# AN ARCHITECTURE ENABLING BLUETOOTH™/JINI™ INTEROPERABILITY

**On Shun Chau, Pan Hui, Victor O.K. Li**

Department of Electrial and Electronic Engineering, The University of Hong Kong, Pokfulam Rd, Hong Kong, China
Email: {h0118759, panhui, vli}@eee.hku.hk

**Abstract** – Service Discovery Protocols allow clients to discover services without actual knowledge of the locations or characteristics of the services. Jini[1] and Bluetooth SDP are two common service discovery protocols. They may meet each other in many environments. But there is still no general architecture to bring them together. In this paper, we introduce an architecture for Bluetooth client to discover Jini services. We introduce the JINI™ Profile for Bluetooth, which runs in three modes of operations, namely, Surrogate, Bridge, and Client. We describe the implementation of a Jini/Bluetooth Surrogate as well as the Bridge which acts as a proxy and allows a multihop connection to a Jini network. To enable reliable end to end connection, we also incorporate session management into our design.

**Keywords** - Bluetooth, Bluetooth Service Discovery Protocol (SDP), Jini, XML

## I. INTRODUCTION

Nowadays, while there are still some debates on whether Bluetooth [1-3] technology will be successful in the future, many Bluetooth products have already entered the consumer market. We can easily find Bluetooth headsets, PDAs embedded with Bluetooth modules or CF cards. Due to its low price and low power consumption, some people believe that Bluetooth will be the future transmission medium between mobile devices such as PocketPC or PDA. Bluetooth, if viewed as a transmission medium, will play an important role for realizing pervasive computing in the future. Jini[4-6], built over a TCP/IP network, is a middleware technology that provides the facilities for service discovery. The service provided may be a toaster, MP3 player, etc. Jini allows service to be advertised in a TCP/IP network. Services provided are recorded in a well-known entity called Lookup Service Register (LUS) such that clients can find available services there. This article will study the interoperability of Bluetooth and Jini.

[1] Jini™ And Bluetooth™ are trademarks of Sun Microsystems and Bluetooth SIG respectively. To simplify the notation, in the rest of the paper we will use Jini™ and Jini, and Bluetooth™ and Bluetooth interchangeably.

The rest of the paper is organized as follows: First we introduce Bluetooth Technology, and then we describe Jini middleware technology. Section IV compares the Bluetooth Service Discovery Protocol (SDP) and Jini Service Lookup Protocol. Section V studies the current strategies for Jini and Bluetooth interoperability and discusses its limitation. Section VI introduces our system architecture. Section VII shows how Bluetooth clients look up Jini Service. Section VIII talks about route discovery and session management. Section IX is the conclusion and description of future work.

## II. BLUETOOTH SYSTEM ARCHITECTURE

Bluetooth is a short range wireless technology developed by the Bluetooth Special Interest Group (SIG). It operates in the unlicensed Industrial, Scientific and Medical (ISM) band, which is centred around 2.45 GHz. It is originally designed for the replacement of cable. Bluetooth uses fast (1600 hops/sec) frequency hopping (FH) technique with 79 channels centered at (2,402 + k)MHz where K = 0 , 1 , 2 ... 78. FH technique is employed for the sharing of the transmission medium and for security. The maximum asynchronous data rate in Bluetooth version 1.1 is 732 kbits/s. Each Bluetooth time slot lasts for 625μs.

Bluetooth devices can operate in one of two modes: Master or Slave mode. Bluetooth devices are organized into Piconets. In a particular Piconet, one Bluetooth device acts as master, and the others as slaves. It is the Master that schedules the data traffic over the Piconet. A collection of Slave devices operating with one common Master is referred to as a Piconet. The maximum allowable number of active Bluetooth devices which may actively participate in a particular Piconet is 8 (1 Master and 7 Slaves). Each active member is indicated by a 3-bit number called Active Member address (AM_ADDR). A slave can send packets to the master only if the master has sent it a data packet. Thus, the slaves cannot send packets to each other directly. Like TCP and UDP in the TCP/IP protocol, there are two kinds of link in Bluetooth communication. They are Synchronous Connection-oriented (SCO) and Asynchronous connectionless (ACL) links. An SCO link can be used for transmission of voice packets which are never retransmitted. An ACL link is used for transmitting data packets. An ACL link supports broadcast and if there is a packet loss (may be due to collision), an ACL packet can be retransmitted.

0-7803-8523-3/04/$20.00 ©2004 IEEE.

A Scatternet can be formed by linking several Piconets together in an ad hoc fashion to accomodate more Bluetooth devices. The discussion of scatternet is out of the scope of this article but our proposed architecture will assume a scatternet environment.

Bluetooth implements Service Discovery Protocol (SDP). The service provided by a bluetooth device is called a profile. Bluetooth v1.1 standardizes serveral profiles[7]. In the upcoming v1.2 more profiles will be standardized. The service registers itself to the SDP Database as a service record. A service record is represented by attribute-value pairs. Here are some important attributes for a service.

ServiceRecordHandle (0x0000) -- acts as a primary key to identify a service in the SDP server.

ServiceID (0x0003) -- UUID that uniquely identifies a service.

ProtocolDescriptorList(0x0004) -- lists the protocol required to support the profile.

The service request procedure is done by a request-respond scheme, through SDP Protocol Data Units (PDU). The Bluetooth protocol stack is illustrated in Fig. 1.

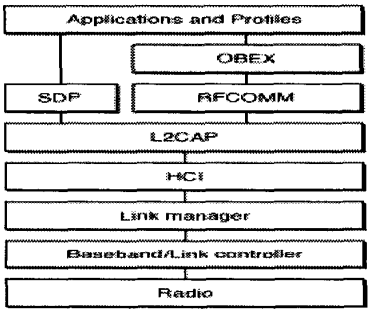


Fig. 1. The Bluetooth protocol stack

## III. JINI ARCHITECTURE

Java Intelligent Network Infrastructure (Jini) is a middleware technology built on the TCP/IP layer to implement a service discovery protocol.

The service provider may be a computer or a hardware device with a controllable interface. As the name "Jini" implies, it is implemented in Java programming Language. If the device does not have a Java Programmable interface, the solution is to add another middleware called COBRA to bridge the two languages.

The core component of the Jini Service Lookup Protocol is the Lookup Service Register (LUS). The service provider advertises its existence and availiblity by registering itself to an LUS. Normally the LUS is the first component to be up in a Jini community and stays in the Jini community rather statically. The LUS advertises its existence by sending out UDP packets with a well-known multicast address. The interested entities will listen for the packets to determine the existence of the LUS. Then the client will communicate with the LUS to search for the services using an unicast discovery protocol. When a service provider do query the LUS, the LUS returns a registrar object (an object which acts as a proxy for the service provider). A copy of the service object will be placed at the LUS.

When a client wants to search for a service, it first creates a ServiceTemplate. Service templates are used by both service providers and the clients for service request matching. It accepts three important parameters.

**ServiceTemplate**(ServiceID serviceID,
java.lang.Class[] serviceTypes, Entry[] attrSetTemplates)

**ServiceID:** An UUID which uniquely identifies the service.

**Class[]:** An array of "Class" objects that defines the Class type of the service provider.

**Entry[]:** An Entry object will conceptually represent the attributes of the service in an object format. The type of the object passed can be regarded as the type of attribute of the service, and the value parameters that follow the Object type are the attribute values.

If there is service match, the client will get copies of matched service object from the LUS. The client can communicate with the service provider through the Remote Method Invocation (RMI) and the service provider will tell the client where to download the service implementation codebase.

## IV. COMPARISON OF BLUETOOTH SDP AND JINI DISCOVERY PROTOCOL

Jini and Bluetooth adopt very different architectures in storing service records. In Jini all of the service records to be discovered are stored in a centralized database called LUS, though it is possible to have more than one LUS in a Jini community. In Bluetooth, the SDP database is distributed, and all the service records of each device are located in its own local SDP database. It is distributed in the sense that if a service want to find all the services available in range it has to perform searching in all the SDP databases of all Bluetooth devices in range.

In order to connect to a service, the client has to connect to the service provider by page inquiry and page scan. After creating an L2CAP layer, the client browses or searches for the services available. From the architecture's point of view, the service lookup protocol of Jini is a bit more complex than Bluetooth SDP. However, Bluetooth is Point-to-Point communication, and service discovery is per device. In Bluetooth, connection to the service provider is made before service discovery, whilst in Jini, the procedure is reversed.

## V. CURRENT JINI/BLUETOOTH INTEROPERABILITY

Jini operates in a TCP/IP network. A solution for integrating Bluetooth and Jini is to utilize the LAN profile or PAN

profile to create a TCP/IP Layer over a Bluetooth network through a Bluetooth access point. One of the critical points for Bluetooth devices to participate in a Jini community is that the access point (AP) should be able to route multicast packets in and out of the Bluetooth network (enabling multicast discovery protocol). Though Jini also supports unicast disocvery protocol for the LUS, it would be rather meaningless for pervasive computing if LUS's IP address has to be known in advance. (For Unicast Discovery Protocol, the address of the LUS has to be known in advance).

The specification of the Jini surrogate architecture (v1.0)[8] is not released until recently. This architecture is a framework that allows resource-constrained devices to participate in a Jini community through a surrogate. The surrogate host communicates with the device through the private interconnect protocol. One of the important requirements of the interconnect protocol is that it must be able to tell whether the surrogate host and devices are connected or not. The Jini surrogate specifications only give a general architecture. It does not give implementation details for Jini/Bluetooth interoperability and its services are only limited to one hop. There is some work on Jini/Bluetooth interoperability [9]. But again the range of the Bluetooth client to the Jini surrogate is only limited to one hop. Our goal is to develop an architecture which enables Bluetooth devices to look for Jini services up to multiple hops away.

Serveral factors affect our architecture design:

1. The limit on the range of Bluetooth radio:

Life would be easy if every Bluetooth device is just one hop from the access point. One of the design features of the Bluetooth device is power conservation. To limit power consumption, the power of the bluetooth radio is rather small. Currently most of the Bluetooth modules use Class 3 radio (1mW). The maximum range is 10m, but due to interference, the typical range of the Class 3 radio is just 5-6m. In the case of the PAN or LAN profiles, if the access point is not within range of the client device, multicast routing of packets has to be peformed over multiple hops. This will complicate the system even though we have not taken into account the possible leaving of intermediate nodes. In this article, we will propose a bridge architecture for routing of Jini service requests in a Bluetooth network.

2. The ad hoc characteristics of Bluetooth network

In a Bluetooth network, it is assumed that all the nodes have a rather high mobility; they can freely join or leave arbitrarily. Our proposed architecture should be able to deal with possible changes of the Bluetooth network topology.

3. The constraint of Bluetooth device

Bluetooth is rather lightweight and simple, it is designed for cable replacement and for embedded systems. Likewise, any protocol built on the Bluetooth network should also have the same quality.

## VI. SYSTEM ARCHITECTURE

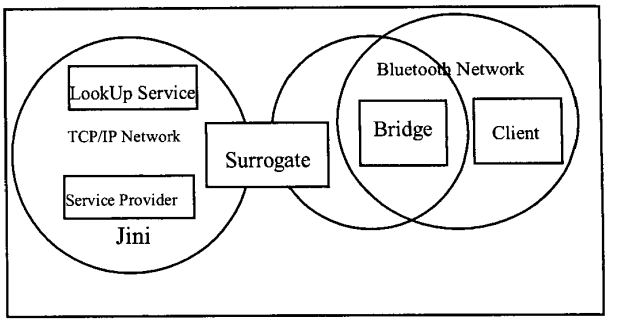


Fig. 2. Architecture Overview

The proposed architecture is presented in Fig. 2. There is a Jini community deployed on the TCP/IP network. In order for a Bluetooth device (client) to use Jini service, it has to connect to the Surrogate. If the Surrogate is not within range, the client cannot connect to it directly. Instead, a bridge device will be used to route the service request to the Surrogate. The following will discuss how the system can be implemented, accounting for the limitations described above.

In our proposed architecture, we build a new profile for Bluetooth called the Jini™ profile. This profile runs in three modes of operations, namely the Surrogate, Bridge, and Client. Fig. 3 shows the Bluetooth protocol stacks needed for Jini Service Lookup.

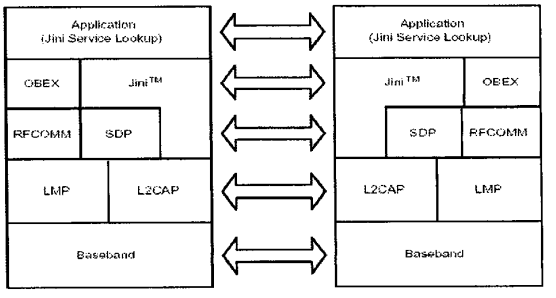


Fig. 3. The Bluetooth Protocol Stack for Jini Service Lookup

The Jini™ profile defines the architectural components and necessary procedures cooperating with the Jini Surrogate architecture to allow a client in a Bluetooth network to perform Jini service lookup.

Surrogate performs session management and parses Jini service lookup requests from an XML document sent by a device in a Bluetooth network. The Client initiates Jini service lookup requests. The request description is put into an XML file and transmitted through an OBEX-FTP profile.

The Bridge routes such Jini service lookup requests in a Bluetooth network to enable multihop service discovery.

**Surrogate (For Access to the Jini network)**

The architecture of the Surrogate is shown in Fig. 4. It consists of two large modules, one interfacing with the Bluetooth network and the other the Jini community. The Surrogate can be decomposed into serveral modules. (The term "Surrogate" in this article is defined as Bluetooth Jini Profile--Surrogate operation mode. This definition is different from that of SUN™; however our proposed "Surrogate" includes the majority of the functions in SUN™'s definition of surrogate )

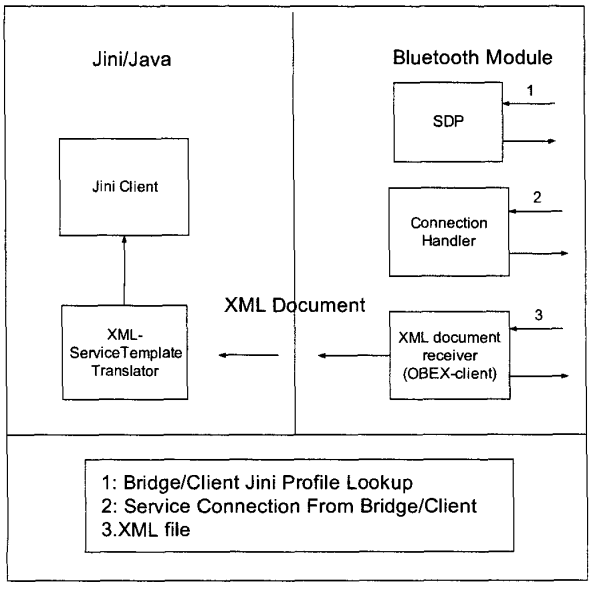


Fig. 4. The internal structure of a Surrogate with Jini Client included

1. Surrogate SDP Database:

The Surrogate's SDP Database has a record Jini Profile with "Surrogate" as the mode of operation. It allows its surrogate service to be discovered.

2. Jini Profile Connection Handler:

It receives connection requests from Bridge or Client and performs session management. It can also monitor the client connection status.

3. XML document receiver (OBEX-Client)

It acts as the client for OBEX-Client for the XML document. When the file is fully received, the file is stored in a temporary place. OBEX, the Object Exchange Profile, is a protocol allowing object push, pull and initialization of the Object Exchange Session. It forms the basis for other profiles such as the FTP profile which allows flat file transfer over the Bluetooth network. We will use OBEX-FTP for the transmission of XML documents in our proposed architecture.

4. XML-ServiceTemplate Translator

Converts XML data to Jini ServiceTemplate (a Java object that represent a service query). The translation process is written in Java Programming language.

5. Jini Client

Uses the ServiceTemplate constructed by XML-ServiceTemplate Translator to query for service in LUS.

In this Surrogate design, we assume that every Bluetooth device in our proposed architecture supports OBEX-File FTP profile for the transfer of XML file. The advantage of using XML document to represent Jini service request is that XML document is platform independent and the intermediate bridges that route the service requests need not understand its contents, and it is only the Surrogate's job to perform the "XML to Jini service request" translation.

**Bridge (An intermediate to the Surrogate)**

The main task of Bridge is to route the Jini service requests to the Surrogate. The Bridge will not register itself to the local SDP database if it cannot find a Bridge or Surrogate within range. Once the Bridge registers itself to the database, there would be a path for the Jini service messages to get to the Surrogate for further processing. The Bridge is divided into three different modules illustrated in Fig. 5

1. Surrogate Lookup Module:

The Surrogate Lookup Module performs the job of searching possible paths from the Bridge to the Surrogate. Decision on routing is based on a shortest path algorithm. When a bridge first starts up, it performs the following actions

(1) Performs SDP search for all devices within radio range for Jini profile.

(2) For any devices running Surrogate or Bridge mode found in range (neighbors), creates a data structure neighbor(*). Gets the required data (neighbor_address, role and distance_to_Surrogate) from its neighbors. At initialisation, last_refresh_time is set to current system time. The data structure is implemented as a list. The Bridge will periodically send ping packets to neighbors and listen to replies for their existence. If (current_system_time - last_refresh_time) is larger than certain timeout, the neighbor entry for that neighbor will be removed from the list.

Note: neighbor denotes a neighboring device running Surrogate or Bridge Mode, neighbor denotes the data structure representing a neighbor.

```
struct{
        bd_addr neighbor_address; // 48 bit hardware address of the
        neighbor
```

```
int role // 0 if the neighbor is Surrogate, 1 for Bridge
        ;
int distance_to_Surrogate; // number of hops to Surrogate
Time last_refresh_time; // last time the neighbor responds with a
"ping"
} neighbor;
```

(3) If the number of neighbors found (we call it neighbor) is larger than 0, the Bridge registers its service to the local SDP Database, scans the list generated in (1), finds the least distance_to_Surrogate, sets its "distance to Surrogate" to distance_to_Surrogate + 1. If there is no neighbor found, terminates the action. Then restarts from (1) periodically.

The routing decision is that the Bridge will route the first incoming request of a session to the neighbor with the smallest value of distance_to_Surrogate. For later messages, the path chosen will follow the first routing decision made and the Session Manager described below will manage it.

2. Session Manager:

In our proposed architecture, the Surrogate is a well-known entity as it is configured to be discoverable by SDP. However, the Client is not configured to be discoverable by other entities in the SDP database. Also we use request-response scheme for Client-Surrogate communication (shown in Fig. 6). In order to allow the response messages to be routed back to the Client, and allow the path session condition along the path to be monitored, we introduce session management along the Bridges:

When a packet with a new route creates a data structure with format [from_neighbor_bd_addr; session_ID ; to_neighbor_bd_addr ; last_renewal_time ] (*).

(a) from_neigbour_bd_addr is the 48-bit hardware address of the neighbor which sends the request.

(b) to_neighbor_bd_addr is the hardware address of the neighbor which the request is routed to.

(c) session_ID, is a unique id created by the client. The session_ID's format can be the Client's timestamp appended with the Client's hardware address. It is used to uniquely identify a session. The session_ID is generated by the Client and is associated with each Client's request.

(d) last_renewal_time represents the time at which a message with a session_ID pass through this node.

The procedure is repeated until the request is routed to a Surrogate. It allows the request and response messages to flow along the same path.

3. OBEX-FTP module:

The OBEX-FTP module utilizes the OBEX-FTP profile. It is used for routing Jini service description files (in XML format). The routing of the file uses the PUSH command for file transfer. When the Bridge receives files, it acts as the Server. When forwarding it acts as the Client.

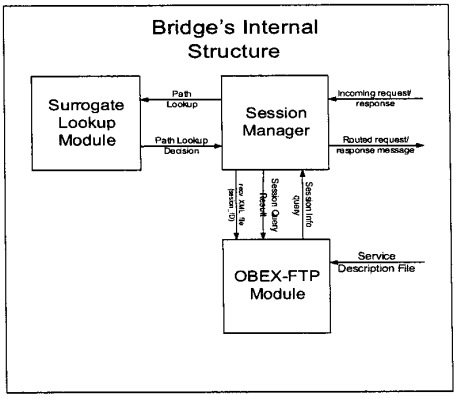


Fig. 5. The internal structure of a Bridge

**Client**

1. The Client performs SDP search for Jini Profile on all the Bluetooth devices within range.

2. For each Surrogate or Bridge found, records its ID on the list of neighbors.

3. Connects to the neighbor with the smallest distance_to_Surrogate value.

4. Starts to send the XML service description document if a successful response is received.

## VII. HOW BLUETOOTH CLIENT LOOKS UP JINI SERVICES

Assumptions:

(a) The Client will not leave during the process. If the Client leaves, the handoff issues should be considered and this is now not yet incorporated into our design.
(b) The routing path has already been set up.
(c) The Jini LUS has already been set up
(d) The XML document describing the service is valid.
(e) All the entities in the Bluetooth network in direct communication are connected after the page inquiry and page scan procedure.

Look up procedure:

(1) The client searches its neighbors for any Jini profile in Bridge mode or Surrogate mode.

(2) The Bridge's SDP replies with ServiceRecordHandles.

(3) Knowing that there is Bridge Service (by doing SDP service search when the Bridge is up), the client connects to the service.

(4) The Bridge knows that there is a Surrogate in vincinity, and requests for connection.

(5) The Surrogate gets the request, and creates a session.

(6) Session creation is successful, the Surrogate replies with a success message.

(7) The Bridge returns a message signalling success to the client, creating a session.

(8) The client starts to send the XML service description document (service.xml) through OBEX-FTP profile.

(9) According to the session setup, the Bridge routes the file to the Surrogate.

(10) The Surrogate performs XML-Jini ServiceTemplate translation.

(11) The Surrogate performs Jini service look up at the LUS.

It is optional that the Surrogate first performs service search in advance and stores the service objects in its data structure. ServiceTemplate matching may be performed locally in the Surrogate if the Surrogate itself has a module which performs ServiceTemplate matching. In this case, procedure (11) mentioned above can be omitted.

The actual application is not limited to one intermediate bridge, as in this case. The Bridge will choose the shortest path to route to the Surrogate.

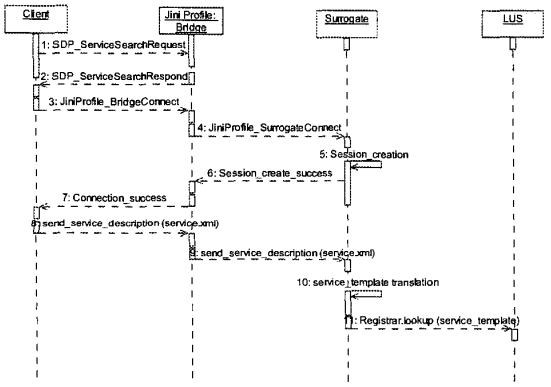


Fig. 6. Chronological Sequence of the Service Look up Procedure

## VIII. ROUTE DISCOVERY AND SESSION MANAGEMENT

The route from the Client to the Surrogate may pass through several bridges. These bridges may be highly mobile and hence the wireless network connection may break easily. To enable reliable service and to minimize retransmissions, we introduce a session management protocol [10] into our design. A session is a reliable message exchange, conducted between any two communication applications. The design of the SM protocol is based on the Open System Interconnection (OSI) Session layer protocol. The Session layer activity and checkpointing procedure are incorporated into the protocol. If the transport and wireless network connections are dropped, then the SM entity will re-establish the transport as well as the wireless network connection. Data transfer will resume at the point of disruption.

## IX. CONCLUSION AND FUTURE WORK

This paper proposes an architecture for the a Bluetooth client to look up Jini services. We describe the implementation details of the Jini/Bluetooth Surrogate. It should be noted that the Bridge and the Client do not require Jini class package to be installed. We suggest using XML which is a system independent data exchange method. We also propose a bridge architecture for a bridge to route Jini service request to the surrogate host machine in the Bluetooth network. Our future work includes designing the protocol for Jini network to search for Bluetooth services. This will enable complete Jini/Bluetooth interoperability.

## ACKNOWLEDGEMENT

This research is supported in part by the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No.AoE/E-01/99).

## REFERENCES

[1] Bluetooth Special Interest Group, "The Bluetooth Specification," http://www.bluetooth.com/developers/specification/specification.asp, April 2000.

[2] J. Bray and C.F. Sturman, Bluetooth: Connect Without Cables, Prentice Hall PTR, Upper Saddle River, New Jersey, 2001.

[3] C. Bisdikian, "An Overview of the Bluetooth Wireless Technology," IEEE Communication Magazine, vol. 39, pp. 86-94, December 2001.

[4] Jini Network Technology, "Jini Specifications V2.0," http://wwws.sun.com/software/jini/specs/index.html, June 2003.

[5] S. Oaks and H. Wong, Jini In A Nutshell, O'Reilly, March 2000.

[6] J. Allard, V. Chinta, S. Gundala and G.G. Richard III, "Jini meets UPnP: An architecture for Jini/UPnP interoperability," Proc. Symposium on Applications and the Internet 2003, January 2003.

[7] Palowireless, "Bluetooth Tutorial-Profiles," http://www.palowireless.com/infotooth/tutorial/profiles.asp.

[8] The *Jini™ Technology Surrogate Architecture Specification* version 1.0 Standard, http://surrogate.jini.org, October 2003.

[9] S. Kasper and L. Buhrer, "Jini Discovers Bluetooth," http://www.tik.ee.ethz.ch/~beutel/projects/sada/2002ss_sa_vincent_bt_jini.pdf, Summer 2002.

[10] A.C.C. Lo, V. Chandrasekaran, W.K.G. Seah, and C.P. Soh, "A session management protocol for mobile computing," in Proc. of IEEE GLOBECOM, vol. 5, pp. 2592-2598, Nov.1998.