Web Services in Ad-Hoc and Mobile Infrastructure Bootstrapping with Service Discovery

Pekka Vuorela
Jussi Saarinen
Risto Pitkänen

28th February 2006

Contents

1 Introduction

The purpose of the WeSAHMI project is to study how personalized services can be offered and used in an ad-hoc network. The type of service is concentrated on web pages, possibly enhanced with new technologies to enable more interactive applications.

This document describes the basic environment of operation and the needs this kind of system sets to service discovery technologies. A few possible documents are evaluated for their suitability. The idea is to use service discovery system for bootstrapping the network so that the mobile nodes can register themselves to be available. SIP seems like the protocol of choice for the registrations at the moment.

The rest of the document is structured as follows. Section 2 discusses the overall environment of operation. and section 3 lists some requirements for service discovery protocol. Section 4 evaluates the suitability of some available service discovery protocols and their implementations. Conlusions are found on the section 5.

2 Environment

2.1 Description of environment

The environment of the WeSAHMI project will consist of an ad-hoc network with mobile nodes connected to one or multiple fixed networks via gateways. An example is sketched in figure 1. The purpose is to study how the fixed network services can be offered and "pushed" to the mobile nodes. The system should allow services to be offered to certain individuals or groups instead of everyone in the ad-hoc network. The main type of service are web pages, possibly enhanced with new techonologies to enable more interactive applications.

The network is IP based using either IPv4 or IPv6. IPv4 autoconfiguration method is described in section 2.2. The connectivity to the external server is enabled with a gateway node. It can be made on the IP layer using NAT or giving ad-hoc nodes extra addresses. Or alternatively it can be accomplished on application layer using proxy servers on the gateway nodes.

The first step in experimenting the service pushing is to add SLP functionality to the gateway node and create web pages on the server. The URLs of these pages are registered the gateway which advertises them to the ad-hoc network. This kind of service advertising does not support personalized services, they are all advertised to everybody. One possibility could be using service attributes to list the target users or groups, but it might not be reasonable way. Several protocols use MTU as the maximum size of packet and it would not be sufficient for large number of target users.

On the next step an SIP server might be added to the gateway node. This would enable registering mobile nodes to it using identities and the gateway node to call back on the mobile nodes to invite them to services.

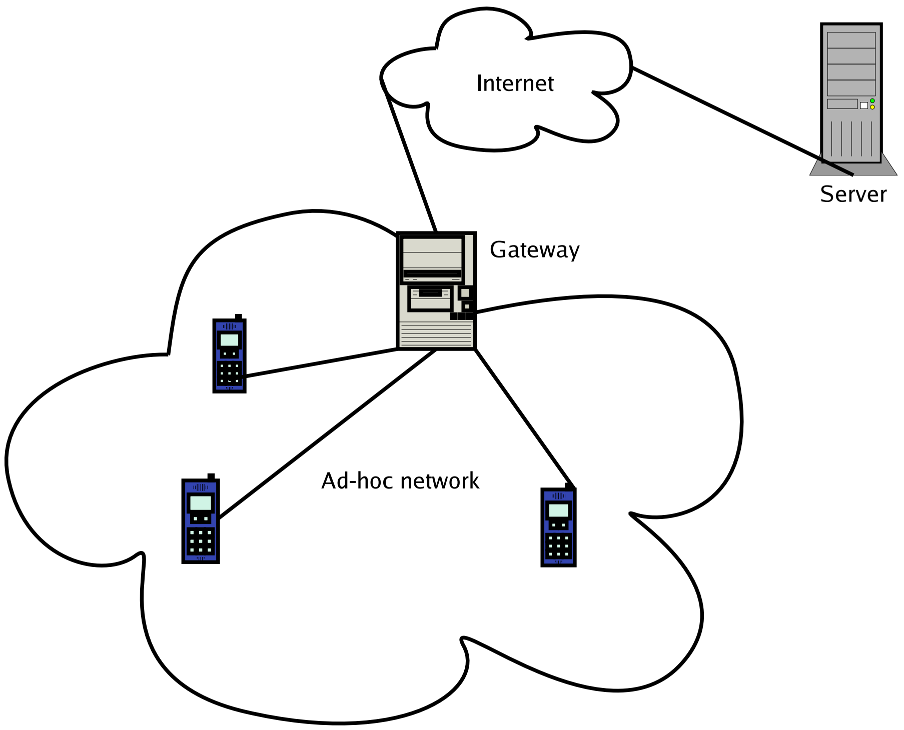

Figure 1: Example WeSAHMI network

2.2 Dynamic configuration of IPv4 addresses

2.2.1 Overview

The dynamic configuration of IPv4 link local addresses is specified by RFC 3927 [2]. It defines a mechanism for how a node on the network can configure an IPv4 address without the use of external services such as DHCP server. The address is defined to be link local which means that it can be used to communicate with nodes in the same link. Being on the same link means that the nodes can communicate directly with each other so that the link-layer packet payload arrives unmodified. Address block 169.254/16 is registered for link local addresses.

2.2.2 Selecting an address

The basic mechanism for obtaining an address is following:

1. Selection – a host selects an address using a pseudo-random number generator with a uniform distribution in the range from 169.254.1.0 to 169.254.254.255 inclusive.
2. Probing – the host tests if the selected IPv4 link-local address is already in use. On a link-layer such as IEEE 802 that supports ARP [14], conflict detection is done using ARP probes. The ARP probes query which host has a specific IP address and the owner responds to it. The probe is repeated a few times with small delays. If any host claims to

own the address by sending an ARP reply, or tries to find out the owner in similar manner, the address is considered taken and the algorithm returns to previous state to select a new address.

3. Announcing – When a host has found an available IP address it announces the new address to the network. It broadcasts a number of ARP announcements. The announcements are like probes but the sender and target IP addresses are both set to the host's newly selected IPv4 address.

Example is shown in Figure 2.

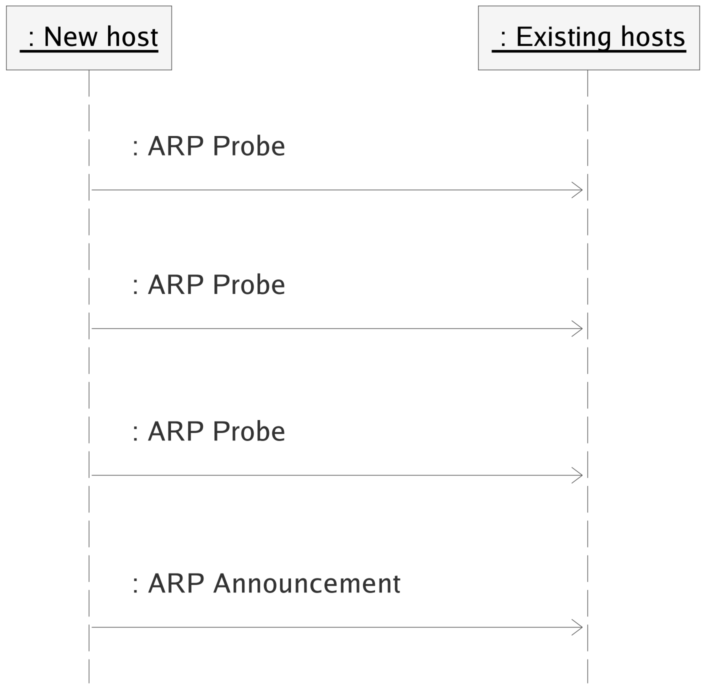

Figure 2: New host selects an address. All messages are broadcasted.

2.2.3 Conflict detection

Address conflict may occur at any time of the operation, not only during the address selection. For example, two hosts may select the same address when they don't have connectivity between them and later become aware of each other. At any time, if a host receives an ARP packet on an interface where the 'sender IP address' is the IP address the host has configured for that interface, but the 'sender hardware address' does not match the hardware address of that interface, then this is a conflicting ARP packet, indicating an address conflict.

On a conflict, a host has two possible options.

- the host may immediately configure a new IPv4 link-local address.

- the host may attempt to defend its address by recording the time that the conflicting ARP packet was received, and then broadcasting one single ARP announcement, giving its own IP and hardware addresses as the sender addresses of the ARP. Having done this, the host can then continue to use the address normally without any further special action. However, if this is not the first conflicting ARP packet the host has seen, and the time recorded for the previous conflicting ARP packet is recent, the host must immediately cease using this address and configure a new IPv4 link-local address.

Forced address reconfigure may be disruptive, causing TCP connections to be broken. However, it is expected that such disruptions will be rare. Before abandoning an address due to a conflict, hosts should actively attempt to reset any existing connections using that address.

2.2.4 Available solutions

A simple implementation [15] by Arthur van Hoff is available on the Internet. It uses an external shell script which it calls when the address needs to be changed. The program itself can be run without modifications and it does not depend on external libraries or such.

3 Requirements for Service Discovery

The primary requirement for service discovery (SD) mechanism used to bootstrap any service is the ability to push service location information from server to a client residing in ad-hoc network. It should be able to do this also although the address of the client is previously unknown. The client side implementation should be non-blocking in the sense that the application should be able to function normally while still receiving updated service information through the push mechanism. A secondary requirement for the mechanism is to enable easy configurability by allowing remote regisrations of services from the fixed network. Though, if the only purpose of the SD mechanism is to bootstrap a static SIP Proxy, external registrations are not absolutely essential. Furthermore the implementation of the mechanism should be light enough to run in mobile devices with limited resources. Issues such as generated network overhead, cpu utilization, and battery consumption should all be taken into account.

4 Protocols

4.1 Web Services Dynamic Discovery

4.1.1 Overview

We Services Dynamic Discovery (WS-Discovery) is a multicast discovery protocol to locate services. It is specified by Microsoft and its currently under development. The specification is provided for use as-is and for review and evalution only. The protocol is built on top of Web Services which means the messages are sent using SOAP over UDP or TCP, and are formed using XML.

WS-Discovery entities are client, target service and discovery proxy. Target services are entities that make themselves available for discovery. Discovery proxies relay service information on behalf of the target services. They are an optional component in the architecture. The primary mode of discovery is a client searching for one or more target services. To find a target service by the type of the target service, a scope in which the target service resides, or both, a client sends a probe message to a multicast group. Target services that match the probe send a response directly to the client. To locate a target service by a name, a client sends a resolution request message to the same multicast group, and again, the target service that matches sends a response directly to the client. [12]

To minimize the need for polling, when a target service joins the network, it sends an announcement message to the previously mentioned multicast group. By listening to this multicast group, clients can detect newly-available target services without repeated probing. [12]

The messages used in the WS-discovery are following:

Hello A message sent by a Target Service when it joins a network.

Bye A Message sent by a Target Service when it leaves a network.

Probe A message sent by a Client searching for a Target Service by Type and/or Scope.

Probe match Reply to a Probe message.

Resolve A message to resolve Web Services address to transport address (i.e. DNS name or IP address).

Resovel Match Reply to a Resolve message.

Messages sent by the client and target service are shown in Figure 3.

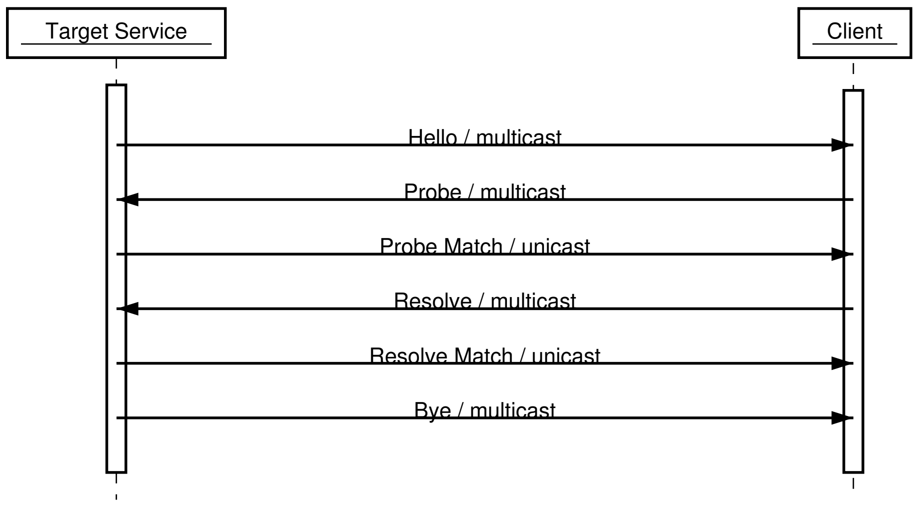

Figure 3: WS-Discovery functions [12].

4.1.2 Proxy support

To scale to a large number of endpoints, the WS-Discovery defines a discovery proxy entity. When a discovery proxy detects a probe or resolution request sent by multicast, the discovery proxy sends an announcement of itself. By listening for these announcements, clients detect discovery proxies and switch to use a discovery proxy-specific protocol. However, if a discovery proxy is unresponsive, clients revert to use multicast protocol.

Figure 4 shows how a client probes for services and becomes aware of a Discovery Proxy which it uses to get the rest of the service information.

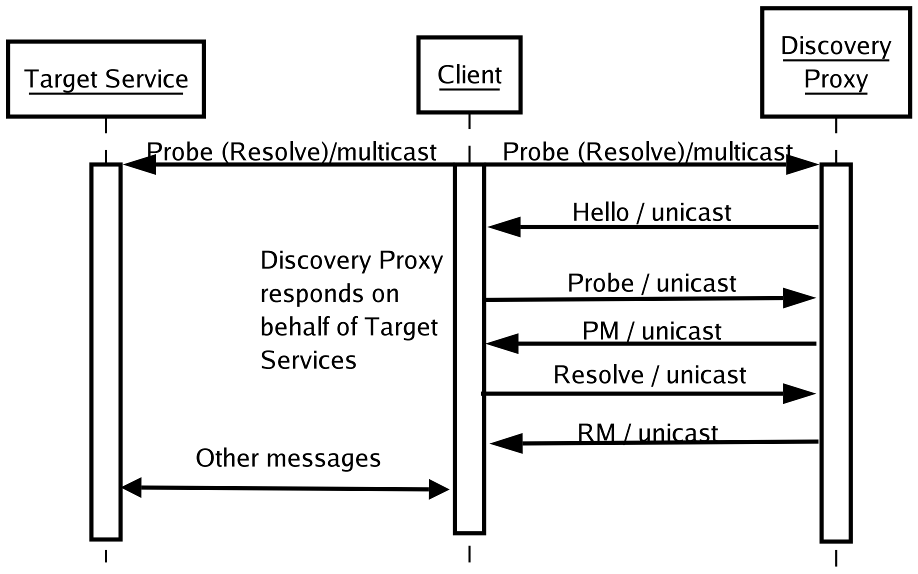

Figure 4: Discovery Proxy message exchanges [12].

Pointers to Research Groups And Open Source Implementations

There does not seem to be existing available implementations of Web Services Dynamic Discovery as the protocol is still quite new and is having changes. The following Open Source implementations for generic Web Services libraries are available:

- **gSoap** available at [8].
- **Axis** available at [1].
- **Java Web Services Developer Pack** available at [10].

4.2 Service Location Protocol

Service Location Protocol [9] is a protocol specified by IETF. Its open source c implementation OpenSLP version 1.0.11 [13] was used in the service discovery scheme of SESSI project, the predecessor of WeSAHMI.

SLP includes three entities that perform service discovery functions:

- User Agents (UA) perform service discovery.
- Service Agents (SA) advertise the location and attributes of the services.
- Directory Agents (DA) store and distribute service information.

The first example shows how an UA multicasts a service request and a SA replies with unicast. In the second example, the UA has found a DA and uses it as a proxy to find services.

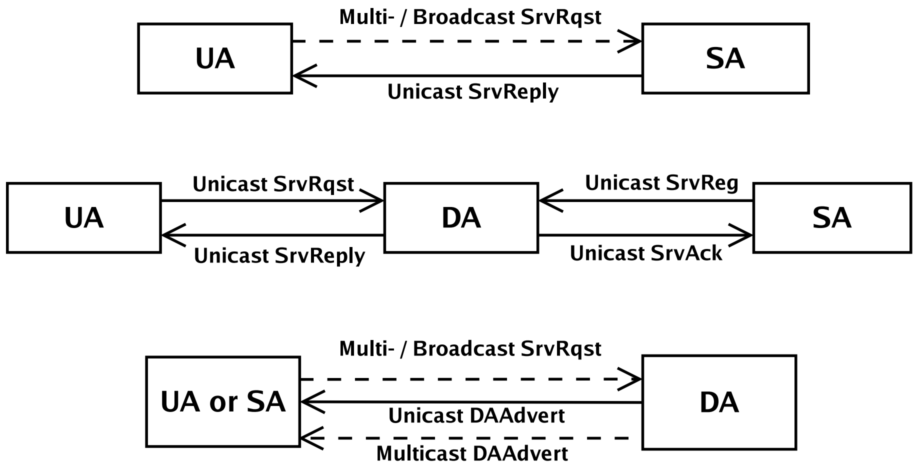

Figure 5: SLP agents and most common protocol messages.

The original SLP provides service location functionality that is based on UAs sending broadcast requests to SAs, or if DA is available, unicast requests to it. The only push -functionality included in the unmodified protocol is SAs and DAs ability to advertise themselves with SA and DA Advertisement messages.

Within SESSI project a Passive Discovery (PD) functionality for SLP was implemented. It provides a method for SA initiated service discovery: UAs may listen to advertisements of services that have a certain service ID and SAs may periodically advertise their services with broadcast Service Advertisement messages. The operation of PD is illustrated in Figures 6 and 7. It provides a method for the Wesahmi framework to push service information from servers to nodes in the ad-hoc network. However if remote registrations are required enabling them will require changes to the original scheme. Currently the PD registrations are stored in a different data structure as the normal SLP registrations and they can be made only locally. The changes that are required include modifications to service registration messages, to mark the difference between active and passive registration, and modifications to slpd that enable it to process them accordingly. This could be done by adding a new message type that would be then processed separately. If security features are to be used, a specifically crafted extension block could be used to carry the necessary information.

4.3 UPnP

UPnP is an architecture [4] for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. It specifies device addressing, service location, device control, eventing and presentation. The protocols are based on TCP, UDP, HTTP and XML. UPnP is being developed by the **UPNP Forum** [5], which is an industry initiative

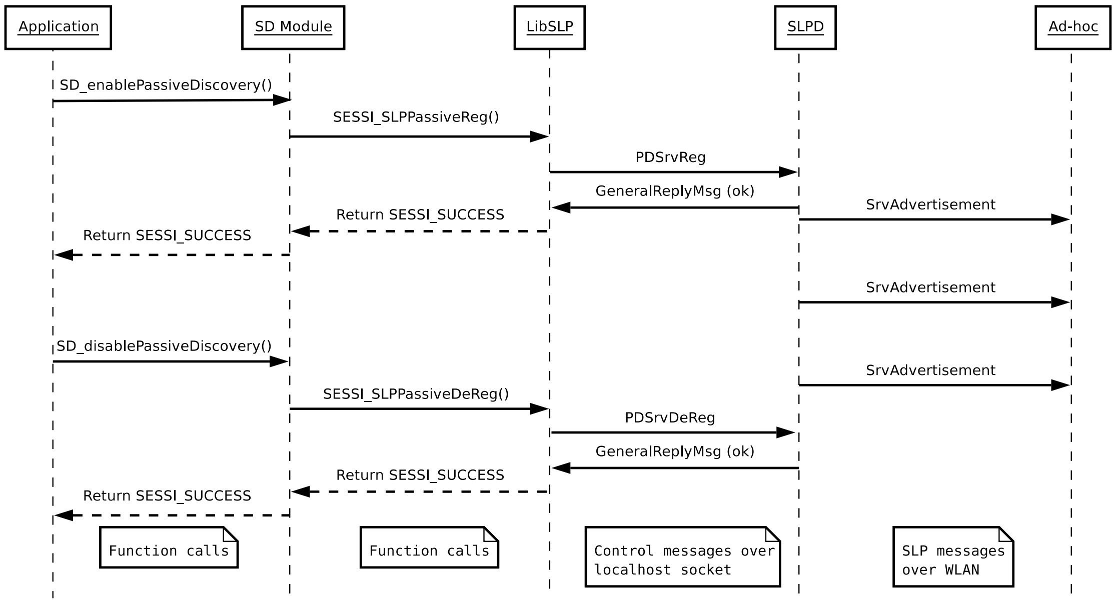

Figure 6: Registering and unregistering services for PD

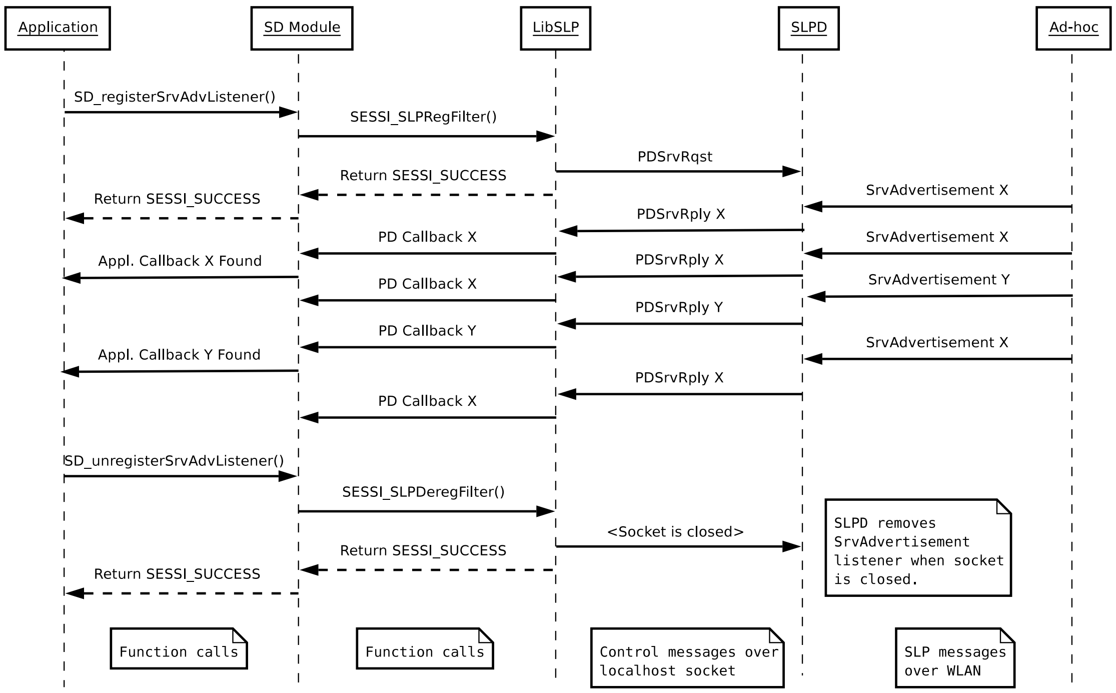

Figure 7: Discovering services with PD

headed by Microsoft.

SSDP, Simple Service Discovery Protocol, is the service discovery component of the UPnP architecture. It's functionality is illustrated in Figure 8. Discovery occurs when a SSDP client, e.g. Control point, multicasts a HTTP UDP discovery request to the SSDP multicast channel/-Port. SSDP services listen to the SSDP multicast channel/Port in order to hear such discovery requests. If a SSDP service hears a HTTP UDP discovery request that matches the service it offers then it will respond using a unicast HTTP UDP response. SSDP services may send HTTP UDP notification announcements to the SSDP multicast channel/port to announce their presence. Hence two types of SSDP requests will be sent across the SSDP multicast channel/-port. The first are discovery requests, a SSDP client looking for SSDP services. The second are presence announcements, a SSDP service announcing its presence. [6]

The presence announcements of SSDP could be used in Wesahmi to enable the SIP bootstrapping.

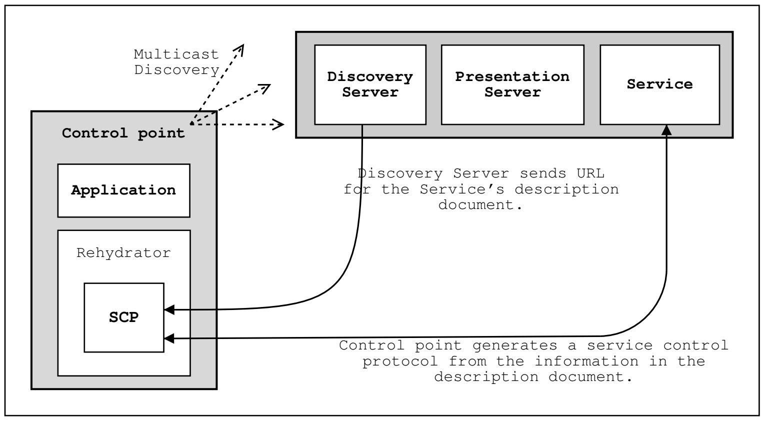
Multicast Discovery
Discovery Server
Presentation Server
Service
Control point
Application
Rehydrator
SCP
Discovery Server sends URL for the Service's description document.
Control point generates a service control protocol from the information in the description document.

Figure 8: UPnP entities and some service discovery messages [7]

Atleast two notable open source implementations of UPnP are available: Linux SDK for UPnP Devices [11] and Cyberlink for Java [3]. Both of these have been developed actively for several years and follow the original UPnP specification. Thus they support presence announcement mechanism but will most likely require an extension to support remote service registrations. To enable this functionality with the Linux SDK use of a separate daemon that listens for registrations and then uses API to register a device that provides the service could be one solution. If the Java implementation is used it could possibly be enhanced with the Java's Remote Method Invocation (RMI) system thus enabling remote device/service registrations to be made.

5 Conclusions

In this document we discussed the run-time environment, requirements for the used SD mechanism, features of three different service discovery protocols and their capability to support service information pushing. The environment was determined to consist of ad-hoc network and a fixed server with autoconfigured IPv4 addresses.

SLP that has been extended with passive discovery functionality in the SESSI project was deemed to be the most suitable solution for the target environment. The WS-Discovery protocol has no open source implementations and is therefore an infeasible option. UPnP has several open source implementations but since, in comparison to SLP, it provides similar functionality with added complexity it was not selected.

References

[1] Axis website. At http://ws.apache.org/axis/, February 2005.

[2] Stuart Ceshire, Bernard Aboda, and Erik Guttman. Dynamic configuration of link-local ipv4 addresses. RFC 3927, Internet Engineering Task Force, March 2005.

[3] Cyberlink for java website. At http://www.cybergarage.org/net/upnp/java/, February 2006. Development package for UPnP devices.

[4] UPnP Forum. *UPnP Device Architecture version 1.0*, June 2000.

[5] UPnP Forum. Upnp forum website. At `http://www.upnp.org/`, April 2004.

[6] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright. Simple service discovery protocol/1.0. Technical report, October 1999.

[7] Richard III G. Golden. Service Advertisement And Discovery. *IEEE Internet Computing*, 2000.

[8] gsoap website. At http://www.cs.fsu.edu/ engelen/soap.html, February 2005.

[9] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, IETF, June 1999.

[10] Java web services developer pack website. http://java.sun.com/webservices/jwsdp/index.jsp, February 2005.

[11] Linux sdk for upnp devices. At `http://upnp.sourceforge.net/`, February 2006. An Open Source UPnP Development Kit.

[12] Microsoft. *Web Services Dynamic Discovery specification*, April 2005. Available at `http://msdn.microsoft.com/ws/2005/04/ws-discovery/`.

[13] OpenSLP Project Group website. At `http://www.openslp.org`, April 2004.

[14] David C. Plummer. An ethernet address resolution protocol. RFC 826, November 1982.

[15] `http://www.zeroconf.org/AVH-IPv4LL.c`.