Proxying UPnP service discovery and access to a non-IP Bluetooth network on a mobile phone

Proxying UPnP service discovery and access to a non-IP Bluetooth network on a mobile phone

A. Delphinanto, A.M.J. Koonen
Technische Universiteit Eindhoven
Eindhoven, The Netherlands
A.Delphinanto@tue.nl

M.E. Peeters, F.T.H. den Hartog
TNO
Delft, The Netherlands
Frank.denHartog@tno.nl

Abstract— **The current service- and device discovery protocols are not platform- and network independent. Therefore, proxy servers will be needed to extend the range of IP-based discovery protocols to non-IP domains. We developed an architecture of a proxy that enables Universal Plug and Play (UPnP) devices and services to be discovered and accessed on the Bluetooth network and vice versa. We optimized and implemented the architecture on a mobile computing platform. This proxy implementation is then used for interworking the UPnP Content Directory service with the Bluetooth File Transfer profile. Our performance study showed that our proxy implementation reduces invocation time and data throughput to about 50% of the bare Bluetooth and UPnP performance, but it is still acceptable for an end user.**

Service discovery, device discovery, UPnP, Bluetooth

I. INTRODUCTION

Service discovery technology is an important component for communication and service collaboration in distributed computing environments. In private networks, such as home networks and in-car networks, these environments are very heterogeneous considering the great variety of devices, network technologies, control protocols and application platforms being used. The existing service discovery frameworks use different sets of protocols and infer specific transport protocols. None of the protocols is completely platform- and network independent. For example, UPnP (Universal Plug and Play) operates at the network layer and above, and only runs on Internet Protocol (IP) networks. Bluetooth is a link layer protocol and Bluetooth Service Discovery Protocol (SDP) only runs on Bluetooth networks (and in the future probably also on UltraWideBand). As a result, Bluetooth devices such as headsets and some MP3 players cannot discover and use UPnP-enabled content servers in the IP part of a heterogeneous home network. Most probably, time will not solve this problem. Although Bluetooth can support IP already in many years, it does not in practice because industry still considers IP as too resource intensive for small devices (headsets, sensors, etc.) and too vulnerable for critical communication such as car control. Therefore, proxy servers will be needed to provide an interoperability platform.

Koponen et al [1] presented a bidirectional proxy for Jini and Service Location Protocol (SLP) interoperability. Allard et al [2] presented a proxy architecture Jini-UPnP interoperability. The proxy uses the client interfaces from both SDPs to listen to any server announcements, generates a virtual server object for each discovered server and finally announces the virtual server to the counterpart network. With a comparable architecture, Jun et al [3] proposed a unidirectional Bluetooth SDP-UPnP proxy server enabling a UPnP control point to control Bluetooth servers. In [4] we presented our design of a bidirectional proxy that enables UPnP services on an IP network to be discovered by a Bluetooth client in a non-IP Bluetooth network (piconet) and vice versa.

In this work, we show the results of our efforts to integrate and test a proxy implementation on a mobile phone to provide seamless interworking between the UPnP Content Directory Service (CDS) and the Bluetooth file transfer profile (FTP) over the two domains.

The paper is structured as follows. In the following section we discuss the proxy server definition and architecture. Then, we explain the interworking between the UPnP CDS and the Bluetooth FTP. Then our implementation of the proxy is presented, whereas its performance evaluation is consequently discussed in section V. The final section summarizes the conclusions.

II. PROXY SERVER DEFINITION AND ARCHITECTURE

A. *Definition*

A proxy server is a server that handles the requests of its clients by forwarding requests to other servers. A client sends a request for a file, control action, web page, or other resource to the proxy server. The proxy server provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy server may optionally alter the client's request or the server's response, and sometimes it may serve the request without contacting the specified server. A well known proxy example is a web proxy, which is used for many purposes, such as enforcing acceptable network use policy, providing security, caching services, or reformatting web pages (e.g. into the ones that fit in cell phones or PDAs). In this paper we are looking at proxies that are used to reformat device- and service discovery and –control messages. In particular it is focused on the example of forwarding a service access request of a non-IP Bluetooth client to any UPnP server,

The work presented in this article has been carried out in the collaborative Freeband Communication technology program which is supported by the Dutch Ministry of Economic Affairs.

and when a return is expected, the return is reformatted and forwarded back to the Bluetooth client.

B. *Architecture*

Jun et al [3] showed the way how to design a mono-directional proxy that makes Bluetooth services available to a UPnP network. In [4], we gave a detailed architecture of a remotely upgradeable bi-directional proxy server. Here, in Figure 1, we show the elements of our proxy architecture that enable Bluetooth clients to discover and access UPnP devices and services, namely: UPnP client and Bluetooth interface (BI), an adapter, a proxy registry, a mapping library, and dynamic virtual server objects. We divide making a UPnP server accessible by a Bluetooth client into two steps, namely discovery and access.

The main goal of the discovery step is to make a UPnP server available for prospective Bluetooth clients. This step is initiated by the UPnP client or control point (UC) either sending discovery message or listening to any UPnP server (US) advertisement (action 1 in Figure 1). The UC then passes this information to the adapter module (2). The adapter filters the incoming service information to the mapping library if mapping for this service is supported, after which the adapter registers the service in the proxy registry (3). The adapter then generates a Bluetooth virtual server (BVS) (4). This virtual server is a software object that implements any Bluetooth service class and the UPnP client interface. To make the virtual server available to the Bluetooth client, this server must be registered as a Bluetooth service object in the BI (5). Through the UPnP client interface of the proxy, the virtual server subscribes to the US's events to be aware of any server state changes (6).

The access step accommodates the Bluetooth client to access the UPnP server via the BVS. Bluetooth SDP does not specify a server advertisement, thus to get information about a Bluetooth server and its services, a Bluetooth client needs to search or browse a Bluetooth server (7). This searching is possible after the physical server is discovered and the link is established. The BVS will act as an actual Bluetooth server application. When a Bluetooth client desires to access a UPnP service,, the client may send an invocation message to the BVS through the BI (8). Accordingly, the BVS will reformat the message to a UPnP environment message and forward it to the US (9). A return message may be expected (10). When that is the case, the BVS will reformat the UPnP return message into a Bluetooth environment message and forward it to the Bluetooth client (11 and 12).

C. *Requirements*

We expect the proxy service to be provided by service providers (operators) in conjunction with mobile phone vendors. We therefore demand the proxy to interoperate with as many as possible other consumer electronics currently on the market. It should also be possible to implement it on a mobile computing device. This leads to the following design requirements.

In Bluetooth environments, services can be discovered using the Bluetooth SDP but have to be accessed using other protocols. In UPnP, the discovery as well as the access is defined by the protocol. To achieve seamless discovery of and access to current devices and services, we apply a filtering process in the adapter module. The adapter only accepts UPnP services if they are currently standardized and converts them if these services correspond with any standardized Bluetooth profile offering similar functionalities. The example being further elaborated in this paper is a UPnP Content Directory Service (CDS), which corresponds to the Bluetooth File Transfer Profile (FTP), both having a file server service. The service provider can configure and upgrade this filtering function via remote management when new UPnP and Bluetooth standards are released.

To minimize the need for internal resources, we combine the device announcer and discoverer modules from [4] into the UC. Furthermore, we allow only one Bluetooth virtual server at a time.

III. INTERWORKING UPNP CDS WITH BLUETOOTH FTP

A. *UPnP CDS*

UPnP is an interoperability framework for devices and services in a relatively small-scale IP network. It is based on the client/server model and distinguishes three logical entities in the network: UPnP Services, which represent the service functionality of a device, UPnP Devices, which act as services

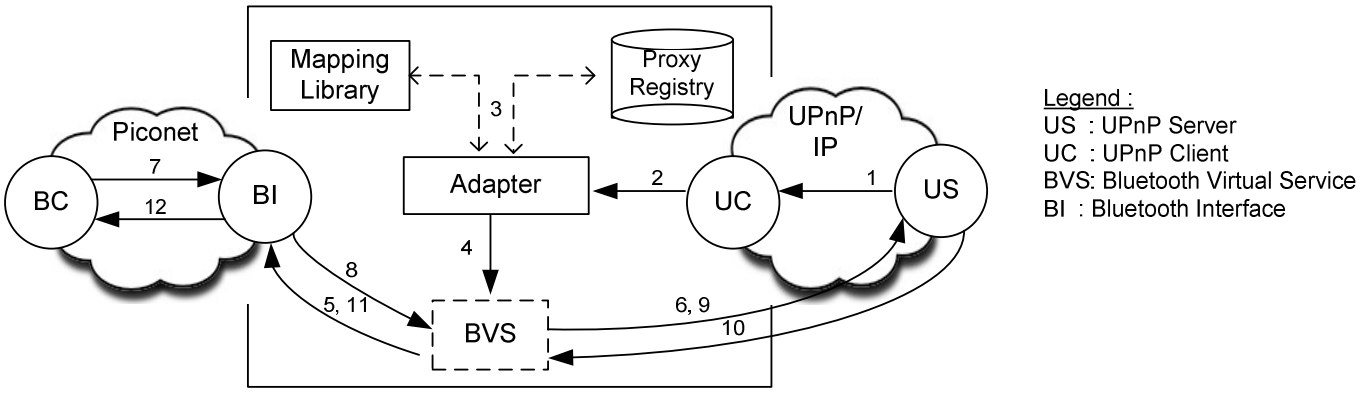

Figure 1. The architecture of a proxy that enables seamless interworking between a UPnP server and a Bluetooth client

servers, and UPnP Control Points (CPs), which act as clients for controlling the services. UPnP defines Dynamic Host Configuration Protocol (DHCP), Simple Service Discovery Protocol (SSDP), Simple Object Access Protocol (SOAP), and General Event Notification Architecture (GENA) for addressing, discovery, control, and eventing, respectively. Device and service descriptions are expressed and partially standardized in eXtended Markup Language (XML) templates.

The UPnP *ContentDirectory* service (CDS) [5] is one of the UPnP standard services. This service is aimed to provide a uniform mechanism for a user interface application to browse, locate, or transfer any content (e.g. songs or video files) from a server and to obtain detailed information about individual content objects. Natively, the UPnP CDS is one of default services in the *MediaServer,* one of the standard UPnP devices, which is used to expose search and browse capabilities [6]. Some commands in the CDS that are relevant with this work are *Browse(), Search(), and ImportResource().* For file transfer the CDS uses existing transport protocols i.e. *HTTP GET* (for downloading) and *HTTP POST* (for uploading).

B. Bluetooth FTP basics

Bluetooth is an industrial specification for wireless personal area networks. Primarily designed for low-power consumption and short-range coverage, Bluetooth provides a wireless way to connect and exchange information between mobile devices such as mobile phones, laptops, digital cameras and etc. A Bluetooth network is an ad-hoc network (*Piconet*) that works in a master-slave fashion. A master may have connections to 7 active slaves and up to 255 inactive (parked) slaves. In Bluetooth communication, data can only be exchanged between a master and one slave at a time. However, these devices can switch roles and the slave can become the master at any time. Bluetooth link technologies (i.e. the Bluetooth core communication protocol) and applications (i.e. Bluetooth profile) are maintained by Bluetooth Special Interest Group (SIG).

One of the Bluetooth core protocols is the Bluetooth Service Discovery Protocol (SDP). As the SDP is designed for highly dynamic communications, it only provides a means for client applications to browse available services or search for a desired service in a Bluetooth server. Each Bluetooth service is referred to as a Bluetooth service record and is an implementation of a Bluetooth service class. Each service class is distinct and is distinguished by a universal unique identity. To access the services, the Bluetooth SIG defines Bluetooth profiles, which describes standard operations and protocols for applications that use a Bluetooth communication interface.

The relevant profile for this work is the Bluetooth File Transfer Profile (FTP) [7]. The profile requires the devices to follow the Bluetooth Generic Object Exchange Profile (GOEP) and uses OBject EXchange (OBEX) as the file transport protocol. It enables a client to browse, transfer and delete files in a server. Those commands are carried out by combinations of OBEX commands (*CONNECT*, *DISCONNECT*, *PUT*, *GET*, *SETPATH* and *ABORT*). For example, the client browses files in the server using *GET FolderListing* and sets a current folder using *SETPATH, or* the client may retrieve or upload a file using *GET* and *PUT*.

IV. IMPLEMENTATION

To validate our current architecture, we created a scenario and implemented the concept in a mobile platform. Here, we focus on the service access functionality of the proxy and the respective performance analysis rather than service discovery (which has been evaluated already in [4]). Our implementation consist of a mobile platform (HTC TyTN, 400 MHz processor, 64 MB RAM), an operating sytem (Microsoft Windows Mobile 5.0), an internal database (Microsoft SQLCE Server), a UPnP library (Intel UPnP CE Stack, Bluetooth library (32Feet Bluetooth Stack) and an XML Parser library (System.XML library provided by the .Net framework).

The scenario is as follows. In a home, a user with a portable MP3 player wants to download a music file from a server. This activity cannot be done straight forward because the MP3 player has only a Bluetooth interface, the server only has a Wifi IP/Ethernet interface, and the Bluetooth applications cannot directly communicate with UPnP applications. The user has a mobile pocket PC that has Bluetooth and Wifi interfaces and runs the proxy application. Using this pocket PC as an intermediate, the user effortlessly downloads the file.

The implementation of the scenario is explained using a sequence diagram (depicted in Figure 2) that contains a client representing the Bluetooth MP3 player, a server representing the UPnP server, and the proxy. The actions in the scenario can be grouped into the following technical activities (distinguished by different time slots in Figure 2): *initialization*, *browsing* (a file), *changing folder* information and *downloading* (a file).

1) Initialization

Before starting the other activities, an initialization is carried out. The server that contains a UPnP CDS should be discovered by the proxy, for instance by sending a *notify()* message. Then, the proxy should perform a UPnP-to-Bluetooth service static description mapping, for instance as described in [3]. Upon receiving the server's notification, the proxy invokes the server with a *Browse()* action. The server will reply with an XML file (A1 in Figure 2) containing folder (root and its children) information. This information is used as an initial reference to the folder locations. Simultaneously, the client discovers the proxy and establishes a connection, whereafter the client looks for a Bluetooth FTP.

2) Browsing,

To browse the server, the client invokes the proxy with a *GET* action with a parameter *FolderListing*. To respond to the invocation, the proxy will first reformat the file A1 into another XML file (B1) and send this file to the client. The file reformatting is needed because the UPnP XML file has a different structure and meaning from the Bluetooth XML file. For example, UPnP identifies folders by an identity while Bluetooth (OBEX) identifies folders by a name. For this reason, the proxy should keep pairs of Bluetooth folder names and UPnP folder identities. When the Bluetooth client has received the file B1, the client may parse it to a user-friendly

format for the end user. The user can browse through the folder like any other file browser he is used to.

3) Changing folder,

To change folder hierarchy, the Bluetooth client invokes the proxy with a *SETPATH* action with a folder name as parameter. The proxy will match the given folder name with available folder identities and in case of successful matching, the proxy replies the invocation with an *ok* message. To get the final folder structure, the client invokes the proxy again with a *GET(FolderListing)*. Accordingly, the proxy invokes to the server with a *Browse()* action with the matched folder identity as parameter. The server will respond the proxy's invocation with a new XML file (A2) containing new structure information of the requested folder. Similarly, like in the browsing, A2 is converted to a new OBEX file (B2) and sent to the client.

4) Downloading

The client downloads a file from the server by sending a *GET* action to the proxy with the desired file name (*X*) as parameter. The proxy looks up the location (i.e. *file-uri*) of file size. The proxy wraps each of these packets into a Bluetooth OBEX packet and sends it to the client. Upon each successful packet reception, the client must confirm to the proxy with an *ok* message, after which the proxy can continue downloading the file. This is repeated until the file is completely transferred. The proxy acknowledges the client about the completed transfer by sending a *download-finished* message.

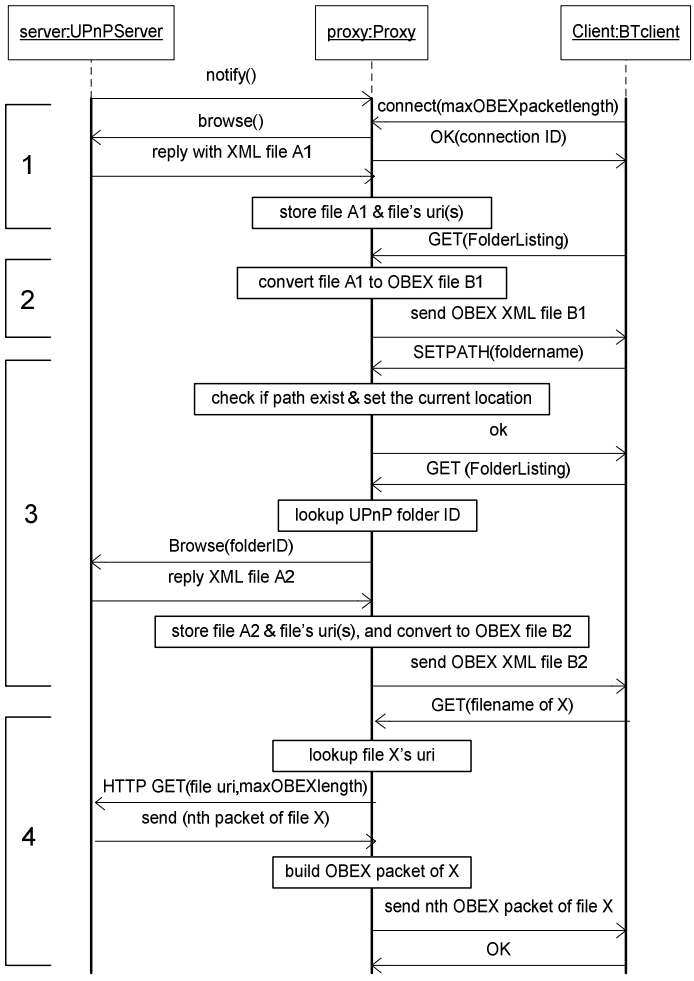

Figure 2. Sequence diagram for interworking between UPnP CDS service and Bluetooth client

During implementation, we found that the UPnP CDS and Bluetooth FTP work quite differently, though they have similar functionality and commands. For example, for browsing, the client needs to send two invocations (i.e. *SETPATH* and *GET*) while the server only needs one invocation (i.e. *browse)*. From this we conclude that the proxy needs specific solutions for each conversion between Bluetooth profiles and UPnP services.

V. PERFORMANCE EVALUATION

Using the proxy implementation, we measured the proxy's response times for browsing files and downloading a file. For the browsing response time we measured the total browsing time and the UPnP browse time as perceived by the proxy. The total browsing time is measured from the time the proxy receives the client's request (i.e. *SETPATH*) until the time the proxy replies the *Get FolderListing* to the client. This total time is identical with the time slot number 3 in Figure 2 (i.e. changing folder). The total browse time minus and the UPnP browse time yields how much latency the proxy added to the original UPnP browse time. The downloading time is measured from the time the proxy receives a client download request (*GET file)* until the time the proxy sends the download-finish acknowledgment to the client. Each measurement is repeated 40 times to minimize sampling error.

Figure 3 shows the measured total browse time and UPnP browse time as a function of folder level (i.e. the position in the folder hierarchy). Level 1 represents a browsing to the root (top) folder hierarchy of the file server. In our case the root contains three folders. The root information is obtained during the initialization process (as discussed in section IV). Therefore, the root browsing does not require a separate server invocation. This explains why the UPnP browse time at level 1 is zero. Level 2 represents browsing to one of the three subfolders of the root folders. Level 3 means browsing to one of eleven subfolders of the level 2's folders. Finally, level 4 shows the browsing to the lowest position of the folder hierarchy, which contains 28 files. The figure shows that the total browse time is almost double the UPnP browse time and both response times increase with the amount of folder/file information available. From this we can conclude that the proxy processes for the browsing activities (i.e. looking up folder/file identities, updating new folder information and converting UPnP XML folder structures to OBEX folder structures) take about half the total browsing time, which is significant. However, also for complicated file folder structures (e.g. with four levels), the total browsing time is still acceptable for the end user. From this we conclude that our proxy design can indeed be implemented on current mobile platforms, adding value to current Bluetooth and UPnP services.

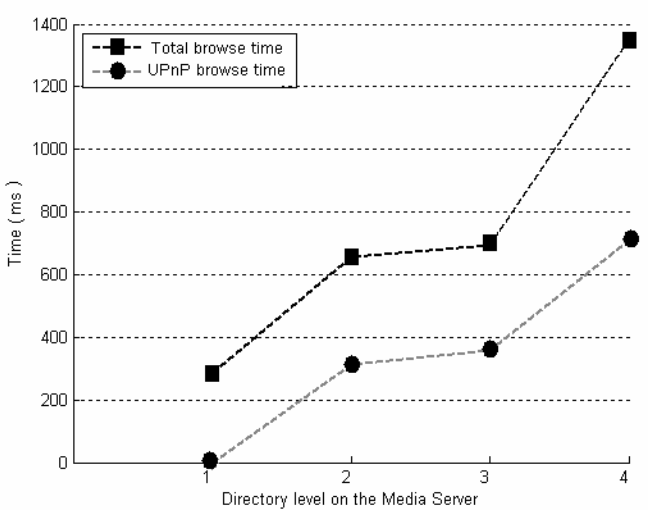

Figure 3. The Proxy browse time at different levels of file folder

We also measured the payload data throughput when downloading a file from a UPnP server to a Bluetooth client via the proxy. We found a throughput of 150 ± 30 kbps, which is about half the native Bluetooth throughput. The throughput on the UPnP network is many Mbps, so the loss can be accounted to the processing activities of the proxy. For e.g. streaming audio, the measured rate is still high enough.

We also repeated the measurement adding a second Bluetooth FTP client that downloads another file simultaneously during the download time measurement. We then obtained a rate of 70 ± 20 kbps. This rate drop occurs because the Bluetooth link fairly divides the number of channels between the two clients. In this case the audio file has to be downloaded or partly buffered before being consumed to achieve an acceptable audio quality.. From this we conclude that the number of Bluetooth links to the proxy should be limited.

VI. CONCLUSIONS

Proxy servers can provide an effective interoperability platform for service discovery and access in a heterogeneous private network. We developed a proxy architecture for seamless interoperability between Bluetooth and UPnP servers that can run on the high-end mobile devices of today. This paper described our experiences when implementing this architecture on a mobile platform.

We found that the UPnP CDS and Bluetooth FTP work quite differently, though they have similar functionality and commands. From this we conclude that the proxy needs specific solutions for each conversion between Bluetooth profiles and UPnP services

For our performance measurements we only focused on the effect of the proxy on service access times, because we evaluated the discovery performance before [4]. We measured the effect that the proxy has on browse times and data throughput when using a Bluetooth client to access files on a UPnP server. We found that our proxy implementation reduces browse time and data throughput to about 50% of the bare Bluetooth and UPnP performance, but this is still acceptable for an end user.

ACKNOWLEDGMENT

We thank Rob van der Veer for his valuable comments.

REFERENCES

[1] T. Koponen and T. Virtanen, "A Service Discovery: A Service Broker Approach", in Proc. of the 37th Hawaii International Conference on System Sciences, 5-8 Jan. 2004.

[2] J. Allard, V. Chinta, S. Gundala, and G. R. III, "Jini meets UPnP: An Architecture for Jini/UPnP interoperability", Proc. of Symposium of Applications and the Internets, pp. 268-275, 27 – 31 Jan. 2003

[3] S.M. Jun and N.H. Park, "Controlling non IP Bluetooth devices in UPnP home network," Proc. of the 6th International Conference on Advanced Communication Technologies, Vol. 2, pp. 714-718, 2004.

[4] A. Delphinanto, J.J. Lukkien, A.M.J Koonen, A. Madureira, I.G.G.M. Niemegeers, F.T.H den Hartog, F. Selgert, "Architecture of a bidirectional Bluetooth-UPnP Proxy", Proc. on 4th IEEE Consumer Communications and Networking Conference, Las Vegas, January 2007.

[5] A. Presser et al, "ContentDirectory:2 Service Template version 1.01", UPnP forum, 31 May 2006.

[6] A. Presser et al, "MediaServer:2 Device Template version 1.01", UPnP forum, 31 May 2006.

[7] D. A. Gratton, "Bluetooth Profiles: The Denifitive Guide", 1st ed., Prentice Hall, 30 Dec. 2002.