DESIGNING MIDDLEWARE FOR PERVASIVE COMPUTING

Lucian Burja and Mihai Tanase

Abstract.Intelligent cell phones, palmtops, PDAs and other mobile devices are fast becoming popular and place new challenges on software development. Embedded devices that are highly mobile create a demand for network technologies that enables them to have access to services anywhere, at any time, and on any device. This paper covers the design and implementation of a service-oriented platform, codenamed Airborne, which aims to provide an efficient, robust and secure framework for developing service-oriented applications for the challenging world of mobile, pervasive computing.

1. Introduction

Technological advances over the last few years have revolutionized the world of personal computing. The IT world is at the beginning of a new information paradigm: *pervasive computing* - anywhere, anytime data access on any device. It is a transition from a closed, constrained world of desktop computers communicating with each other using wires (copper or optics fiber) to a more natural environment with mobile users, equipment and resources. This paradigm could be historically compared with the invention of flight - an invention that allowed people improved transports and more efficient communications, hence the name of this project: *Airborne.*

As mobile devices tend to become ubiquitous in our modern society, the need for software grows at a rapid rate. However, software for mobile devices tends to have different characteristics than traditional software written for desktop computers. Why would the software be different? At first, this question seems justified. After all, what really changes is the communication means between devices, problem well covered by electric engineering. Nevertheless, analyses done so far have pointed out several particularities that influence application development within mobile devices directly:

- Resources: Even though handheld and mobile devices have come a long way, they are still a generation away in terms of capabilities compared to desktop systems. This mismatch is apparent on nearly all aspects: processing power, screen size, battery power etc. Additionally, bandwidth for wireless access is clearly lower compared to wire line networks even with the arrival of next generation wireless 3G networks. In fact, one may easily argue that *pervasive systems will always be one order of magnitude or more inferior to that of traditional computing systems.*

- Mobility: A user with a mobile device will move between several networks and domains. Because of this, an application must be aware and handle well unexpected connection state changes. This implies several layers of management, such as IP address management, caching management, authentication management etc. For example it is possible that the caller wants to change the service provider due to administrative or quality-related reasons. This shouldn't result in data loss for the user, or an unresponsive application. A user migration in another geographical region offers the motivation for building a new kind of computing: *location-aware computing.* This kind of computing could benefit from the location information in many ways, from information regarding the weather to direction information.

- Security is a factor that has to be taken seriously into consideration. Distributed, mobile systems are implicitly more vulnerable than wired systems because the communication is performed in open air. This fact implies a platform providing powerful security measures that prevent the interception or altering of communication.

Having the particularities of mobile devices in mind, we can outline important features that software should have:

- Display independence: Mobile devices often have small displays, with different shapes and dimensions. Ideally, software should be able to render a user interface correctly, no matter what the output characteristics of the device are. This can be accomplished by using a XUL-like language for describing user interfaces and having modules dynamically build the user interface at runtime according to the particularities of the device.

- Efficient algorithms: Because computer power is low, algorithms should be extremely well implemented. Algorithms that are too complex can't be used and must be delegated to more powerful units like desktop computers or servers. An interesting idea would be to combine the power of multiple small devices in order to complete a given task, an idea that brings us close to *distributed computing*.

- Low footprint: Speed is not the only reason why complex algorithms can't be implemented. Because of the limited memory that most of the devices have, code size can be a problem. To reduce the code size, obfuscators can be used, but ultimately clean and simple designs will have a major impact.

- Reliable networking: Because wireless device users are going to be mobile users, they will make new kinds of demands on network resources. The application will have to possess the intelligence, and the processing power, to give that user maximum performance no matter where he or she is, and no matter what device he or she is using.

- Service oriented: Software tends to be inherently fragile to modifications. However, the dynamic world of business demands flexibility in software design because of frequently changes needed in order to adapt to market conditions. By choosing a service-oriented architecture, we gain important advantages like interoperability, platform and language independence and resistance to changes, due to isolation of services.

Because of the constraints mentioned above imposed on applications, designing programs for mobile devices is challenging. A framework to hide complexity from the programmer would be a welcomed addition that could improve the quality of software and boost productivity.

2. Overview of existing technologies

Several projects and initiatives have tried in the past to fill the need of a service oriented platform for embedded devices. Each of them has succeeded to a degree, but has limitations that make it more or less suitable for a given task.

Jini is a distributed network technology that is designed to handle dynamic environments. It is based on the assumption that Java is present in the network, and ensures a homogeneous environment by use of mobile code that is transferred between clients in the network. All resources in the network are defined as services, and can be found through a lookup service, a sort of repository holding reference to all services. Jini technology has several disadvantages:

- The assumption that "Java is everywhere" might not be true. Jini needs a full J2SE environment and most of the devices on the market today are too resource-restrained to fulfill this premise.
- Jini is dependent on TCP/IP which is most often not supported in wireless, mobile devices;
- The reference implementation from Sun uses RMI which implies downloading of stubs and might be to heavyweight for resource-constrained devices.

JXTA is a distributed platform that tries to standardize peer-to-peer. The objective is to provide interoperability between entities in the network, be platform independent and offer ubiquity, i.e. any digital device can participate. JXTA defines a number of concepts that are common for peer-to-peer networks like peer, peer group and pipes. These concepts are the primary components of the JXTA platform.

The base of the JXTA specification is a set of protocols that enable discovery of resources on the network and the ability to connect and communicate with these resources. JXTA provides a package that makes it possible for J2ME-enabled devices to participate in the community. However, some drawbacks of JXTA are:

- JXTA constitutes an extensive framework. This might prevent the use of the specification since there is plenty to learn before it can be used, and a lot to keep track of.
- JXTA protocols do not by themselves promise interoperability. Applications must be designed to be interoperable.
- The J2ME implementation uses HTTP as the underlying protocol and requires a proxy for the communication between two devices. Although there is an initiative for a proxy-less implementation, it is in the early stages and it's not clear if it will be completed.

Web Services have been much hyped as a technological revolution lately. Although interoperability and language independence can be achieved through use of web services, the situation is a little different in the world of embedded devices. Current implementations suffer from a major drawback that makes them unsuitable for our purposes: a mobile device can consume services but can not be a service-producer. Thus, it is assumed that a powerful server that can produce web-services is always present, and a peer to peer networking is not possible.

3. Detailed description of the *Airborne* middleware platform

The purpose of this project is to offer a *service oriented framework* for supporting applications that run within the heterogeneous, unstable and constrained environment of wireless communications. Due to the particularities of the wireless environment, Airborne is a *distributed system*, built upon mobile entities, capable of offering robust services to the user.

The architecture is based on several components:

- Service - Main part of the architecture, the service is the reason for using this platform. The service is asked for, identified and used to perform a well defined task. Services vary from information and financial to localization and computing.
- Service Provider - A service provider is a functional unit that provides a service. It must also provide additional information and means of calling the service and gathering results. The service is performed inside a service provider.
- Service Consumer - A service user's abstraction is a service consumer. It is the one that asks for a service and after finding one will transmit necessary parameters and will gather results.
- Service Publisher - The service publisher is the unit that makes the services of a certain service provider known to service consumers. A service provider uses a publisher to announce its services and, in turn, a consumer asks a publisher to find the providers of a certain service.

- Message -The message is the abstraction of information exchanged between various units of the system. Messages may be simple, of the question - answer type or may be more complex, containing auxiliary data.
- Mobile Device - The mobile device is the equipment used to access the system. This abstraction would contain information about a specific device and its capabilities of executing specific tasks. This information contains processor performance, memory and storage and communication performance.
- Communication Channel - This would represent an abstraction of the communication channel between system entities. It should achieve transparency over communication environments used by the system.

In order to enhance productivity, security and robustness, the project provides **core services** that implement common functionality useful to most applications:

- Address resolution service- maps devices names to addresses. According to the protocol used for communication, a name could map to a TCP/IP address or a Bluetooth one. Because of this service address transparency can be achieved;
- Service publishing and discovery - provides mechanisms to publish and discover other services. A device can be both service provider and service consumer, thus peer-to-peer networking is possible;
- Routing service - this service is responsible for providing communication in ad-hoc networks. Messages are routed within a heterogeneous environment from one equipment to another;
- Communication management service - monitors connection status and implements queuing and caching mechanisms when communication becomes unstable. Devices can sometimes operate in offline-mode and send cached data when a connection becomes available.
- Device management service - this service is responsible with displaying information according the capabilities of a particular device. To achieve device independence, user interface are not hard-coded. A XUL-like XML language is used to describe user interfaces, this service being responsible with rendering the interface on a particular device.

- Security management service - handles user authorization and authentication inside the system and also secures information transfer through encryption.

From an implementation's point of view, the architecture is built around a so-called *request dispatcher*. Running in the background it has the purpose of listening to incoming requests. When a request is received, the request dispatcher analyzes the header and delegates the appropriate service to handle the request. Because the service knows the structure of the message, it can interpret the message and issue an appropriate response to the remote device.

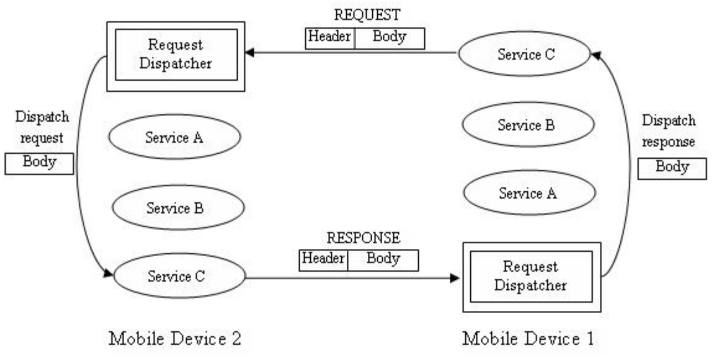

Figure 1: Invocation of a service

The header of a message looks as follows:

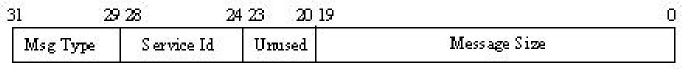

Figure 2: Message header

where:

- *Message type* is a 3-bit field, encoding the type of the message. The type can be one of the following: REQUEST, RESPONSE, BROADCAST and ALERT. 4 values are reserved for future extensions;

- *Service Id* is a 5-bit field that uniquely identifies a service. Using this field, the request dispatcher knows to which service to pass the message.
- *Message size* is uses 20 bits to specify the length of the message body. The message body can be up to 1MB.

As we can see, 4 bits remain unused. They are reserved for future uses and could be useful for extending the size of the message above 1MB or for adding support for more than 32 services.

The message body, following the header is not interpreted by the request dispatcher, but passed as it is to the appropriate service. The body is usually XML that will be parsed to extract the information. By use of XML opposed to a binary format, language and platform independence is achieved.

As an example we will analyze how the *discovery service* works. Supposed we want to find out what services run on a remote device, following steps are performed:

- Using the *address resolution service*, we obtain the IP (or Bluetooth) address for the remote device;
- We build the message by setting the header fields to the correct values (type=REQUEST, Service Id=DISCOVERY_ID, length=body size in bytes) and providing the necessary XML body. This would look like this:

  ```
  <msg>
  <sender "my Device"/>
  <service cmd="list"/>
  </msg>
  ```

  Thus, the remote service would know to perform the proper action (list available services) and send the response to our device (”my Device”).
- On the remote side, the dispatcher will receive the message and by analyzing the header it will delegate the discovery service to take care of the request.
- The remote discovery service will send the response back to us.

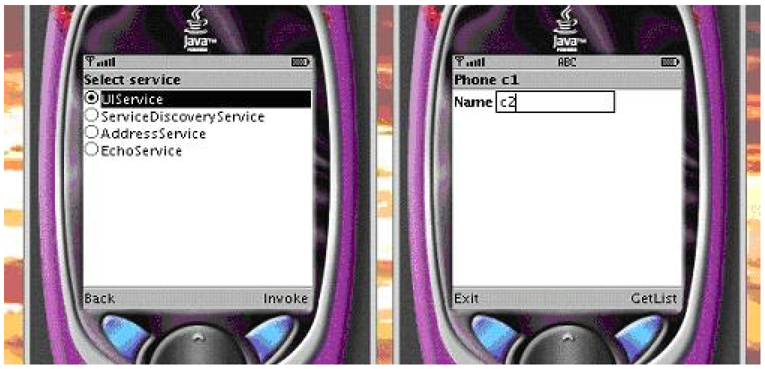

Figure 3: Invoction of discovery service

To provide the most hardware and software independence we decided to use Java technology in combination with XML. Java is a popular choice for developing applications called MIDlets, and most modern cell phones have Java support built-in.

The use of XML for communication makes it possible for services to interoperate independent of the programming language used. Hence, interoperability with C# or C++ clients on devices where Java is not available is possible.

One goal of the design was to be able to invoke remote services that are unknown to the local device and process the response correctly. This was achieved by using an XML description of the UI that can be interpreted and displayed on the local device.

4. Conclusions and future work

The Airborne project aims to simplify development of mobile device applications by providing a robust, scalable and secure platform upon which applications can evolve. Built around Java and XML, great care has been taken to create an open platform that can integrate in heterogeneous environments, interoperating with other technologies based on C# or C++.

Core elements of the platform have reached a functional level.Components like service publisher, service producer, service consumer and core services have been successfully implemented. The system is able to discover remote services, invoke them and process the response. However, some crucial aspects need to be further addressed for the platform to reveal its full potential. Key services, like caching or message routing need to be further addressed for the platform to provide reliable communication. Security must also be well considered, because wireless communication is inherently insecure.

References

[1] Anind K. Dey, *Providing Architectural Support for Building Context-Aware Applications*, Georgia Institute of Technology.

[2] Michael Juntao Yuan, *Overcoming Challenges in Mobile J2ME Development*, Prentice Hall PTR., 2004

[3] Miguel A. Munoz, Marcela Rodriguez, Jesus Favela, Ana I. Martinez-Garcia, Victor M. Gonzales, *Context-Aware Mobile Communication in Hospitals*, Computer magazine, September, 2003

[4] Sveen Lyngstad, *Network technologies for Java-enabled mobile devices*, Master Thesis, Norwegian University of Science and Technology, July 2002

[5] Swarup Acharya, *Application and Infrastructure Challenges in Pervasive Computing, Bell Laboratories*, Lucent Technologies, Inc.

Authors:

Lucian Burja
Technical University of Bucharest
Spaliul Independentei, BucharestRomania
e-mail: lucian.burja@gmail.com

Mihai Tănase
Technical University of Bucharest
Spaliul Independentei, Bucharest
Romania
e-mail: mihai.tanase@uti.ro