A J2ME transparent middleware to support HTTP connections over Bluetooth

Vincenzo Auletta, Carlo Blundo, and Emiliano De Cristofaro
SCNLab - Dipartimento di Informatica ed Applicazioni
Università di Salerno
I-84084, Fisciano (SA), Italy
Email: {auletta, carblu, emidec}@dia.unisa.it

Abstract— Over the last years, a call for embedding computation into the environment has been arisen. This philosophy has been often referred to as pervasive or ubiquitous computing, to remark the aim to a dense and widespread interaction among computing devices. User intervention and awareness are discarded, in opposition to an automatic adaptation of applications to location and context. To this aim, much attention is drawn to technologies supporting dynamicity and mobility over small devices which can follow the user anytime, anywhere.

The Bluetooth standard particularly fits this idea, by providing a versatile and flexible wireless network technology with low power consumption. Operating in a license-free frequency, users are not charged for accessing the network nor they need an account with any company. Bluetooth dynamically sets up and manages evolving networks, by providing the possibility of automatically discovering devices and services within its transmission range.

Research studies have forecasted that within a few years, most of the devices accessing the Web will be mobile, and presumably most of them will be Bluetooth-enabled. Therefore, we need solutions that encompass networking, systems, and application issues involved in realizing mobile and ubiquitous access to services.

In this paper, we present a transparent middleware which extends the possibility of accessing Web resources also from Bluetooth-enabled smartphones. All the implementation details will be hidden both to users and to application developers, allowing an easy and complete portability of applications working on traditional TCP/IP communication protocols towards the Bluetooth technology.

I. INTRODUCTION

The evolution of technology drove a deep transformation of users habits, with an increasing requirement of support to mobility and connectivity. Technology advances have also led to the need of providing a brand new set of applications, which fit the mobile environment and allow device interactions over wireless channels. Smartphones are nowadays small and powerful enough to turn into fundamental working instruments, and to be considered for the deployment of complex applications.

In the last years, several protocols have been presented for wireless communications, such as IRDA, WLAN, and GPRS/UMTS. Among them Bluetooth [1] emerged as a low-cost, robust, and flexible short-range wireless network technology. Since it has been designed to achieve a low power consumption, it particularly fits the requirements of resource-constrained smartphones. Furthermore, it operates in a license-free frequency range, so that user is not charged for accessing the network nor needs an account with a third-part entity. The Bluetooth technology is used by many widespread different devices, such as mobile phones, smartphones, laptops, PDAs. A thorough overview on Bluetooth is given in [3] and [12].

Research studies have forecasted that within a few years, most of the devices accessing the Web will be mobile, and presumably most of them will be Bluetooth-enabled. Therefore, we need solutions that encompass networking, systems, and application issues involved in realizing mobile and ubiquitous access to services. In this paper, we analyze how to extend the possibility of accessing Web resources from Bluetooth-enabled smartphones.

Our goal is to provide a transparent middleware which allows user to access Web resources by using a Bluetooth connection. To this aim, we provide programmers with a lightweight and transparent tool to let their smartphone-targeted applications instaurate HTTP connections over a Bluetooth channel. In this way, the cost of communication is brought to zero, and the power-consumption is kept low. Furthermore, the transparency of the middleware allows programmers to ignore all the implementation details and to develop mobile applications in the standard and usual fashion.

Useful applications massively using HTTP connections are Web browsers, such as Opera Mini [4], the Web browser released by Opera Software [5]. Being written in Java, the Opera Mini browser is platform independent and can be easily deployed on every J2ME-powered smartphone. Whenever a user wants to surf the Internet, the application instaurates a HTTP connection over WAP, GPRS, or UMTS, which are the available protocols supporting TCP/IP. The use of our transparent middleware would allow Opera Software to release a version of Opera Mini which works on the free Bluetooth communication channel, without refactoring the source code.

Finally, our performance evaluations confirmed the real applicability and lightness of our middleware, showing that it is efficient enough to be used in a real world scenario for a wide set of applications.

This work has been partially supported by the European Commission through the IST program under contracts FP6-1596 (AEOLUS).
II. ENDORSED TECHNOLOGIES

In this section, we present all the technologies used in our work and we justify our design choices. The technologies on which our work relies are Java 2 Micro Edition (J2ME) and the Bluetooth Standard [1]. The former describes how to write Java applications on mobile devices, while the latter defines details for the communication between devices. Then, we give an overview of the networking management within J2ME. Finally, we present the API needed to use Bluetooth within J2ME, defined by the JSR-82 standard [11].

A. Bluetooth

The Bluetooth technology [1] allows communication to take place by mean of integrated and cheap devices with small energy consumption. This technology achieves its goal by embedding tiny, inexpensive, and short-range transceivers into the electronic devices that are available today. When two Bluetooth products come within their ranges, address and capability details can be automatically exchanged. Bluetooth devices operate in a licence-free frequency (starting from 2.4 GHz). Using version 1.2, one can establish a 1 Mbps link (a 2 Mbps link is supported by Version 2.0) [1]. Moreover security and error support allows to ensure efficient and reliable connections even in environments with a strong presence of interferences, electromagnetic fields, i.e., electrosmog. The Bluetooth standard allows the creation of WPAN (Wireless Personal Area Networks) [7]. Bluetooth-enabled devices communicate among each other building and dynamically setting up the network. This is achieved through the use of SDP (Service Discovery Protocol), which allows Bluetooth devices to discover services exposed by the others, and to instaurate a real communication channel between each pair of devices. The protocol that provides data exchange is the Logical Link Control and Adaptation Layer Protocol (L2CAP). It handles multiplexing and segmentation, through the use of the PSM (Protocol and Service Multiplexing) and SAR (Segmentation And Reassembly). Group abstractions and Quality of Service (QoS) are supported as well. Higher level protocols are built over this basic layer, like RFCOMM or OBEX. Figure 1 presents an overview of the Bluetooth stack. RFCOMM provides emulation of serial connections, it supports framing and multiplexing and implements serial data exchange. OBEX is built on top of RFCOMM, to implement exchange of objects, such as files and vCards.

At the application level, Bluetooth profiles describe several scenarios where Bluetooth technology is responsible for transmission. Each scenario is described by a user model. The corresponding profile provides to the application a standard interface to interact with the Bluetooth protocols. For more details over the Bluetooth technology, we refer to [12].

B. J2ME

The J2ME (Java Platform Micro Edition) is a collection of Java APIs for developing applications targeted to resource-constrained devices such as PDAs and smartphones. Formally, J2ME is an abstract specification, however the term is frequently used to also refer to the runtime implementations. The advantages of using Java as programming language are the code portability and an increase of the mobile devices flexibility. In particular, it provides the support for deploying dedicated applications, named MIDlets. They allow programmers to increase available features and capabilities of mobile devices. Since the range of micro devices is so diversified and wide, J2ME was designed as a collection of configurations, where each configuration is tailored to a class of devices. Each configuration consists of a Java Virtual Machine and a collection of classes that provide a programming environment for the applications. Configurations are completed by profiles, which add classes to provide additional features suitable to a particular set of devices. J2ME defines two configurations: the Connected Limited Device Configuration (CLDC) [9] and the Connected Device Configuration (CDC) [8].

CDC is addressed to small, resource-constrained devices such as TV set-top boxes, auto telematics. It can add a graphical user interface and other functionalities; CLDC, instead, is addressed to devices with limited memory capacity. In this paper we restrict our attention to the CLDC configuration. CLDC is a low level specification that includes a set of APIs providing basic features for limited-configuration devices, such as smartphones and PDAs. Producers should add features to CLDC by providing new libraries and thus creating a Profile. The first profile proposed for CLDC was the MIDP (Mobile Information Device Profile) [10]. MIDP is a set of Java libraries that permits to create an application environment for mobile devices with limited resources. Here, limitations include: amount of available memory, computational power, network communications with strong latency, and low bandwidth. MIDP 1.0 specification was produced by MIDPEG (MIDP Expert Group), as part of the JSR-37 standardization effort; while, the MIDP 2.0 specification was released with the JSR-118 standardization effort. MIDP 2.0 devices have to meet the following requirements:

- Memory, 250 KB of non volatile memory for MIDP components, 8 KB for user data.
- Display, 96x54 resolution, 1-bit color depth, 1:1 aspect ratio.
- Networking, bidirectional and wireless communication,
limited bandwidth.

C. J2ME Networking

The J2ME has to support a large variety of mobile devices with different sizes and shapes, different networking capabilities, and I/O requirements. Therefore, networking management in J2ME should be both flexible and device specific. To this aim, the CLDC defines the Generic Connection Framework. Such a framework delineates the abstractions of the networking and file I/O, in order to support the largest variety of devices, while leaving device manufacturers to provide real implementations. Abstractions are defined as Java interfaces and the device manufacturers choose which one to implement.

Figure 2 shows the interface diagram of the javax.microedition.io package (CLDC). There is one class (Connector) and seven connection interfaces (Connection, ContentConnection, DatagramConnection, InputConnection, OutputConnection, StreamConnection, and StreamConnectionNotifier) defined in the Generic Connection framework. We remark that no implementation is given at the CLDC level. The actual implementation is left to MIDP.

![Diagram of javax.microedition.io package](image)

Fig. 2. javax.microedition.io class diagram

We remark that MIDP-powered devices must at least support HTTP connections. HTTP is the most used protocol and it is easily implemented over different wireless networks. The use of HTTP allows user to exploit server-side infrastructure which are available for cabled networks.

HTTP connection parameters can be set up by the following methods:

- `setRequestMethod(String method)`: choose GET, POST, OR HEAD operations.
- `setRequestProperty(String key, String value)`: sets up a generic request.

In Figure 3, we show an example of how to execute from a MIDP-powered device a simple HTTP post operation to a Java Servlet on a remote Web Server. We remark that the operating system is in charge of instaurating a physical connection. If more network interfaces are available, it selects a default one or asks user to choose one.

```java
StringBuffer messagebuffer = new StringBuffer();
/* Open up a http connection with the Web server for both send and receive operations */
HttpConnection hc = (HttpConnection) Connector.open(defaultURL, Connector.READ_WRITE);
/* Set the request method to POST */
hc.setRequestMethod(HttpConnection.POST);
/* Send the string entered by user byte by byte */
OutputStream dos = hc.openOutputStream();
byte[] request_body = requeststring.getBytes();
for (int i = 0; i < request_body.length; i++)
    dos.writeByte(request_body[i]);
dos.flush(); dos.close();
/* Retrieve the response back from the servlet */
InputStream dis = hc.openInputStream();
int ch;
long len = hc.getLength();
if (len!=-1) {
    for (int i = 0; i < len; i++)
        if ((ch = dis.read())!=-1)
            messagebuffer.append((char)ch);
} else { // if the content-length is not available
    while ((ch = dis.read()) != -1)
        messagebuffer.append((char) ch);
}
dis.close();
```

D. JSR-82

Although the synergy between MIDP and J2ME technologies supplies a large number of communication schemes, it does not provide support for the Bluetooth technology. Therefore, the Java Expert Group JSR-82 [11] introduced the Java APIs for Bluetooth Wireless Technology (JABWT) that provides a standard and high-level support for handling Bluetooth communications in Java applications. These APIs operate on top of CLDC to extend MIDP functionalities. Their development is still in progress, but about twenty mobile vendors have adopted them in their devices. The last released version (Version 1.1) provides support for:

- Data transmission on the Bluetooth channel (audio and video are not supported).
- Protocols: L2CAP, RFCOMM, SDP, OBEX.
- Profiles: GAP, SDAP, SPP, GOEP.
The interaction between J2ME environment and the Bluetooth API is showed in Figure 4. Using JABWT, it is possible to interact with the Bluetooth stack in a Java application. In particular, it is possible to call services such as device and service discovery, establishment of RFCOMM, L2CAP, and OBEX connections.

The JSR-82 API is implemented to work within the Generic Connection Framework and follows the same strategy of connections presented in the previous section.

III. OUR MIDDLEWARE

In this section we present the architecture of our client-server middleware, targeted to establish HTTP connections over a Bluetooth channel. The entities involved are:

- Client Bluetooth (CBT): it is a J2ME-powered mobile device, which sends and receives HTTP messages over a Bluetooth channel. On this side, all the work is performed by the BtHttpConnection interface.
- Web Server (WS): it is a standard Web Server (e.g. Apache) which responds to HTTP requests.
- The Bluetooth HTTP Server Proxy (BHSP): it interfaces the WS with the CBT, by listening to requests on the Bluetooth channel, forwarding them to the WS, and giving back results to the CBT.

The BHSP has been implemented as a Bluetooth listener daemon. It has been designed to operate at a low-level and to be lightweight (it has a small footprint) in order not to affect performance. It does not interpret processed data but it simply forwards it.

Whenever an incoming request is received from the Bluetooth channel, it is inserted in a queue and processed as soon as possible. The BHSP uses the Apache Commons HttpClient package [2] to execute the required method on the WS. Once that it has got the response, it forwards it back on the Bluetooth channel to the CBT.

We remark that the BHSP has been designed as a supplementary module of the WS (i.e., it is a daemon starting together with the WS), conceptually allowing the WS to accept requests from a Bluetooth channel, too. Within this scenario, a Web Server administrator can decide to provide its resources not only through the Internet, but also to Bluetooth-enabled mobile devices which are within its transmission range. This Web server will be listening upon port 80 (the standard HTTP port) and upon Bluetooth RFCOMM port, see Figure 5(a). Moreover, our middleware fits another scenario as well. In fact, the BHSP can act as a real proxy and be implemented over any device with two interfaces: a Bluetooth one to interact with the CBT and any other interface supporting TCP/IP that can interact with the WS. In this way, CBT and WS are not required to be within transmission range, see Figure 5(b). In

![Fig. 4. J2ME - Bluetooth API interaction architecture](image)

![Fig. 5. Two scenarios for our middleware](image)

the scenario depicted by Figure 5(a), the BHSP will post the received request to localhost, while in the scenario in Figure 5(b) it will post the request to the domain sent by the CBT. We remark that in the first case requests are bounded to the WS in the transmission range, while in the second one they can address any WS reachable from the BHSP.

Figure 6 shows how the middleware works. The top section of the diagram depicts the association of the BHSP’s Bluetooth address to the http:// URL, performed by the BtHttpConnection interface through an inquiry. The bottom section shows a generic operation over the instaurated connection: data is exchanged between the client and the BHSP over the RFCOMM channel; requests to the WS are posted by the BHSP through the use of Apache HttpClient package [2].

A. The BtHttpConnection interface

As discussed in Section II-C, the connection string given as parameter in the Connector.open polymorphically drives the type of the object that will be returned, according to the cast made by the developer.

Our goal is to provide a new interface in the Connection hierarchy which provides HTTP support over Bluetooth. We named this interface BtHttpConnection and we implemented by extending the HttpConnection interface. As a result, we can invoke the Connector.open method with a
HTTP-based connection string and decide to cast the generic Connection object returned either as a HttpConnection or a BtHttpConnection object. In this case, operations will be carried out over a Bluetooth channel.

Whenever a BtHttpConnection is instantiated by an invocation of the Connector.open method, the following operations are performed:

- Establish an association between the “http”-starting URL (given in the connection string) and a Bluetooth remote device (the BHSP).
- If the association has not been previously established, perform an inquiry operation to discover a Bluetooth device which exposes a “Web Server” service and which corresponds to the URL in the connection string. Afterwards, store the association in a local database.
- If the association has been previously established, recover the correspondent Bluetooth address from the local database.
- Once the Bluetooth Address has been found, establish a RFCOMM connection with the BHSP, which will be used to send/receive HTTP requests/responses.

We remark that the extra work required to implement HTTP connections over Bluetooth is totally transparent to application developers. In fact, BtHttpConnection provides programmers with the same interface as HttpURLConnection, masking all the implementation details of the Bluetooth interactions. Indeed, suppose that a programmer has implemented the MIDlet showed in Figure 3 to performs a HTTP post to a servlet using WLAN from his handheld. If he wants to deploy its application on cheaper smartphones, which do not support WLAN but do support Bluetooth, then he has only to cast to BtHttpConnection the object returned from the Connector.open method invocation (see Figure 7).

```java
StringBuffer messagebuffer = new StringBuffer();
/* Open up a http connection with the Web server for both send and receive operations */
BtHttpConnection bhc = (BtHttpConnection) Connector.open(
    defaultURL, Connector.READ_WRITE);
/* Set the request method to POST */
bhc.setRequestMethod(HttpConnection.POST);
/* Send the string entered by user byte by byte */
byte[] request body = request string.getBytes();
for (int i = 0; i < request body.length; i++)
    dos.writeByte(request body[i]);
dos.flush();
dos.close();
/* Retrieve the response back from the servlet */
DataInputStream dis =
    new DataInputStream(bhc.openInputStream());
int ch;
long len = bhc.getLength();
if (len!=-1) {
    for (int i = 0; i < len; i++)
        if ((ch = dis.read()) != -1)
            messagebuffer.append((char) ch);
} else // if the content-length is not available
    while ((ch = dis.read()) != -1)
        messagebuffer.append((char) ch);
dis.close();
```

Fig. 7. A simple HTTP post operation performed over a Bluetooth channel.

The same mechanism would allow the Opera Mini J2ME Web browser presented in Section I to operate over Bluetooth, without refactoring the source code.
IV. Performance Evaluation

In this section, we analyze the performance of our middleware in order to evaluate its lightness and its usability in real world scenarios. To this aim we set up the following test bed:

- The WS and the BHSP lie on a PC IBM ThinkCentre 50 Personal Computer, with Pentium 4 2.6 GHz and 760 MB RAM, running Windows XP Professional SP 2, with a Bluetooth TrendNet TBW-102UB USB dongle, and BlueCove [6] implementation of the JSR-82 Bluetooth API for Java.
- For the CBT we have used a Nokia N73 mobile phone running Symbian OS 9.1, compliant with MIDP 2.0 and JSR-82 standards.

We evaluated the overhead taken by our middleware to let a mobile client interact with a Web Server using HTTP connections over a Bluetooth channel. To this aim, we have compared times to post growing size strings for the following applications:

1) A MIDlet which posts strings to a Bluetooth-enabled Web Service using the BtHttpInterface.
2) A simple MIDlet which sends strings to a remote device over Bluetooth.

Figure 8 shows times for strings ranging from 1 KB to 50 KB with an increasing size of 0.5 KB. Each test was repeated 50 times to get significant average times. As we can see in the diagram, the use of BtHttpConnection slightly slows down the computation. The other application can directly access the Bluetooth channel and send data over it, while the BtHttpConnection implies a small computation overhead to handle the connection and to give a higher profile to the application. However, the overhead is almost constant and small enough to consider our middleware efficient enough to be used in real world scenario with complex applications.

![Fig. 8. BtHttpConnection evaluation](image)

**REFERENCES**