HTTP over Bluetooth: a J2ME experience *

Vincenzo Auletta, Carlo Blundo
Dipartimento di Informatica ed Applicazioni
Università degli Studi di Salerno
I-84084 Fisciano (SA) - Italy
{auletta,carblu}@dia.unisa.it

Emiliano De Cristofaro
Information and Computer Science
University of California Irvine
Irvine, CA, 92617 - USA
edecrist@uci.edu

Abstract— **Over the last years, computation and networking have been increasingly embedded into the environment. This tendency 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 neither 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 lightweight solution to extend the possibility of accessing Web resources also from Bluetooth-enabled mobile phones. 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.

Index Terms— **Ubiquitous Computing, HTTP, Bluetooth, Mobile Phones, J2ME.**

I. Introduction

The evolution of technology has led to a deep transformation of users habits, with an increasing requirement of support for mobility and connectivity. Furthermore, nowadays a brand new set of applications fit mobile environments and allow device interactions over wireless channels. Today's mobile phones are small, powerful, and usable enough to be fundamental working instruments, and to be considered for the deployment of complex applications.

Modern applications, however, require connectivity and thus a critical issue for the diffusion of mobile devices is the capacity to run network applications, especially Web applications. In the last years, several new protocols have been presented for wireless communications, such as IRDA, WLAN, and GPRS/UMTS. However, IRDA connections are limited to two devices with a direct line of sight, and thus IRDA is not practically useful for a real intercommunication scheme. WLAN instead has been designed as a powerful technology to support multipoint connections, but diffusion of WLAN on mobile devices and particularly on mobile phones is still low. GPRS/UMTS are widely supported but they provide connectivity at modest speed and requires a personal account with a phone company. At the same time, we witnessed the growth of Bluetooth, that is a low-cost, robust, powerful, and flexible short-range wireless network technology with low power consumption [4]. It operates in a license-free frequency range, so that user is not charged for accessing the network nor needs an account with any company, thus allowing a relevant decrease of communication costs. Nowadays, the evolution of Bluetooth technology is driven by the Bluetooth SIG, that consists of over 7000 member companies that guarantee a large support to this technology. In fact, Bluetooth technology is used in many widespread different devices, such as handhelds, mobile phones, smartphones, laptops, PDAs. A thorough overview on Bluetooth is given in [6] and [24].

A recent study has pointed out that the number of users accessing the web from a mobile device has overtaken the one using a "standard" terminal [9]. 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 mobile phones.

Our goal is to provide a transparent middleware which allows user to access Web resources by using a Bluetooth connection. In particular, we provide application developers with a lightweight solution to let their mobile applications establish 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, we have in mind a transparent middleware allowing programmers to ignore the implementation details related to the underlying Bluetooth channel.

Common applications massively using HTTP connections are Web browsers, e.g. Opera Mini [18] – the Web browsed released by Opera Software [19]. Being developed in Java, the Opera Mini browser is platform independent and can be easily deployed on every J2ME-powered mobile phone. 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. Provided

*A preliminary version of this work has been published in ICSNC 2007 [23]

that a Bluetooth connection is available to a device acting as a gateway to the Internet, 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.

Paper Organization. The rest of the paper is organized as follows. In Section II, we present the endorsed technologies, i.e. Bluetooth, J2ME, and the JSR-82 APIs. In Section III, we give an overview of our solution to allow J2ME application developers to establish HTTP connections using Bluetooth as the communication channel. Then, Sections IV and V present the details of our implementation respectively for the client-side and the server-side. Subsequently, Section VI discusses the transparency of our solution and presents some application scenarios. Finally, Section VII briefly evaluates the performance overhead.

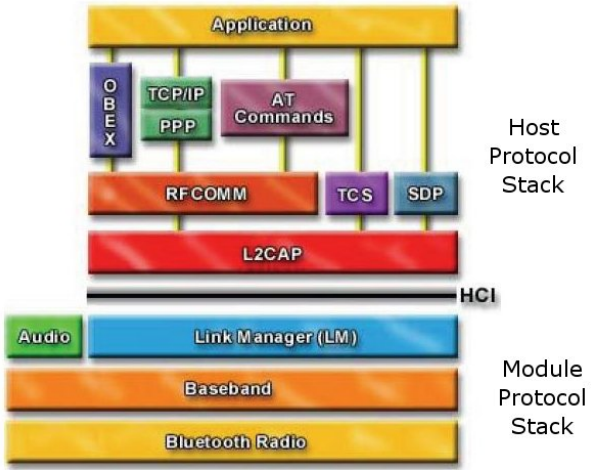

Fig. 1. The Bluetooth Stack [26].

II. Endorsed Technologies

In this section, we present all the technologies on which our work relies: the Bluetooth Standard [4] and the J2ME [11]. The former defines details for the communication between devices, while the latter describes how to write Java applications on mobile 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 [15].

A. The Bluetooth Wireless technology

Bluetooth specification was introduced in 1994 by Ericsson to provide radio communications between mobile phones, headsets and keyboards. The specifications were then formalized by the Bluetooth Special Interest Group (SIG) [4] in 1998. Within this technology, radio communications can take place by means of integrated and cheap devices with small energy consumption. This is achieved by embedding tiny, inexpensive, short-range transceivers into electronic devices that are available nowadays. The Bluetooth standard defines the following requirements:

- The system must operate globally, and the required frequency band must be license-free and open to any radio system.
- The system must provide peer connections.
- The connection must support both voice and data.
- The radio transceiver must be small and operate at low power.

Bluetooth devices operate in a license-free frequency range (starting from 2,4 GHz). The available bandwidth is divided into 79 channels. In version 1.2 one can establish a 1 Mbps link (a 2 Mbps link is supported by Version 2.0) [6]. Moreover, security and error support allow to assure efficient and reliable connections even in environments with a strong presence of interferences and electromagnetic fields.

Bluetooth-enabled devices can dynamically *discover* other devices in their range and their supported services, through an inquiry process. A *Piconet*, consisting of one *master* device and up to seven *slave* devices, will be settled once the peer connections have been established. Piconets can be interconnected to form a *Scatternet*.

An overview of the Bluetooth stack is presented in Figure 1. The *radio* level is the lowest one and defines the technical details of the communication. Bluetooth adopts the FHSS (Frequency Hopping Spread Spectrum), making 1600 hops per second; thus each physical channel is occupied for 625μs. These intervals are referred to as slots and they are numbered sequentially. Frequency hopping occurs by jumping from one physical channel to another in a pseudorandom sequence. The *baseband* layer handles channels and physical links, providing services such as error correction and security. It supports multipoint communications through FH/TDMA (Frequency Hopping/Time Division Multiple Access). The master device is in charge of defining the hopping sequence to all the slave devices. A physical channel is shared between the master and a slave using a time division scheme in which data are transmitted in one direction at time, with transmissions alternating between the two directions: the master transmits in even slots; slaves transmit in odd slots and may hold the transmission for 1, 3, or 5 consecutive slots. Links between master and slaves can be either *Synchronous Connection Oriented* (SCO) or *Asynchronous Connection-Less* (ACL). The first type of link is used in real-time applications and it allows a symmetric point-to-point communication achieving 64 Kbps transmission rate. The second is used for asymmetric transmission between master and slaves with 723 Kbps for downlink and 57 Kbps for uplink. Bluetooth standard defines two different types of packets for ACL links: Data Medium Rate (DM) which provides a 2/3 FEC Hamming code (i.e., error correcting capabilities), and Data High Rate (DH) which provides no FEC coding (i.e., no error correction at all). Therefore, we have six different data packets according to the slots assignment and data encoding: DH1, DH3, DH5, DM1, DM3, and DM5 (digits denote the number of occupied slots). Up in the stack we find: the *Link Management Protocol* (LMP) handling link setup, authentication, and link configuration; the *Host Controller Interface* (HCI) which provides a uniform

method of accessing the Bluetooth baseband capabilities; the *Logical Link Control and Adaptation Protocol* (L2CAP) which deals with data multiplexing and segmentation. Finally, on top of L2CAP, we find several data communication protocols. The main protocols are:

- SDP (Service Discovery Protocol), which handles the discovery of devices and services within the device's transmission range.
- RFCOMM, which implements emulation of serial connections, setting up point-to-point connections. It supports framing and multiplexing and achieves all the required functions for serial data exchange.
- OBEX (Object Exchange), which is built on the top of RFCOMM to implement exchange of objects, such as files and vCards. Originally, it was developed by IrDA (Infrared Data Association) for IR-enabled devices.
- TCS (Telephony Control protocol Specification), which defines ways to send audio calls between Bluetooth devices.

The Bluetooth technology is also composed by a set of profiles. Bluetooth profiles describe several scenarios where Bluetooth technology is responsible of transmission. Each scenario is described by a user model and the corresponding profile gives a standard interface that applications can use to interact with the Bluetooth protocols. The profile concept is used to decrease the risk of interoperability problems between different manufacturers' products, for instance, some profiles are:

- FTP (File Transfer Profile), which defines how folders and files on a server device can be browsed by a client device.
- HPF (Hands-Free Profile), which describes how a gateway device can be used to place and receive calls for a hand-free device.
- VDP (Video Distribution Profile), which defines how a Bluetooth enabled device streams video over Bluetooth wireless technology.

Other details on the Bluetooth specification can be found in [4] or in [27].

In order to interface applications to the physical layer a Bluetooth Stack implementation is necessary. The stack provides a standard interface between the application layer and the Bluetooth specification. This interface is used to overcome the compatibility problems between application and different Bluetooth devices. Indeed, Bluetooth stacks are responsible of implementing the Bluetooth wireless standards specifications. There are several different stacks targeted to different devices, applications, and operating systems. To our knowledge, currently available Bluetooth stack implementations are:

- Mobile devices vendors' embedded stacks. Vendors providing Bluetooth-enabled devices have to build their own Bluetooth stack; for mobile phones stack implementations depend on the OS (e.g., Symbian).
- Broadcom BTW (not free) [8]. It is addressed to PC OEMs and accessory manufactures to quickly and easily add Bluetooth technology to desktop PC and notebooks running Windows. It includes the object code for the Protocol Stack (L2CAP, SDP, RFCOMM, OBEX, PPP, BTM-Bluetooth Manager), an application programming interfaces (APIs), and test tools.
- Microsoft BT Stack [20]. It is the Microsoft version of the Bluetooth stack and is embedded in Windows XP SP 2. It provides the support for most of Bluetooth profiles, essentially the ones based on the RFCOMM protocol.
- BlueZ (free and open-source) [7]. It is the official Linux Bluetooth Stack. The code is licensed under the GNU General Public License and is included in the Linux 2.4 and Linux 2.6 kernel series. It provides a direct access to the transmission layer and allows developers to set several parameters of the communication (i.e., choosing an ACL or SCO connection, choosing different time shifting, etc.).

B. J2ME

The J2ME (Java Platform Micro Edition) is a collection of Java APIs supporting the development of applications targeted to resource-constrained devices such as PDAs and mobile phones. Formally, J2ME is an abstract specification, however the term is frequently used also to refer to the runtime implementations. The advantages of using Java as programming language are the code portability and an increase of mobile devices' flexibility. In particular, J2ME provides support for deploying dedicated applications, named MIDlets. 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 then completed by profiles, which add classes to provide additional features suitable to a particular set of devices. J2ME defines two configurations: the *Connected Device Configuration* (CDC) [12] and the *Connected Limited Device Configuration* (CLDC) [13].

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, which is the most diffused one. CLDC is a low level specification that includes a set of APIs providing basic features for resource-constrained devices, such as mobile phones 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) [14]. MIDP is a set of Java libraries that allows 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 [17] standardization effort; while, the MIDP 2.0 specification was released with the JSR-118 [16] 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. Networking in J2ME

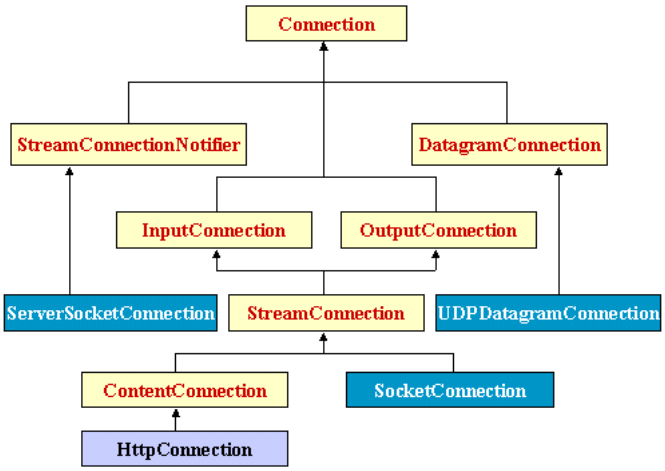

Fig. 2. The J2ME Connection hierarchy diagram.

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 manufactures to provide real implementations. Abstractions are defined as Java interfaces and the device manufacturers choose which one to implement.

Networking features within J2ME are defined by the MIDP in the `javax.microedition.io` package. It supports the following communication protocols:

- HTTP and HTTPS connections
- datagram
- socket
- secure socket
- serial port communication

Figure 2 shows the interface diagram of the `javax.microedition.io.Connection` hierarchy. Those interfaces are part of the Generic Connection Framework of J2ME's CLDC, together with the `Connector`) class. We remark that no implementation is given at the CLDC level. The actual implementation is left to MIDP. The `Connector` class is the core of the Generic Connection Framework. All connections are created by its static method `Connection open(String connect)`. Different connections are instantiated according to different connect strings. The connect string has a URL-like format:

PROTOCOL:[//TARGET][PARAMS],

where PROTOCOL defines the type of connection (e.g., file, socket, http); TARGET defines a hostname, a port number, or a file name; PARAMS defines optional parameters. Polymorphically, different parameters in the connection string make the `Connector.open` method return a different `Connection` object. For example, a connection string starting with *http://* will drive the `open` method to return a `HttpConnection` object.

We remark that MIDP-powered devices are required to support at least 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. The `HttpConnection` interface defines the MIDP API for HTTP. This interface extends another interface, `ContentConnection`, to add fields and methods required for: URL parsing, request management, response parsing.

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

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

In Figure IV, 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 establishing a physical connection. If more network interfaces are available, it selects a default one or asks user to choose one.

The Java code performs the following operations:

(1) Open a HTTP connection with the Web Server for both send and receive operations.
(2) Set the request method to POST
(3-6) Send the string entered by user byte by byte.
(7-8) Close the output stream.
(9) Open an InputStream on the connection.
(10-18) Retrieve the response back from the servlet.
(19) Close the input stream.

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 [15] introduced the *Java API for Bluetooth Wireless Technology* (JABWT) that provides a standard and high-level support for handling Bluetooth communications in Java applications. This API operates on top of CLDC to extend MIDP functionalities. Its development is still in progress, but about twenty mobile vendors have adopted it 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 Generic Access Profile (GAP) defines the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices. The

```
(1) HttpConnection hc = (HttpConnection) Connector.open(defaultURL, Connector.READ_WRITE);

(2) hc.setRequestMethod(HttpConnection.POST);

(3) DataOutputStream dos = hc.openDataOutputStream();
(4) byte[] request_body = requeststring.getBytes();

(5) for (int i = 0; i < request_body.length; i++)
(6)            dos.writeByte(request_body[i]);

(7) dos.flush();
(8) dos.close();

(9) DataInputStream dis = new DataInputStream(hc.openInputStream());

(10) int ch;
(11) long len = hc.getLength();

(12) if(len!=-1) {
(13)     for(int i = 0;i<len;i++)
(14)         if((ch = dis.read())!=-1)
(15)             messagebuffer.append((char)ch);
(16) } else { // if the content-length is not available
(17)     while ((ch = dis.read()) != -1)
(18)         messagebuffer.append((char) ch);
}

(19) dis.close();
```

Fig. 3. A simple HTTP post operation from a MIDlet

Service Discovery Application Profile (SDAP) defines the features and procedures for an application in a Bluetooth device to discover services registered in other Bluetooth devices and retrieve any desired available information pertinent to these services. The Serial Port Profile (SPP) defines the requirements for Bluetooth devices necessary for setting up emulated serial cable connections using RFCOMM between two peer devices. The Generic Object Exchange Profile (GOEP) defines the requirements for Bluetooth devices necessary for the support of the object exchange usage models.

The interaction between the J2ME environment and the Bluetooth API is shown in Figure 4.

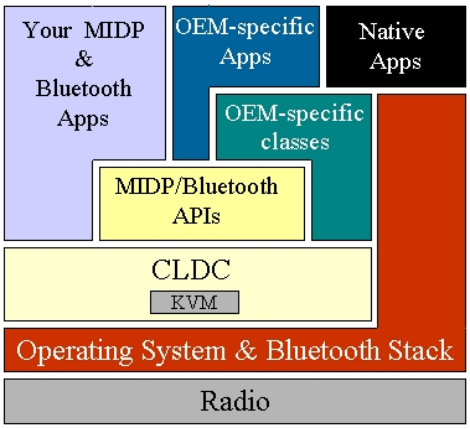

Fig. 4. J2ME - Bluetooth API interaction architecture [25].

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.

In order to use the Java APIs for Bluetooth, a real implementation of the JSR-82 specification is necessary on the device. To our knowledge, the current JSR-82 implementations are:

- Mobile devices vendors' embedded JSR-82 implementations.
- Atinav aveLink suite (not free) [5]. It offers both a standard implementation of the Bluetooth stack and the implementation of all the standard profiles for ANSI C, JSR-82 for J2SE Java, JSR-82 for J2ME, Windows and Windows CE.
- Impronto Rococo (not free) [10]. It is a complete product that provides the Bluetooth Stack and the integration layer, the JVM and the JSR-82 implementation layer both for J2SE and J2ME.
- Avetana (not free) [2]. It enables writing J2SE applications to access the Bluetooth layer; it is available for Windows, MacOS X, and Linux platforms.
- BlueCove (free) [3]. It provides the Java JSR-82 support for J2SE applications over the Windows XP SP2 Bluetooth stack.

III. HTTP Connections over Bluetooth

In this section, we will present our solution to allow a J2ME application developer to establish HTTP connections using Bluetooth as the communication channel.

Since a few years, Bluetooth has been exploited to connect a PC to the Internet through Internet connections available on a paired mobile phone, such as cell network service (e.g. GPRS, EDGE, UMTS) or WLAN. In this scenario, the mobile phone acts as a gateway, while the communications between the phone and the PC are carried over Bluetooth.

However, the cell network internet service is often expensive or not always available. Also, mobile phones supporting WLAN are still in a minority of the market and are available only on high-end cost phones. In this work, we want to provide HTTP connections to mobile phone users without requiring them to use the cell network Internet connection, i.e. using a Bluetooth connection.

Therefore, we refer to the scenario shown in Figure 5, where a client establishes a HTTP connection with a server.

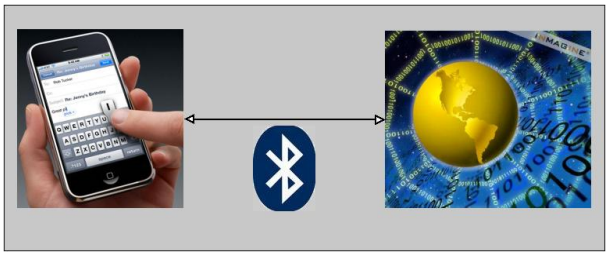

Fig. 5. Ideal scenario: a mobile phone accesses the Web using Bluetooth.

We observe that wireless communication by means of HTTP over GPRS/EDGE/UMTS and WLAN is widely supported within J2ME. On these channels it is easy to create a HTTP connection and deploy MIDlets that network over this channel by means of `HttpConnection` objects. However, to the best of our knowledge, there is no implemented support for creating a `HttpConnection` object which uses an underlying Bluetooth channel. To overcome this limitation, we created ourselves the BtHttpConnection class and introduced a new entity, the `BHSP` (Bluetooth Http Server Proxy), that takes care of interfacing clients to the Web Server. We argue that we can maintain the same server-side architecture and guarantee the interoperability of applications running on the mobile device with any Web Server. Moreover, no modification to the MIDlet is required to use Bluetooth as communication channel, instead of the previously supported channels, i.e. WLAN or GPRS/EDGE/UMTS. Figure 6 illustrates the operational scenario to which we refer.

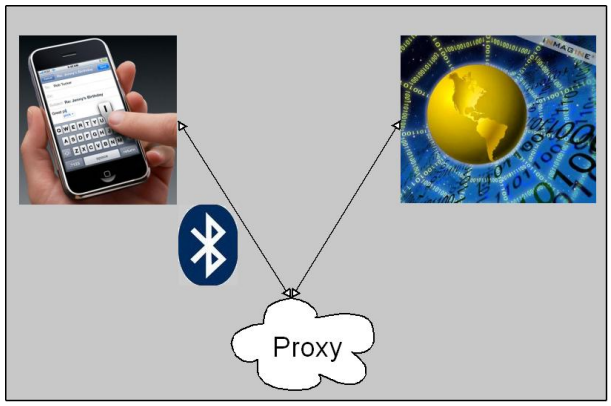

Fig. 6. Operational scenario of our solution.

The resulting framework is then composed of four different entities:

- Client Application. It runs on a Bluetooth-enabled mobile device and it establishes HTTP connections on a Bluetooth channel using the J2ME standard way, i.e. using the `HttpConnection` class.
- `BtHttpConnection` interface performs all the work required to communicate on the Bluetooth channel and to achieved the transparency for the client application's developer.
- `BHSP` - Bluetooth Http Server Proxy. It interfaces clients with Web Servers, by listening to requests on the Bluetooth channel, forwarding them to the Web Server, and giving back results to the client application.
- Web Server. It is a standard Web Server (e.g. Apache) that replies to clients' requests communicating through the `BHSP`.

We remark that the `BHSP` has been designed as a supplementary module of the Web Server (i.e., it is a daemon starting together with the Web Server), conceptually allowing the Web Server 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 7(a). Moreover, our solution fits another scenario as well. Indeed, the `BHSP` can act as a real proxy and be implemented over any device equipped with two interfaces: (i) Bluetooth, used to interact with the client application, and, (ii) an interface supporting TCP/IP, that can interact with the Web Server. In this way, the client application and the Web Server are not required to be within transmission range, see Figure 7(b).

In the scenario depicted by Figure 7(a), the `BHSP` will post the received request to *localhost*, while in the scenario in Figure 7(b) it will post the request to the *domain* request by the client application. We remark that in the first case requests are bounded to the Web Server in the transmission range, while in the second one they can address any Web Server reachable from the `BHSP`.

IV. The BtHttpConnection Class

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. However, in order to do that we should modify the source code of the `Connector.open` method. Although the whole J2ME environment has recently gone open source, this modification should then be reflected in all the J2ME implementations by phones' vendors. Therefore, although the `BtHttpConnection` class extends the `HttpConnection` class, we cannot polymorphically cast the `HttpConnection` object to `BtHttpConnection`. Hence, we let the `Connector.open` method still return a `HttpConnection` object, but then we instantiate a `BtHttpConnection` method which takes in input all the information about the `HttpConnection` object:

Whenever a `BtHttpConnection` is instantiated, the following operations are performed:

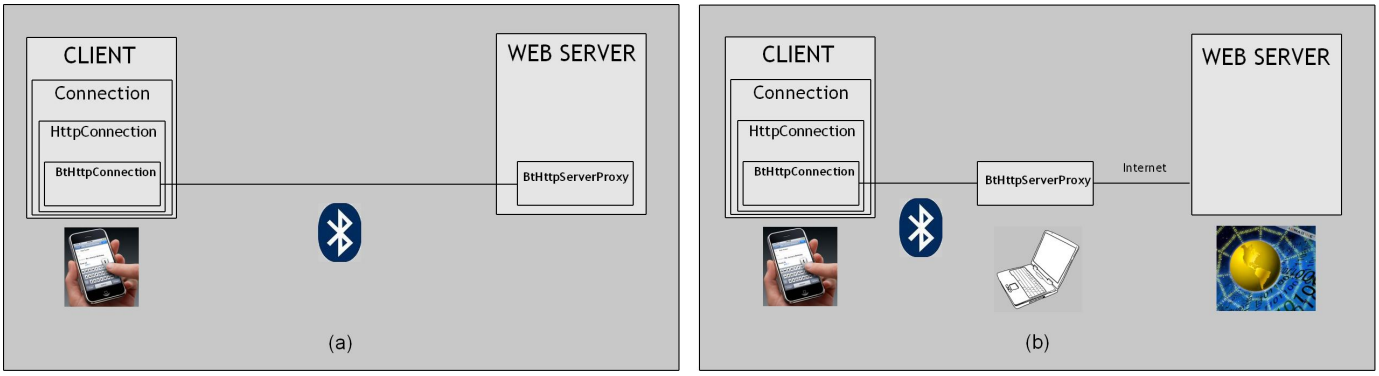

Fig. 7. Two possible settings for the BHSP.

```
(1) HttpConnection hc =(BtHttpConnection)
       Connector.open(defaultURL, Connector.READ_WRITE);

(2) BtHttpConnection bhc =new BtHttpConnection(hc);
```

Fig. 8. From a HttpConnection to BtHttpConnection

- 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.

Code reuse. 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 `HttpConnection`, masking all the implementation details of the Bluetooth interactions. Suppose that a programmer has implemented the MIDlet showed in Figure IV to performs a HTTP post to a servlet using WLAN from his mobile phone. If he wants to deploy its application on cheaper mobile phones, not supporting WLAN but supporting Bluetooth, then he has only to use the `BtHttpConnection` instead of `HttpConnection`. We stress that all the methods are unaltered, so no modification to the code is required. The operation is showed in Figure 9, which differs from only for the line code (2).

Figure 10 shows the resulting new class diagram with the new `BtHttpConnection` class to extend the `HttpConnection` class.

V. BHSP: The Bluetooth Http Server Proxy

The `BHSP` has been implemented as a Bluetooth listener daemon. It will run as a J2SE application on a desktop

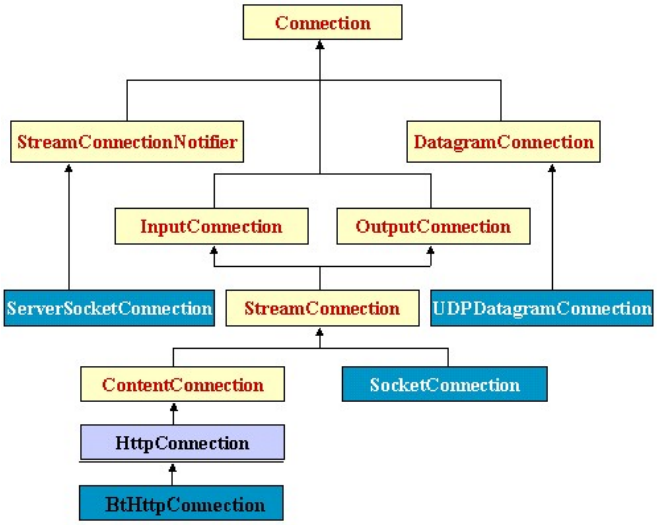

Fig. 10. The J2ME Connection hierarchy diagram

computer but 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 [1] to execute the required method on the Web Server. Once that it has got the response, it forwards it back on the Bluetooth channel to the client application.

The work is performed by a `BTServer` class, which takes care of:

- setting the device in *discoverable* mode
- activating a listening connection
- accepting incoming connections
- performing I/O on the Bluetooth channel
- instantiate a `Poster` object.

The `Poster` class is in charge of:

- performing an HTTP post operation on the Web Server, using the Apache *Commons HttpClient* package
- giving back the request to the `BTServer` object

In order to have a licence-free and JSR-82 compliant implementation of our `BHSP`, we have considered two alternatives:

```
(1) HttpConnection hc =(BtHttpConnection) Connector.open(defaultURL, Connector.READ_WRITE);

(2) BtHttpConnection bhc = new BtHttpConnection(hc);

(3) bhc.setRequestMethod(HttpConnection.POST);

(4) DataOutputStream dos = bhc.openDataOutputStream();
(5) byte[] request body = requeststring.getBytes();
(6) for (int i = 0; i <; request body.length; i++)
(7)    dos.writeByte(request body[i]);

(8) dos.flush();
(9) dos.close();

(10) DataInputStream dis =  new DataInputStream(bhc.openInputStream());

(11) int ch;
(12) long len = bhc.getLength();
(13) if(len!=-1) {
(14)     for(int i = 0;i<len;i++)
(15)         if((ch = dis.read())!=-1)
(16)             messagebuffer.append((char)ch);
(17) } else { // if the content-length is not available
(18)     while ((ch = dis.read()) != -1)
(19)        messagebuffer.append(10) ;
}

(20) dis.close();
```

Fig. 9. A simple HTTP post operation performed over a Bluetooth channel

1) To use BlueCove [3], the free implementation of the JSR-82 API within the Microsoft Windows XP SP2.
2) To use BlueZ [7], the Linux Bluetooth stack, and provide a JSR-82 implementation for BlueZ.

We remark that for this part we have modified the implementations of the works in [21] and [22] that performed a similar operations for Web Services invocation over Bluetooth.

Figure 11 shows the main steps of the `BHSP`. The Java code in Figure 11 executes the following operations:

(1) Set the device in discoverable mode.
(2-4) Activate a listening connection on localhost, on the channel 1, named "rfcomm test".
(5) Accept incoming connections.
(6) Open an InputStream on the connection.
(7-8) Read data on the stream.
(9-10) Post the HTTP request at the specified address, using the `Poster` class, to get the response.
(11) Open an OutputStream on the connection.
(12) Write data, i.e. the HTTP response.

VI. A TRANSPARENT AND EFFICIENT SOLUTION

In Figure 12, we summarize how our solution 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 established connection: data is exchanged between the client and the `BHSP` over the Bluetooth channel; requests to the Web Server are posted by the `BHSP` through the use of Apache *HttpClient* package [1].

We stress that the work needed to use a Bluetooth connection is totally transparent to both the application developers and the user. In fact, `BtHttpConnection` provides programmers with the same interface as `HttpConnection`, the only change required w.r.p. a normal HttpConnection-based MIDlet is to use the `BtHttpConnection` instead of `HttpConnection`.

We envision several applications for our solution. For instance, there could be waiting rooms, such as stations or airports, that provide a free Internet access to users, for timetable information, emails, weather forecasts. The same scenario could take place in trains, buses, coffee shops, restaurants. No massive money investment is required for this goal, other than exposing a BHSP (or more according to the expected number of users), to which users can connect with their simple J2ME- and Bluetooth-enabled mobile phone.

Also, we envision a scenario where Sun embeds our solution in the official J2ME specification, so that all the implementations will provide the support for HTTP connections over Bluetooth. The only requirement would be to have a BHSP proxy available in order to support the communication. We remark that our solution does not work on the TCP/IP protocol, but only allows simple POST/GET operations on Web Servers. As a result, there would be no support for congestion control, sessions, and all the other nice features provided by the protocol stack. However, we argue that Bluetooth communications, though as not reliable as TCP/IP ones, provides completely free communications on tiny and inexpensive devices. Moreover, we argue that the new Bluetooth technology (2.0) implemented by the new generation's chips is efficient enough to support simple HTTP operations, such as chats or Internet browsing on most of the web sites normally accessed by mobile phones' users..

Furthermore, we claim that our work fills an important gap of the J2ME environment. In spite of limitations such as communication speed or the necessary presence of a Bluetooth proxy, our solution finally gives the appropriate tool to application developers to deploy complex applications which work on Bluetooth. Indeed, we argue that this free technology could be exploited for many solutions, and not only for simple files exchange anymore. Thanks to its transparency, our solution is ready-to-use for real scenarios on many low-end price class of devices massively available and spread out today.

```
(1)  (LocalDevice.getLocalDevice).setDiscoverable(DiscoveryAgent.GIAC);

(2)  (StreamConnection) notifier = (StreamConnection)
(3)           Connector.open("btspp://localhost:1;name=rfcommtest;master=true;encrypt=false;authorize=false;
(4)           authentication=false;receiveMTU=512;transmitMTU=512");

(5)  notifier.acceptAndOpen();
(6)  InputStream input = notifier.openInputStream();

(7)  /* Perform buffered readings to get the request */
(8)  String request = input.read();
(9)  Poster poster = new Poster(address);
(10) String response = poster.doPost(request);
(11) OutputStream output = notifier.openOutputStream();
(12) output.write(response.getBytes());
```

Fig. 11. Java code for the BHSP.

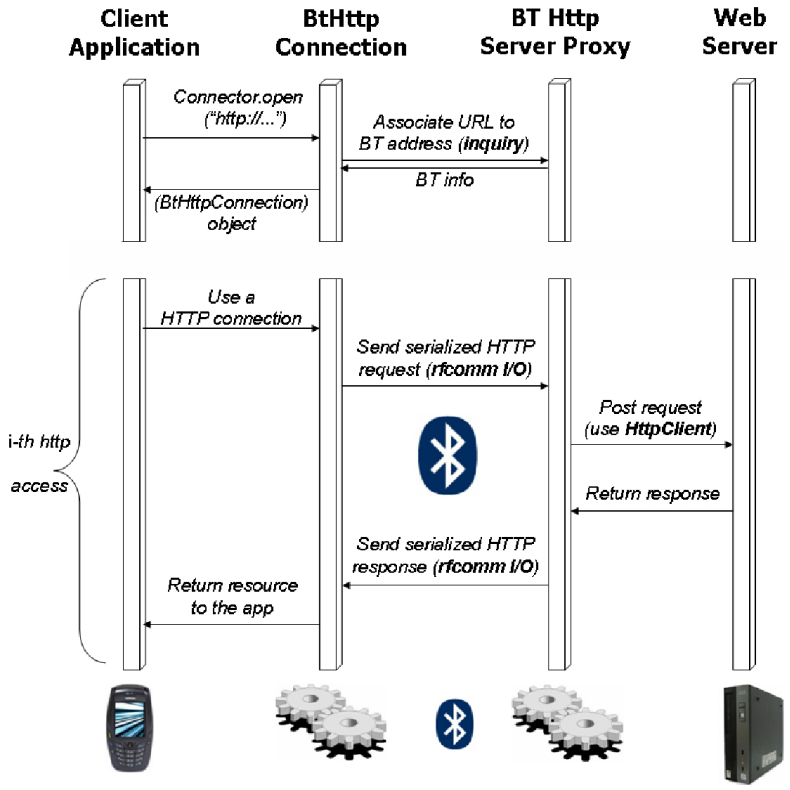

Fig. 12. Time Diagram of a client application accessing a Web resource.

VII. Performance Evaluation

In this section, we analyze the performance of our solution 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 [3] implementation of the JSR-82 Bluetooth API for Java.
- The client application runs on a Nokia N73 mobile phone (Symbian OS 9.1), compliant with MIDP 2.0 and JSR-82 standards.

We evaluated the overhead taken by the `BtHttpConnection` class 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 `BtHttpConnection`.
2) A simple MIDlet which sends strings to a remote device over Bluetooth.

Figure 13 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 solution efficient enough to be used in real world scenario with complex applications.

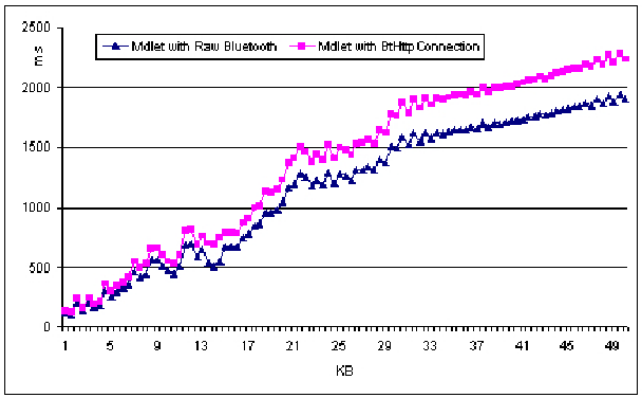

Fig. 13. Performance evaluation of `BtHttpConnection`.

VIII. Conclusion

We have presented a solution to establish HTTP connections over Bluetooth channels from low-end price class J2ME-enabled mobile phones. The resulting solution is lightweight and works at no extra cost for users and application developers, but it only requires the presence of a Bluetooth connection to a device connected to the Internet. The provided implementation requires no code modification and allows programmers to enhance the features of HTTP-based MIDlets with basically no effort, extending their range of action from mobile phones provided with GPRS/UMTS connections (sometimes expensive or not available) or WLAN access (available only on high-end price cost devices), but also to inexpensive mobile phones provided with Bluetooth interface, which is free to use.

Our performance evaluation confirms the real applicability and lightness of our solution, showing that it is efficient enough to be used in a real world scenario for a wide set of applications.

IX. Acknowledgements

This work has been partially supported by the European Commission through the IST program under contracts FP6-1596 (AEOLUS) and by the *Foundations of Adaptive Networked Societies of Tiny Artifacts* project, funded by the European Commission as project number 215270.

References

[1] Apache Commons HttpClient. `http://jakarta.apache.org/commons/httpclient/`.
[2] Avetana jsr-82 implementation. `http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml`.
[3] BlueCove. `http://sourceforge.net/projects/bluecove/`.
[4] The Bluetooth SIG Standard. `http://www.bluetooth.com`.
[5] Bluetooth solutions by Atinav AveLink. `http://www.avelink.com/bluetooth/index.htm`.
[6] Bluetooth Wireless Technology. `http://www.ericsson.com/technology/techarticles/Bluetooth.shtml`.
[7] BlueZ: Official Linux Bluetooth protocol stack. `http://www.bluez.org/`.
[8] Broadcom Bluetooth Solutions. `http://www.broadcom.com/`.
[9] Critical Mass – The Worldwide State of the Mobile Web . `http://www.nielsenmobile.com/documents/CriticalMass.pdf`.
[10] Impronto Rococo Software. `http://www.rococosoft.com/`.
[11] J2ME: Java 2 Micro Edition. `http://java.sun.com/j2me/`.
[12] JSR 218, Connected Device Configuration(CDC).
[13] JSR 30, JSR 139: Connected Limited Device Configuration (CLDC). `http://java.sun.com/products/cldc/`.
[14] JSR 37, JSR 118: Mobile Information Device Profile (MIDP). `http://java.sun.com/products/midp/`.
[15] JSR 82: Java APIs for Bluetooth. `http://www.jcp.org/en/jsr/detail?id=82`.
[16] Mobile Information Device Profile 2.0 (MIDP 2.0): JSR 118. `http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html`.
[17] Mobile Information Device Profile (MIDP): JSR 37. `http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html`.
[18] Opera Mini Web Browser. `http://www.operamini.com/`.
[19] Opera Software Company. `http://www.opera.com/company/`.
[20] Windows support for Bluetooth. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/bluetooth/bluetooth/about_bluetooth.asp`.
[21] V. Auletta, C. Blundo, E. D. Cristofaro, and G. Raimato. A Lightweight Framework for Web Services Invocation over Bluetooth. pages 331–338, 2006.
[22] V. Auletta, C. Blundo, E. D. Cristofaro, and G. Raimato. Performance evaluation of web services invocation over Bluetooth. pages 1–8, 2006.
[23] V. Auletta, C. Blundo, and E. De Cristofaro. A J2ME transparent middleware to support HTTP connections over Bluetooth. In *Proceedings of the Second International Conference on Systems and Networks Communications (ICSNC 2007)*, 2007.
[24] B. Chatschik. An overview of the Bluetooth wireless tecnology. *IEEE Communication Magazine*, 39:86–94, 2001.
[25] Q. H. Mahmoud. The Java APIs for Bluetooth Wireless Technology - Part II. `http://developers.sun.com/mobility/midp/articles/bluetooth2`, 2003.
[26] G. Sarswat and J. Noida. Bluetooth Hacking. `http://cnss.wordpress.com/2007/09/11/bluetooth-hacking1`, 2007.
[27] W. Stallings. Wireless communications and networks, 2005.