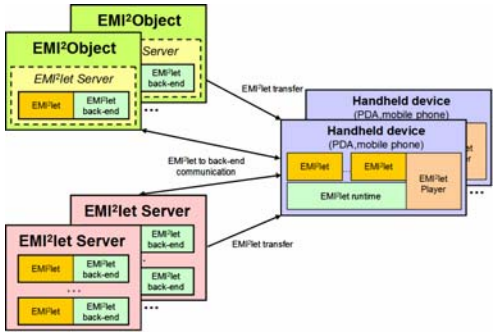

Fig. 2. The EMI²lets in action.

Some EMI²lets may directly communicate with their associated EMI²Objects 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 equipped. For those cases, it will be more convenient to delegate those cumbersome 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.

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 life of a downloaded EMI²let. A version of the player is available for each device type which may act as host of EMI²lets, e.g. PDA, mobile phone or PC.
5. An EMI²let Server which acts as repository of EMI²lets and as running environment of EMI²lets server-sides.

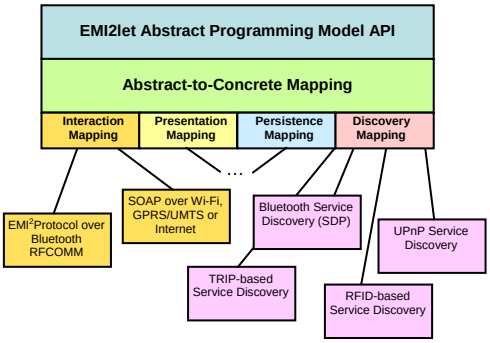

Fig. 3. EMI²lets Internal Architecture

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 *EMI²let Abstract Programming Model API*, to develop the software counterparts of smart objects. This layer offers a set of generic interfaces (abstract classes) covering the main functional blocks of a mobile sentient application:

1. *Discovery* interface to undertake the search for available EMI²lets independently of the discovery mechanisms used underneath.
2. *Interaction* interface to issue commands over the services discovered.
3. *Presentation* interface to specify the graphical controls and events that represent the look and feel of an EMI²let.
4. *Persistency* interface to store EMI²let-related data in the target device.

The *EMI²let Abstract-to-Concrete Mapping* layer translates the invocations over the generic interfaces to the appropriate available mechanisms both in the mobile device and the EMI²Objects in the environment. The discovery, interaction, presentation and persistency abstractions encapsulate the concrete discovery, interaction, presentation or persistency models used. They 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 classes used 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 of 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 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 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 EMI²let. Some examples of the classes defined 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 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* layers and plug-ins will be linked to the arriving EMI²let in an EMI²let Player.

3.3 Implementation

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 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, from which every EMI²let must inherit. 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 EMI²let-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 of the EMI²Object it represents.

- `notifyDisconnected`, offers an EMI²let the possibility of being aware of when the EMI²Object that it controls cannot be accessed any longer.
- `getAddresses`, enables the EMI²let-hosting player to retrieve the addresses at which the EMI²let 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 EMI²lets, namely PC, PDA and mobile phone. As a least common multiple for the definition of the presentation controls of an EMI²let, we have chosen 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 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 assembly code downloaded is linked dynamically (late bound) with the runtime installed in the target device.

4 An EMI²lets Application

The Parking EMI²let, see Fig. 4, is an example application developed with EMI²lets middleware. It shows how a physical object in an outdoors space can be augmented with AmI features. This application 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, their capability to enact an action over an EMI²Object both locally, while in the environment, or remotely, far away from the environment.

Other EMI²lets developed have allowed us to perform as diverse tasks as ordering food in a busy restaurant from our mobile phone, controlling the electronic devices and lights of a room, offering a voice synthesized bus arrival notification for blind people or provide subtitles on the mobile phones of people attending to an opera.

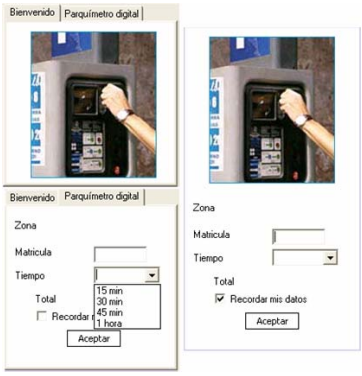

Fig. 4. Parking example for PDA(left) and PC(right).

5 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 [9]. 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).

6 Conclusion

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

- Transforms mobile devices into universal remote controllers of smart objects.
- Enables both local and global access to those smart objects (anywhere/anytime).

- Independent and extensible to the underlying service discovery and interaction, graphical representation and persistence mechanisms.
- Enables AmI spaces using conventional readily-available hardware and software.
- Follows a "write once run in any device type" philosophy for EMI^{2}lets.

The EMI^{2}lets middleware represents a good approach to make the AmI vision reality. With it, we have been able of prototyping several active influence AmI scenarios in a very simple manner.

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.

References

1. Shadbolt N. (2003) *Ambient Intelligence,* in *IEEE Intelligent Systems*, vol. 2, no.3.
2. (2005) *Bluetooth Specification version 1.1*, http://www.bluetooth.com.
3. (2005) *Mobile Developer Center*, http://msdn.microsoft.com/mobility/, Microsoft Coorporation.
4. (2005) *Java 2 Platform, Micro Edition (J2ME)*, http://java.sun.com/j2me/, Sun Microsystems, Inc.
5. (2005) *Symbian OS – the mobile operating System*, http://www.symbian.com/, Symbian Ltd.
6. López de Ipiña D., Vázquez I. and Sainz D. (2005) *Interacting with our Environment through Sentient Mobile Phones*, in *Proceedings of 2nd International Workshop in Ubiquitous Computing* (IWUC-2005), ICEIS 2005, pp. 19-28, ISBN 972-8865-24-4.
7. Vázquez, J.I., López de Ipiña, D. (2004) *An Interaction Model for Passively Influencing the Environment*, in *Adjunct Proceedings of the 2nd European Symposium on Ambient Intelligence*, Eindhoven, The Netherlands.
8. Vázquez, J.I. and López de Ipiña D. (2005) An HTTP-based Context Negotiation Model for Realizing the User-Aware Web, in 1st International Workshop on Innovations In Web Infrastructure (IWI 2005), Chiba, Japan. May 2005.
9. Grace P., Blair G. S. and Samuel S. A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments, in Mobile Computing and Communications Review, ACM SIGMOBILE, vol. 9, no. 1, pp. 2-14.
10. (1999) Arnold K., O'Sullivan B. et al. *The Jini Specification*, Addison Wesley.
11. (2005) *The Universal Plug and Play Forum*, http://www.upnp.org/.
12. Czerwinski S., Zhao B. et al. *An architecture for a Secure Service Discovery Service*, in *Proceedings of MobiCom'99*, August 1999.
13. López de Ipiña, D., Mendonça P. and Hopper A. (2002) *TRIP: a Low-cost Vision-based Location System for Ubiquitous Computing*, in *Personal and Ubiquitous Computing*, vol. 6, no. 3, pp. 206-219.
14. Siegemund, F. and Krauer T. (2004) Integrating Handhelds into Environments of Cooperating Smart Everyday Objects, in *Proceedings of the 2nd European Symposium on Ambient Intelligence. Eindhoven*, The Netherlands.