Towards a communication agnostic middleware for Ambient Intelligence

David Sainz[1], Aitor Almeida[1], Jon Valdés[2], Diego Lopez de Ipiña[1]

Mobility Research Lab – More Lab
University of Deusto
Avenida de las Universidades, 24 4007 Bilbao (SPAIN)

[1]{dsainz, aalmeida, dipina} @eside.deusto.es
[2]juanval@gmail.com

Abstract. . In this paper a middleware framework for the implementation of Ambient Intelligence related applications is outlined. The framework abstracts the developer form the communication methods and technologies and the platform specific user interface problems, allowing him to concentrate in the business logic.

1 Introduction

The concept of Ambient Intelligence (AmI) [1] defines an interaction model between the users and the environment surrounding them, which adapts itself according to the needs and the actual context of each individual. This environment is formed by various common objects equipped with processor power and a certain degree of intelligence that enables them to offer a variety of services (smart objects). Thus, each smart object can adapt separately or in groups to fulfil the required needs.

Mobile devices such as PDAs or mobile phones are the perfect accessories to act on behalf of the users and interact with the environment transparently [9]. They can collect all the desired preferences of a person and actively adapt the context of the environment to them. They can also serve as an intermediate tool to remotely manage smart objects, displaying an interface to the final user. These devices are becoming more common and are also very easily programmable through a set [2] [3] [4] of different technologies.

In this paper we propose a framework to create and run small programs used to control smart objects through mainly mobile devices. This framework is our solution to create and manage AmI scenario. Section 2 describes the overall architecture of the proposed framework, explaining the involved elements. Section 3 shows the design of the graphic controls used to abstract the developer from the target platforms. Section 4 presents the agnostic communication model and finally section 5 suggests future work to be done.

2 The architecture of EMI2lets

The EMI2lets architecture is composed of:

1. *EMI2Peers:* These components are installed inside each network element (enhanced smart objects, Smartphones, PDAs, etc.). They control the functionality of the peer and expose it so it can be managed remotely. The peers form a peer-to-peer network.
2. *EMI2let Players:* Installed on the mobile devices, they offer all the required functions to manage the intelligent environment around them. The players are designed to be as context-aware as possible. When a user finds an AmI environment, the player starts searching for smart objects and their EMI2Peers to offer all the functions they have exposed. Once a concrete EMI2Peer is selected, the required software to control its specific functions is downloaded to the player and run transparently. The EMI2Player renders the graphic controls depending on the host platform. Since the mobile device is aware of its surrounding environment, it can be considered as a sentient device [5].
3. *EMI2lets:* are the pieces of software run on each EMI2Peer. Each EMI2Peer can be composed by two parts, the public one which is transmitted to other EMI2Peers and can have a graphical interface that is renderized by the EMI2Player and the private, which is not delivered to other EMI2Peers and contains business logic. These small programs are designed specifically for each smart object.

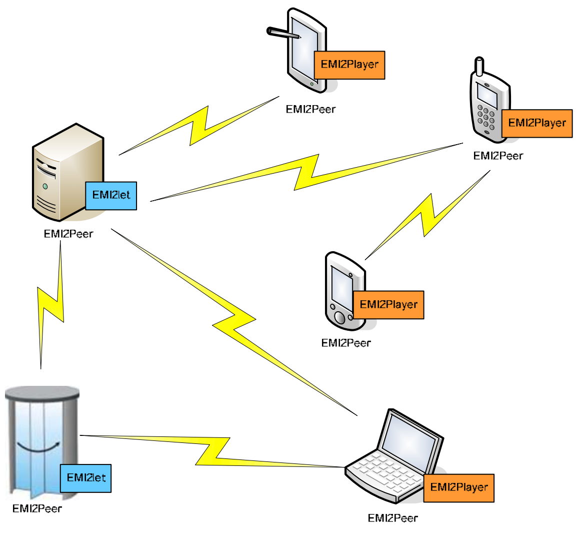


Fig. 1. The EMI2Peers form a peer-to-peer network

One of the key features of the EMI2lets is the fact that the same software can be run in multiple device types, including PDAs, mobile phones and laptops. The architecture is designed so that the graphic controls adapt themselves to the platform they are running on. Another important feature is its agnostic communication model that provides a seamless communication mechanism through different technologies. The architecture automatically chooses the optimal technology to perform the data transfers transparently to both users and EMI2let developers.

The programs used to control the environment are downloaded once the player is near the EMI2Peers using a technology like Bluetooth [6], but they can be used even when the player is far away from the smart object. The program can remain in the player for future needs and use a large distance communication technology if available to communicate. This is one of the features that distinguishes EMI2lets from other similar frameworks like Smoblets [7].

The current implementation is being developed in .NET, taking advantage of its multiplatform capabilities. To make the framework compatible with the three platforms (Desktop, Pocket PC and Smart Phone) the design is based on the Smart Phone subset of classes. This also restricts the set of classes the EMI2let programmer can use.

3. EMI2lets user interface library

EMI2lets includes a graphic controls library called EMI2.Controls that helps to free the developer from having to deal with all the platform-specific GUI controls implementations. The purpose of this library is to overcome the existing differences between all the supported platforms. For instance, Smartphone controls implementation is very limited – it doesn't even support buttons– which leaves the multi-platform developer having to choose between two solutions: use just the small subset of controls available for all platforms, or develop a different GUI for each one.

EMI2.Controls tries to address that problem giving the developer a set of controls that behave in a predictable way no matter which platform the program is being run on. The developer will only work with a class that represents the abstract GUI control, managing its properties programmatically and without caring about the target platform. The control will render itself specifically for the platform at execution time.

3.1. Implemented Controls

As of this moment, a small set of controls has been implemented to test the system, but a complete EMI2.Controls library is intended to be implemented to make a full-featured system to build any kind of platform-independent GUIs. These controls share some common attributes – such as Location, Size, Show or Clicked – and also posses few attributes of its own.

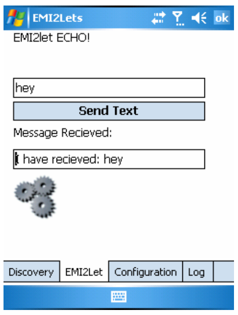

EMI2Lets
EMI2let ECHO!
hey
Send Text
Message Recieved:
I have recieved: hey

Fig. 2. An Emi2let running in a Pocket PC. The controls are renderized differently in each platform.

3.2. Internal EMI2.Control structure

All EMI2.Controls have the same structure consisting of 3 components, as it can be seen in figure 3

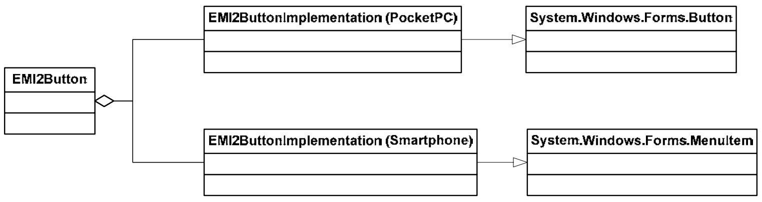

EMI2Button
EMI2ButtonImplementation (PocketPC)
EMI2ButtonImplementation (Smartphone)
System.Windows.Forms.Button
System.Windows.Forms.MenuItem

Fig. 3. Structure of an EMI2Button, showing the programmer-visible layer (EMI2Button), PocketPC and Smartphone implementations (EMI2ButtonImplementation) and the real controls displayed to the user (Button and MenuItem).

Of all these components, EMI2Button will be the only one visible to the programmer, as it is the class that exposes every public atribute of a button. EMI2ButtonImplementation is the real behavior of the GUI control, is platform-dependant and loaded at run-time from an assembly using reflection. Each platform will have its own assembly containing all the required implementations. Therefore, an underlying control that is shown to the user is created. The code to manage a EMI2Button is the same, but the behavior differs as it is merged with the EMI2ButtonImplementation at run-time. For example, as it can be seen in the figure running on a PocketPC EMI2Button will create and display a Button, whereas in Smartphones it will create a MenuItem instead.

However there are some inherent limitations to the chosen design. As PCs and the different mobile platforms have surprisingly different feature sets, the library does not implement some very useful features, as there would be no equivalent in other platforms (e.g. Drag&Drop in Smartphones).

4. The agnostic communication model

The EMI2lets platform is designed to abstract completely the user from the communication technology used by the player to discover EMI2Peers and transfer the data. This is achieved using a common abstract class for each of the plug-in families: IEMI2CommunicationPlugin and IEMI2DiscoveryPlugin. The implementation of communication and discovery methods lies on the technology specific plug-ins.

Each of the EMI2peers has a collection of EMI2Channels, representing each channel a connection with a remote peer. The peers have an array of connection strings, being reachable by multiple transmission methods. When a new EMI2peer is discovered metadata is exchanged in the subsequent handshake. The metadata describes the EMI2peer, the available communication methods and the residing EMI2lets.

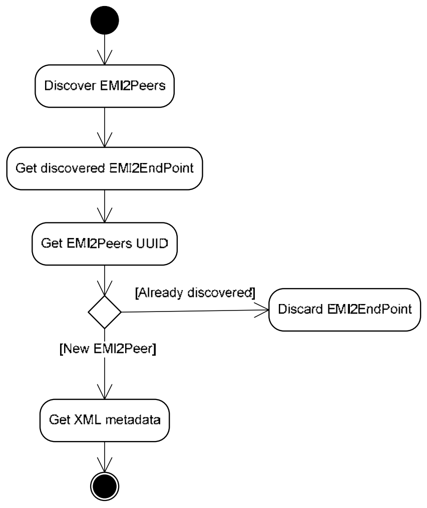
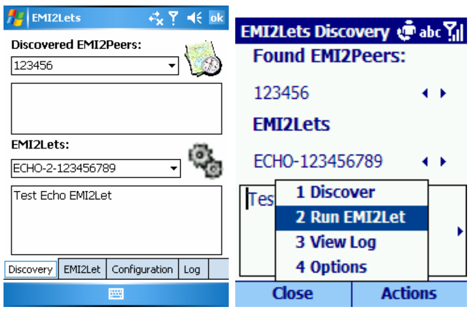


Fig. 4. Diagram of the discovery protocol. The discovery is done in 3 steps. First the EMI2Endpoints are discovered by the player. Then the player requests the UUID of the EMI2Peer who owns the EMI2EndPoint. If the EMI2Peer has not been already discovered XML metadata is requested.

The metadata format is the following:

```
<EMI2PEER>
      <UUID>peer uuid</UUID>
      <NAME>peer name</NAME>
      <DESCRIPTION>peer description</DESCRIPTION>
      <ENDPOINTS>
            <ENDPOINT type='type'>connection string
      </ENDPOINT>
      <ENDPOINTS>
      <EMI2LETS>
            <EMI2LET uuid='uuid'>
                  <NAME> EMI2let name </NAME>
                  <DESCRIPTION> EMI2let
                  description</DESCRIPTION>
            </EMI2LET>
      </EMI2LETS>
</EMI2PEER>
```

Each EMI2Peer, EMI2Channel and EMI2let is identified by a unique UUID. The EMI2lets' assemblies are not transmitted with the metadata to optimize the communication, being recovered by player request only if it wasn't previously downloaded.


Discovered EMI2Peers: 123456

EMI2Lets: ECHO-2-123456789

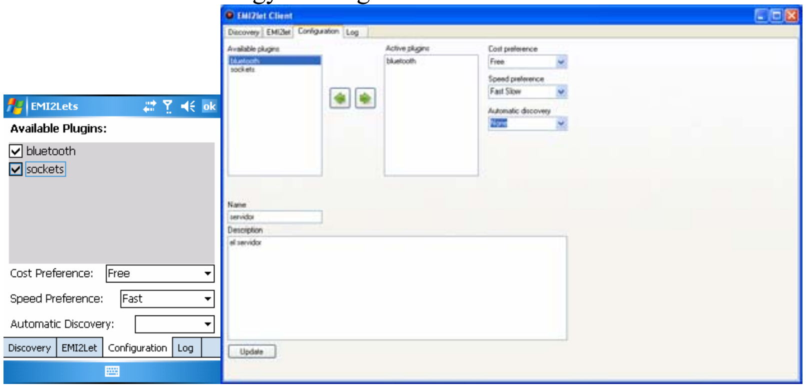
Test Echo EMI2Let

Found EMI2Peers: 123456

EMI2Lets: ECHO-123456789

Fig. 5. A Pocket PC and a Smartphone Client discover an EMI2Peer and its EMI2lets

Users are able to configure connection preferences, choosing whether to use a charge free technology or not. If the connection with an EMI2peer is lost during the use of an EMI2let the client seamlessly uses the next available connection string according to the stated preferences. In the same way, if the client is using a non-free technology (e.g. GPRS) and while moving discovers a free endpoint (e.g. Bluetooth) for the same EMI2Peer technologies will be switched automatically. These operations are completely transparent for the EMI2let developer, being his only concern the business logic of the application. The developer sends and receives data ignoring which underlying communication technology is being used at each moment.


Available Plugins: ☑ bluetooth ☑ sockets

Cost Preference: Free

Speed Preference: Fast

Automatic Discovery:

Available plugins: bluetooth, sockets

Active plugins: bluetooth

Cost preference: Free

Speed preference: Fast Slow

Name: servidor

Description: el servidor

Fig. 6. Users can configure their connection preferences and the communication plug-ins to use.

One of the main drawbacks of the current model is that connection is stateless due to the nomadic and non-reliable nature of the EMI2Peers.

5. Conclusion and future work

In this paper we have defined the architecture of a framework used to manage AmI environments via different types of devices using seamless communication trough various kinds of communication technologies. This framework transforms mobile devices into universal remote controllers capable of managing the smart objects existing inside their area of action, using even large distance communications to ensure global operation anywhere.

This framework offers a high degree of abstraction: The communication and presentation mechanisms are transparent for the user and the developer, who do not have to concern about which communication technology must be used or what kind of graphical interface must be rendered for each device. The design offers also high degree of interoperability since the framework is designed in a way that a single piece of code can run in many device types.

In future work we want to want to add UPnP [8] as new discovery mechanism inside the set of available technologies and cooperation [10][11][12][13] between different EMI2lets.

References

[1] Shadbolt N.: Ambient Intelligence. IEEE Intelligent Systems, Vol. 2, no.3, (2003).

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

[3] Mobile Developer Center, http://msdn.microsoft.com/mobility/, Microsoft Coorporation (2005)

[4] Symbian OS – the mobile operating System, http://www.symbian.com/, Symbian Ltd. (2005)

[5] López de Ipiña D., Vázquez I., Sainz D.: Interacting with our Environment through Sentient Mobile Phones. Proceedings of IWUC-2005, pp. 19-28, ISBN 972-8865-24-4 (2005).

[6] Bluetooth Specification version 1.1, http://www.bluetooth.com

[7] Siegemund, F., Krauer T.: Integrating Handhelds into Environments of Cooperating Smart Everyday Objects

[8] Universal Plug and Play Forum, http://www.upnp.org (2005)

[9] George Roussos, Andy J. Marsh, Stavroula Maglavera. Enabling Pervasive Computing with Smart Phones. Pervasive Computing, April-June 2005 pp. 20-27.

[10] Marco Conti, Enrico Gregori, Gaia Maselli. Cooperation Issues in Mobile Ad Hoc Networks, 24th International Conference on Distributed Computing Systems Workshops - W6: WWAN (ICDCSW'04), March 2004, pp. 803-808

[11] T. Andrews. F. Curbcra. H. Dholakia, Y. Goland, J. Klcin, F. Lcymann, K. Liu, D. Roller, D. Smith, S. Thanc. 1. Trickovic, and S. Wcerawdrana, 2003, Business process execution language for web services version 1.1. Technical report, BEA. IBM, Microsoft, SAP. and Siebel Systems.

[12] William B. Bradley, David P. Maher , 2004, The NEMO P2P Service Orchestration Framework. Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9. pp. 90290c

[13] Aitor Almeida Escondrillas, David Sainz Gonzalez, 2006, Enabling Service Orchestration, Transactionality and Security in UPnP. IADIS International Conference on Applied Computing.