Framework for ubiquitous discovery and access to home services

D. Verslype, J. Nelis, T. Verschueren, W. Haerick, F. De Turck, C. Develder
Ghent University - IBBT,
Dept. of Information Technology - IBCN, Ghent, Belgium
Email: chris.develder@intec.ugent.be

***Abstract*—To relieve the end user of the configuration burden that arises when new devices are installed in the (home) network, service discovery protocols (SDP) have been devised. These enable automatic discovery of the services these devices offer. Yet, the existing SDPs are not compatible or interoperable. In this paper, we propose a framework that automatically translates between the various SDPs. We also solve the problem of current SDPs that only enable discovery within a single local (sub)network. Through a remote access component, we enable sharing of the services between multiple home networks. Hence we realize sharing of services over the boundaries of SDPs and home networks. Results of performance measurements on a proof-of-concept implementation illustrate that the framework operates transparently to the various SDPs: translation has no significant impact on response times compared to native SDP services.**

***Keywords*-service discovery, UPnP, remote access, DNS-SD, SLP**

I. Introduction

The modern home network is evolving towards a myriad of different devices that in turn can offer a multitude of services. Common examples of such devices are printers and multimedia content providers e.g. cell phones and cameras, but also media players, e.g. television sets. An end user typically isn't as savvy as required to perform the necessary installation and configuration steps in order to make these services work. As such, a Service Discovery Protocol (SDP) tries to relieve the end user from this burden by dynamically discovering the available services and providing a means to let these services cooperate with one another. This all works pretty well in theory, however, since there also is a multitude of available SDPs, a couple of problems arise. An application that is written to support services announced by one SDP cannot use services announced through another SDP. This effectively divides the home network in virtual networks. These SDP networks are logically separated in that services from one SDP network can not be discovered in another SDP network. Well-known examples of SDPs are UPnP, SLP and DNS-SD. Suppose a printer service exists in the UPnP network, if the user's printing application only supports SLP, it can only discover printer services in the SLP network. The user will not be able to use the UPnP printer through its printer application. Another problem is the fact that many SDPs make use of IP multicasting or broadcasting to announce and discover services. As these messages are not forwarded onto the Internet, the visibility of the services is limited to the home network. In this article a framework will be described as a solution to solve these problems. This will allow services to be shared between different (SDP) networks, in a fully automatic and transparent way for the users. When a service is discovered by an end user's application, it is unaware of what the original location or the SDP used to announce the service was.

This paper addresses these problems by proposing a design of a framework that is pluggable. Besides that we present a performance analysis of the proposed solution.

The remainder of this paper is structured as follows: in Section II we give an overview of widely used SDPs. Subsequently, we present the architecture of our interoperability framework in Section III and an example scenario in Section IV. Section V contains the performance assessment. Final conclusions are summarized in Section VI.

II. Service Discovery Protocols

In this section, we present an inventory of current relatively wide-spread Service Discovery Protocols (SDPs). We briefly describe their fundamental principles of operation, and present a taxonomic summary.

A. Service Location Protocol (SLP)

The SLP [1] architecture describes 3 components. The *User Agent* is responsible for searching services on the network based on a description or on specific attributes. The *Service Agent* can make one or more services available on the network. The *Discovery Agent* is an optional component that can cache the description of one or more Service Agents. This cache service is discoverable using SLP as well.

A User Agent can search for services using a multicast Service Request message, describing the desired service. All Service Agents matching the query will reply with a unicast Service Reply. If a Discovery Agent was found earlier (by the aforementioned technique or by receiving a periodic Discovery Agent Advertisement message), it will not multicast the Service Request, but unicast it to the Discovery Agent, since all Service Agents have put their description in the cache with Service Registration messages upon discovery of the Discovery Agent.

SLP is most commonly used in printer infrastructure,

like the Common Unix Printing System (CUPS), the HP Jetdirect Print Server, etc. Two open-source SLP Java implementations exist: OpenSLP and jSLP ([2]).

B. Jini

Jini is a SOA developed originally by Sun in 1999, currently maintained in the open source Apache River project.
Jini is based on the Java technology and extends the Java virtual machine (JVM) to services on other JVMs. Services can be discovered by querying the *Lookup Service*.
A Jini service is described by a Java interface and an accompanying *proxy object* that implements this interface. This proxy object is a so called *smart proxy*: it will only contact the remote service through a Jini client if necessary, and will try to do as much work as possible locally. To advertise the service, the interface and the proxy object need to be registered in the Lookup Service, so a client can search for it and load the proxy for direct communication with the service.
The Jini technology is useable in a wide range of applications, as it is based on the platform independent Java technology. Applications reside in the field of eHealth, mobile services, military, etc.

C. Domain Name Service based Service Discovery (DNS-SD)

DNS-SD is an SDP, better known as Bonjour, zeroconf or Avahi, based on DNS. No new messages are defined, it just describes how to use the DNS protocol for advertising services.
A DNS server contains a large number of *Resource Records*. DNS is mainly used to translate IP addresses to domain names, information that is stored in the DNS-A records. But aside this information, also services within an internet domain (like mail servers) can be defined in DNS-SRV records, containing IP address, port, used protocol, etc. of the service, and DNS-TXT records, containing an arbitrary description of the service. DNS-SD requires supporting devices in a LAN to run their own DNS server, populated with SRV records according to the available services on the device. The communication is done through multicast, to prevent interference with normal DNS traffic.
Bonjour is the implementation for Apple Macintosh computers, Avahi for Linux. Devices supporting DNS-SD are printers, scanners and Digital Audio Access Protocol (DAAP) servers.

D. Universal Plug and Play (UPnP)

UPnP ([3]) is a set of protocols that together enable service discovery, service description, service control and eventing for LAN services.
The UPnP architecture consist of two components: the UPnP Device, which advertises several UPnP Services, and the UPnP Control Point, which searches and makes use of devices.
For discovery UPnP uses the Simple Service Discovery Protocol (SSDP). When a UPnP Device comes online and at periodic intervals it advertises itself together with its services in multicast SSDP message. A Control Point can listen to these messages or send SSDP multicast searches defining the type of device or service it is looking for, which in turn get answered by the matching UPnP Devices. The Location field of the SSDP responses contains an HTTP location, where the UPnP XML description of the device and services can be found. Every service has a number of UPnP Actions that can be invoked by a Control Point using SOAP and a number of state variables a Control Point can listen to, to get evented changes of the service state, also with SOAP.
UPnP is a widely adopted technology in different fields, and has a broad range of predefined device and service descriptions. For A/V applications, UPnP AV ([4]) defines descriptions for media servers and renderers (implemented by Windows Media Player, XBox 360, Sony Playstation 3, Philips streamium products, etc.), UPnP QoS ([5]) defines descriptions for devices capable of managing quality-of-service in the home network. For remotely accessing a UPnP network, UPnP Remote Access is defined. There also exist predefined descriptions for printers, HVAC devices, lights, etc.

E. Devices Profile For Web Services (DPWS)

DPWS defines implementation constraints for secure Web Service messaging, discovery, description and eventing, similar to UPnP, but build around Web Services.
For discovery, DPWS uses WS-Discovery, a multicast protocol similar to SSDP. For description, DPWS uses the Web Services Description Language (WSDL), a standardized XML format for Web Service description. For controlling the Web Service the SOAP based WS-Control standard is used. For eventing the WS-Eventing is used. A subscriber can specify the delivery mode of the events. DPWS uses the SOAP delivery mode.
As DPWS was standardized in 2008, the adoption of the technology is not yet widespread. The HP Jetdirect Print Server and Windows Network Projector support DPWS.

F. Taxonomy

Analyzing these SDPs shows us that there exist a number of technologies for achieving one or more of these goals: the automatic discovery of services, description and control of these services and the eventing of the service state. Table I

	SLP	DNS-SD	Jini	UPnP	DPWS
discovery	X	X	X	X	X
description	X	X	X	X	X
control			X	X	X
eventing			X	X	X

Table I
COMPARISON OF SDPS

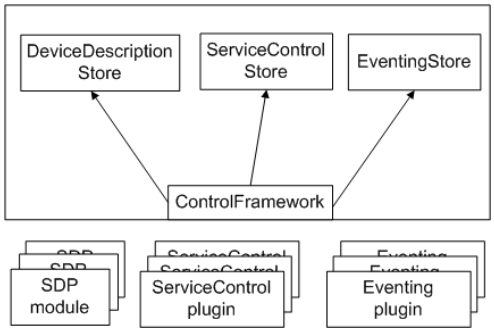

Figure 1. High level architecture of the SDP framework

shows a comparison of the studied SDPs concerning these four aspects. It indicates that the studied SDPs can be seperated in two groups: the ones only supporting service discovery and description, not defining how the services are controlled, and those who offer the complete package.

III. UBIQUITOUS SDP FRAMEWORK

As it is likely that new SDPs will be developed in the future, the proposed framework has to be easily upgradable to support the usage of these new SDPs. The framework therefore uses a plugin model, where support for new SDPs can be added dynamically. Since service types are also subject to change, support for service types, control protocols and eventing mechanisms is provided by plugins.

A. Architecture

The designed framework, depicted in Figure 1, uses a plugin model and acts as a broker between different SDP implementations relying on their respective functionality for service discovery. Furthermore it uses a plugin model for service types, as well as possible control protocols and eventing mechanisms.

An SDP module is responsible for the discovery of services in its SDP network. On the other hand, the SDP module will receive information about services discovered by other SDP modules in the framework, which will allow the SDP module to publish these services in its SDP network. To enable generic service discovery, a generic SDP independent service description is used in the framework. When an SDP module discovers a service on its network, it first translates the SDP dependent service description before registering the service to the framework. The framework will notify all interested SDP modules of the new service. The other SDP modules will translate the service to its service description before announcing it on its network.

Since new service types can be introduced, the framework uses service type plugins that add support for specific types of services. Moreover, control protocols also are supported through the use of plugins. A printer service that uses the LPR [6] protocol for printing requires a plugin for the printer service and a control protocol plugin for LPR is needed. This way an SDP module can rely on these plugins to do the translation for a service announced by an SDP without native control to an SDP that has native control. An example for this is the translation from an LPR printer announced by SLP to a UPnP printer. The UPnP printer interface abstracts the actual printing protocol used, therefore support for LPR has to be available before the translation can succeed.

The research in this paper focuses on the discovery and control of services, but also provides plugin support for eventing mechanisms, since it's possible different SDPs use the same eventing mechanism.

B. Remote access

To address the problem most SDPs have, namely the fact that service discovery and control is limited to the home network, the proposed framework is combined with UPnP-Remote Access (UPnP-RA). UPnP-RA allows the sharing of UPnP services between different networks. This is achieved by forwarding the UPnP services between the two networks over a (secure) tunnel.

As UPnP-RA can only share UPnP services, it has to be used in combination with the framework to enable sharing of non-UPnP services across the home network boundaries. If the UPnP SDP module can translate the service from the non-UPnP presentation to a UPnP service, UPnP-RA will simply share the service across the home boundaries. A second possibility is the UPnP SDP module is unable to translate the non-UPnP presentation to a proper UPnP service. In this case, the framework service description will be added to a UPnP wrapper device such that UPnP-RA can share this device with another home network. Within the remote home network, the local framework will import the services included in the UPnP wrapper device. Like this, non-UPnP services can be shared across home network even without them being translated to UPnP services.

IV. SCENARIO DESCRIPTION

In Figure 2 you can find the sequence diagram of this scenario. It describes what happens when an SDP module discovers a new service.

- the SDP module discovers a new service or device. (step 1 & 2)
- Does there exist a ServiceDescription object and/or DeviceDescription object for the service or device in the framework and can the SDP module do a translation to this object? (step 3,4 & 5)

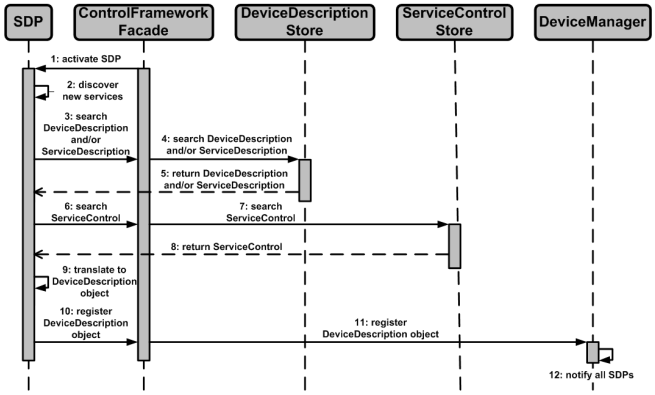

Figure 2. message diagram for the scenario of Section IV

- Yes:
  - If the framework has a ServiceControl object for this service, it gets coupled with this ServiceDescription object. (step 6,7 & 8)
  - The SDP module translates to the available ServiceDescription object and/or DeviceDescription object. (step 9)
  - The translated service or device are passed to the DeviceManager. (step 10 & 11)
- No:
  - An SDP specific DeviceDescription implementation is made without loss of information.
- All SDP modules get notified of the addition of a new DeviceDescription to the framework. (step 12)

V. Performance Assessment

In this section we will evaluate the introduced network load and delay of using a proof-of-concept implementation of our framework on a testbed. We used two techniques for the data: passive discovery, in which an SDP module will only listen to service notification packets; and active discovery, in which an SDP module will broadcast packets itself to actively discover services. The former does not add traffic on the network, the latter does.

A. Proof-of-concept implementation

The most important requirement of the implementation is the pluggability, we need to be able to easily add or remove different SDP modules, plugins for a service type and plugins for a specific control protocol. For this reason we chose OSGi [7] as a development platform.

B. Evaluation setup

In Figure 3 the test setup for the LAN is shown. The framework taking care of service translations is running on PC-A. On PC-B a CUPS print server is sharing a varying number of virtual printers (through SLP or DNS-SD). PC-C will be looking for printer services.

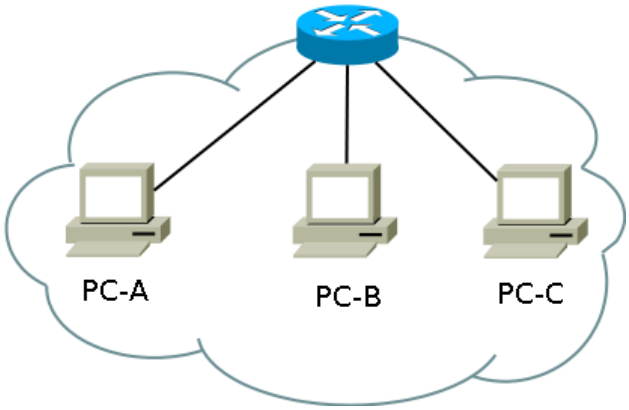

Figure 3. LAN setup

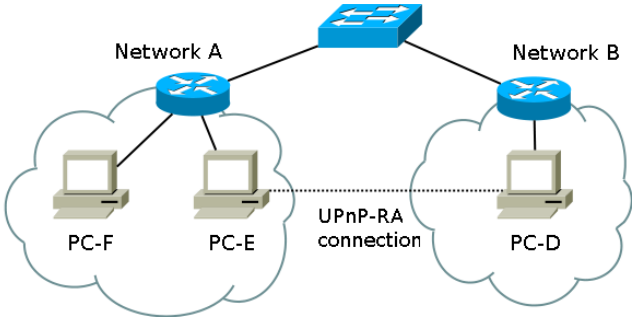

Figure 4. UPnP Remote Access setup

In Figure 4 you can see the setup for the remote access case. It has two home networks A and B connected with each other using UPnP Remote Access. PC-D is doing the service translations for Network B and PC-E for Network A. PC-F is running the CUPS print server. Time measurements were performed using logging information and wireshark.

C. Network load evalution

1) SDP module network load: Every SDP module has the responsibility to look for services in its own network, to offer to the framework for translation. As stated before this can be done using active or passive discovery. Our DNS-SD SDP module uses active discovery on startup, to find the currently available services and afterwards switches to passive discovery. The network traffic added in this case is limited to the few multicast packets for searching all services and their responses at startup. As SLP does not support passive discovery, our SDP module stays polling for new or removed services. When choosing a small polling interval, the extra traffic will be larger. A large polling interval will introduce delay between the introduction of a new service and the translation to all SDP networks. Table II shows some numbers on the load for different polling intervals and different number of available services.

	#SLP services				
delay	0	1	2	3	4
4 s	57 103	99 817	126 079	144 925	162 253
6 s	56 586	92 093	126 483	144 403	150 649
8 s	52 423	80 955	119 332	128 149	146 440
10 s	52 388	80 955	144 825	128 149	134 634

Table II
SLP NETWORK LOAD (BYTES/S)

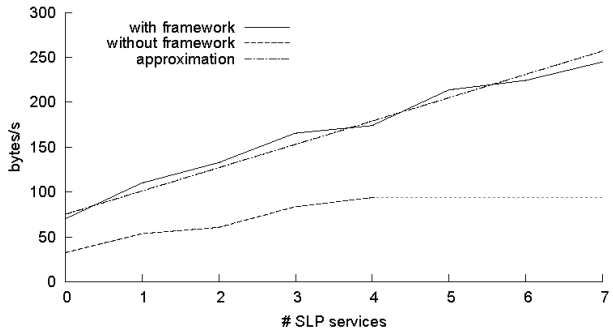

Figure 5. framework network load

2) framework network load: After services are discovered, they are translated to all other SDP networks supported by the framework, which causes additional traffic as well. In this section we will compare the total service traffic, including the traffic generated by the framework, to the situation where no framework is running. On the CUPS server on PC-B (see Figure 3) we're running 4 SLP printers and 3 DNS-SD printers. On PC-B we're polling for SLP printers (with an interval of 10 seconds). In a first stage we measure the SLP traffic without the translation framework. In a second stage we run the framework with the SLP and DNS-SD module started on PC-A.

In Figure 5 you can see the measured traffic in function of the number of active services. When the framework is active the total amount of traffic is linear to the number of services and has roughly doubled in comparison to the situation without the framework.

3) conclusion: The network load for discovery depends on whether a certain SDP can discover the services passively and the polling interval. In both cases the additional traffic is very limited. The total extra traffic depends on the number of services that need to be translated. If no services need to be translated the discovery traffic is doubled (using active discovery). In case there is translation necessary this factor will augment, since the virtual services all contribute to the additional network traffic.

interval	average delay
4 s	5 268 ms
8 s	6 322 ms
10 s	6 769 ms

Table III
SLP DELAYS

test	delay
#1	36 ms
#2	17 ms
#3	33 ms
#4	19 ms
#5	25 ms

Table IV
TRANSLATION DELAY

D. Delay evaluation

1) delay between SDPs: The delay we want to depict here is the one between the announcement of a service in an SDP network and the discovery of that same service in another SDP network after translation of the framework. On the LAN setup we tested 2 scenario's: an SLP printer is announced on PC-B and is discovered with DNS-SD on PC-C with the framework running on PC-A; in the second scenario we announced an DNS-SD printer and discovered it with SLP. The delay discussed here can be split up in 3 parts: the delay for discovery, the translation of the service and the delay for publication. Discovery can introduce delay based on the used method (passive or active).

In Table III we show some averages for the SLP delays for several polling intervals. We measured an average of 1.8 s for the discovery of a DNS-SD service.

Table IV shows some test results for the translation of a service. The translation step only converts the description discovered by the fist step to the framework representation of the service, so the delay of this step is negligible. The delay of the publication step is similar to the discovery delay, only this time the framework is the publisher and the service consumer is the discoverer.

2) delay between home networks: Apart from the delay introduced by sharing services between SDP networks, the delay coming from UPnP-RA has to be taken into account. UPnP-RA creates a secure tunnel between the homes, and makes UPnP services in Network A (see Figure 4) available in Network B (and vice versa). The delay UPnP-RA introduces comes down to three parts: the discovery of the service in A, the transmission from A to B and the publication in B. Since UPnP supports active discovery, the transmission step will introduce the biggest delay. If the original service was UPnP, no additional translation step has to be taken into account.

3) conclusion: The translation delay is negligible compared to the SDP protocol specific discovery and publication delays. The biggest chunk of the delay when sharing services

between home networks comes from the transmission between the networks, and is highly dependent on the distance and network infrastructure between these networks.

VI. Conclusion

Currently, multiple service discovery protocols exist to enable automatic discovery of services in local networks. They are however not interoperable, and usually limited to a single subnetwork. The framework we propose enables for automatic translation between SDPs, and additionally features remote access to allow ubiquitous sharing of services over network boundaries. We successfully demonstrated this concept in a proof-of-concept realization. Performance assessment based on measurements showed that the framework overhead is minimal, and no perceptible performance degradation could be observed when comparing use of services native to a certain SDP versus services translated to that particular SDP by our framework.

Acknowledgment

Part of this work was supported by the IWT Q-MATCH project.

C. Develder is supported by the Research Foundation – Flanders (FWO–Vl.) as a postdoctoral fellow.

References

[1] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan, "Service Location Protocol," RFC 2165 (Proposed Standard), Internet Engineering Task Force, Tech. Rep. 2165, June 1997, updated by RFCs 2608, 2609. [Online]. Available: http://www.ietf.org/rfc/rfc2165.txt

[2] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-OSGi: Distributed Applications Through Software Modularization," in *Middleware*, 2007, pp. 1–20.

[3] A. Presser *et al.*, "UPnP Device Architecture," 15 Oct. 2008. [Online]. Available: http://www.upnp.org/resources/documents.asp

[4] J. Ritchie and T. Kühnel, "UPnP AV Architecture:1," 25 Jun. 2002. [Online]. Available: http://www.upnp.org/specs/av/default.asp

[5] UPnP Forum, "UPnP QoS." [Online]. Available: http://www.upnp.org/specs/qos

[6] L. McLaughlin, "RFC 1179: Line printer daemon protocol," RFC 1179 (Informational), Internet Engineering Task Force, Tech. Rep. 1179, Aug. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1179.txt

[7] N. Goeminne, K. Cauwel, F. De Turck, and B. Dhoedt, "Deploying QoS sensitive services in OSGi enabled home networks based on UPnP," in *Proc. Int. Conf. on Internet Computing (ICOMP2006)*, Las Vegas, NV, USA, 26-29 Jun. 2006.