A Distributed Architecture for Remote Service Discovery in Pervasive Computing

by

Farzad Salehi

A thesis submitted to the

Department of Computer Science

in conformity with the requirements for

the degree of Master of Science

Bishop's University

Canada

December 2011

Abstract

The area of investigation of this dissertation is service discovery in pervasive computing. Service discovery is very important in realizing the concept of pervasive computing. In addition, service discovery protocols must be able to work in the heterogeneous environment offered by pervasive computing. Remote service discovery in particular has not been properly achieved so far.

In an attempt to remedy this we propose a new architecture for enabling typical (local) service discovery mechanisms (without the ability of remote service discovery) to discover services remotely. Our architecture uses Universal Plug and Play (UPnP) as a prototype for normal (local) service discovery protocols, and Gnutella as a prototype for peer to peer distributed search protocols. We introduce a module called *service mirror builder* to the UPnP mechanism, and a remote communication protocol over a Gnutella network. As a consequence, UPnP networks become able to discover services in remote networks (that is, remote service discovery).

Acknowledgments

I would like to thank my supervisor Dr. Stefan D. Bruda for his valuable support in this work. I appreciate his calmness and patience with me. He inspired me and gave me confidence. Without his help and feedback this dissertation could not have been completed.

I would also like to thank Dr. Abbas Malekpour from the University of Rostock and Ph.D. student Yasir Malik from Université de Sherbrooke. Their advice and feedback helped me in writing this dissertation.

I would like to thank English-speaking missionaries and friends, in particular Sister Jean Jensen, who helped me improve my English writing skills and edited some of the text. Also, thanks to Dr. Stefan D. Bruda for his help with English.

Last but not least, I express my deepest love and appreciation to my family for their love and support. I especially thank my father, my mother and my sister Foroogh for having faith in me and for their continuous encouragement.

Contents

List of Figures

Chapter 1

Introduction

This chapter introduces the context (pervasive computing) as well as the main subject (service discovery) of our research. We also outline our objectives and the way we addressed them, together with a motivation for our work.

1.1 Pervasive Computing

In 1988 Mark Weiser gave birth to the vision of anytime, anywhere computing or "ubiquitous computing." He defined it as follows: *Ubiquitous computing is the method of enhancing computer use by making many computers available throughout the physical environment, but making them effectively invisible to the user* [47]. The concept of ubiquitous computing is also known as "pervasive computing" (which we use throughout this dissertation) or "ambient intelligence."

Computing anytime, anywhere, and in any device means a massive presence of computing devices in the physical world. At the same time, people should be able to access information and computation in a user centric way i.e., user interaction with such a system must be natural and comfortable. Pervasive computing is thus a migration from desktop computing to computing integrated into everyday objects. Its ultimate goal is a seamless computing process all around users in a way that users do not even recognize as computing.

Pervasive computing is a vast concept that encompasses numerous other concepts such as smart objects, smart environment, smart home (or connected home), system infrastructure, user interface, embedded systems, artificial intelligence, speech recognition, distributed computing, mobile computing, sensor networks, and so on. One important feature of pervasive computing is context awareness, which is used to design new user interfaces. User interfaces thus designed allow humans to communicate with computing devices in the same or very similar way they (humans) communicate with each other. One particular technology that has helped the idea of pervasive computing substantially is the wireless network technology. Wireless networks have effectively realized almost anywhere computing.

Overall the best definition for pervasive computing would arguably be: *machines that fit the human environment instead of forcing humans to enter theirs* [49].

One possible application of pervasive computing is a smart environment. An instance of such an environment is the smart apartment located in the DOMUS Laboratory at Université de Sherbrooke [15]. This is an apartment saturated with sensors, actuators and cameras, all being administrated by a powerful computer server. The ultimate goal of the DOMUS researchers is to refine this apartment so that it becomes suitable for disabled people. Sensors and cameras can monitor health and physical presence of a disabled or aged person. Sound recognizers can understand that person's questions and then help him in situations such finding things, opening doors, close the window shades, and so on.

1.2 Service Discovery

Pervasive computing offers an environment saturated with sensors, actuators, cameras, and all other sorts of computing devices. One of their goals is assisting humans in their everyday lives. For sure not every person in this world is a computer expert, so these devices should work together and satisfy users' needs with minimal user intervention.

Service discovery protocols are computer network protocols that allow devices to detect each other and their services. Service discovery protocols are evolving day by day and their number is perpetually increasing due to the increasing need for automation. The need for automation also causes the number of computing devices to keep growing. Many technologies such as wireless networks, embedded devices, artificial intelligence, etc. are speeding up the presence of computing in almost every object around humans. For sure detection, configuration, and interoperability between these devices could not be managed manually so that there is a perpetual need for the automation in this respect. Service discovery protocols have been designed to carry this burden. They help different applications, services, electronic and computing devices to recognize each other and work with each other with minimal or no human intervention.

Service Discovery protocols are designed to minimize administrative overhead. They also simplify programming and coding, because designers do not need to foresee all possible interactions between devices and services at design time.

Many service discovery protocols have been designed. Most of them are service discovery and control protocols (service control being the next phase after discovering a service; it facilitates the invocation of the discovered service by a control point or controller). We summarize the dominant protocols, briefly describing their special features.

Microsoft *Universal Plug and Play* (UPnP) [44] uses the Simple Service Discovery Protocol (SSDP) [45] to discover devices and services. The ground base for UPnP was Plug and Play (PnP) [44]. The goal of UPnP is the automatic discovery and configuration of any new devices connected to a computer network. UPnP supports zero configuration networking or Zeroconf. That is, UPnP creates an IP network without any need of manual configuration and configuration servers.

The UPnP forum promotes UPnP technology in compatibility with common device architecture contributed by Microsoft [44]. UPnP uses the Internet protocol suite: TCP/IP, HTTP, SOAP and XML [36, 44].

The *Bluetooth Service Discovery Protocol* [7] governs the interaction between a service discovery protocol server and a client. A list of service files that describe the properties of services linked with the server is maintained by the server. Each service record contains information about a single service. A client may recover data from a service record maintained by the service discovery protocol server by issuing a service discovery protocol request [7].

If the client, or a request linked with the client, decides to use a service, it must open a different connection to the service provider so that it can make use of the service. Only one service discovery protocol server per Bluetooth device is allowed. However, if a Bluetooth device serves only as a client, then it does not require any service discovery protocol server. One Bluetooth appliance can work both as client and server. In the event that there are several applications providing services on a device, a service discovery protocol server may be offered by the device to manage requests for information about the provided services [7]. Similarly, several client applications can use a single service discovery protocol client to inquire servers on behalf of the client applications.

The set of service discovery protocol servers that are accessible to a client may vary substantially based on the RF proximity of the servers to the client. If a server becomes available, a probable client is required to be alerted in a way outside the service discovery protocol so that the client can implement service discovery and question the server regarding its services. Likewise, if a server leaves proximity or otherwise becomes inaccessible, then the service discovery protocol provides no notification mechanism. Nevertheless, the client may use service discovery to poll the server. If the server no longer reacts to requests, then the client may conclude that the server is not available [7].

Apple's solution for zero-configuration networking over IP is *Bonjour* [3] (formerly known as *Rendezvous*), which uses DNS service discovery to discover services. It is derived from the operations of the ZEROCONF Working Group, a division of the Internet Engineering Task Force (IETF). There are three areas covered by the ZEROCONF Working Group's provisions and proposed solutions for zero-configuration networking over IP [3]:

1. *Addressing*, or the allocation of IP addresses to hosts;

2. *Naming* or the use of specific terms to refer to hosts instead of IP addresses;

3. *Service discovery* or the automatic location of services on the network.

While making progress in the area of ease of use, Bonjour permits service providers, hardware manufacturers, and application programmers to maintain a single network protocol (IP). No longer do network users have to assign IP addresses or host names, or even enter in names to retrieve services on the network. Instead users can merely ask which network services are available and then choose from the given list.

This type of browsing is more effective for applications than it is for users. Without the requirement of user intervention, applications can automatically identify needed services or other applications with which they can collaborate, thus allowing automatic connection, communication, and data exchange [3].

Sun's *Jini technology* or *Apache River* [43] uses the advantages of Java technology for service discovery and control. Java and Jini together provide flexibility. The Jini service discovery is based on Service Oriented Architecture (SOA). SOA is an architectural framework and approach that takes common business applications and break them down into individual business functions called services (that is, well-defined business functions that are built as software components); these services are then used as building blocks for the final software [6, 10] . In a Jini architecture there are three main parts: client, server and lookup service. [9, 30, 48].

Network-enabled devices that come with an easy to use procedure for finding available services are provided by University of California at Berkeley's *Ninja Service Discovery Service* (SDS) [24]. The service is directory-style and presents a contact point for complex queries against cached service descriptions presented by services. The behaviour of the service is automatically adapted to deal with the failure of both SDS servers and services, and conceals the complexities of fault recovery from client applications. The security-minded service discovery service can ensure that all contact between components is protected, and it helps in verifying the reliability of specific services [24].

Certain advantages are provided to this service discovery protocol through the use of XML to encode service descriptions and client queries. Service providers can capitalize on the extensibility of XML by creating service-specific tags to more adequately depict the services they provide. Similarly, XML takes advantage of semantic-rich service descriptions to enable clients to issue more powerful queries [24].

IBM Research's *DEAPspace* [32] is especially suitable for changing Ad Hoc environments [32].

The *Salutation Architecture* [38] was provided to solve the problems of service discovery and application among a wide set of devices and in an environment of extensive connectivity and mobility [38]. The architecture offers a standard procedure for applications, services, and devices to portray and to promote their abilities to other applications, services, and devices and to discover their capabilities. The architecture also allows the exploration of other applications, services, or devices for a specific ability, and to ask for and to launch interoperable sessions with them to access their capabilities [38]. The diverse nature of target appliances and equipment in a setting of extensive connectivity makes the architecture independent of processors, operating systems, and communication protocols. Implementation is feasible even in lower-priced devices [38].

The *Service Location Protocol* (SLP) [22] by the Internet Engineering Task Force supports a structure in which client submissions are exhibited as “User Agents” and services are promoted by “Service Agents.” A “Directory Agent” offers accessibility to the protocol. The User Agent distributes a “Service Request” (SrvRqst) on behalf of the client application, detailing the traits of the needed service. The User Agent will obtain a Service Reply (SrvRply) detailing the position of all the services in the network which satisfy the request [32].

The Service Location Protocol structure allows a User Agent to directly send requests to the Service Agents. The request in this instance is multicast. Service Agents that receive a request for a service which they advertise can unicast a reply containing the location of the service. One or more Directory Agents may be used in bigger networks. A Directory Agent operates as a cache [32].

1.3 Challenges for Service Discovery Protocols to Support Pervasive Computing

Available service discovery protocols are designed for home or enterprise environments [50]. The pervasive computing environment is however far more heterogeneous and sophisticated than any home or enterprise one. In particular, in such an environment applications from different vendors and platforms have to work together in a seamless way. The following are the three arguably most important challenges faced by a service discovery protocol in a pervasive computing environment. Challenges faced by service discovery protocols in a pervasive computing environment are more numerous; however, security and interoperability are extremely important and remote service discovery is the main focus of this dissertation beside being important as well.

Security The most challenging problem in pervasive computing is security and privacy. Integration of computing devices with people and their environment poses many questions concerning the privacy of the people and their personal information [50]. The integration of computing devices with people means that substantial information from those people must be stored in digital forms. In other words, digital devices could have a more or less complete image of people. For sure, service discovery protocols can access this information through different available services and devices. For example, it is possible to track a person with an RFID tag on their clothes or with the devices carried in her handbag. As a further example, a person's health information can be disseminated through discovering a wearable medical device [50].

Many established security solutions fail in new environments simply because of the environments themselves [2]. Indeed, most security solutions are defined for particular environments (enterprise, home, public networks, etc.). Each one depends on particular facilities in that environment (for example the existence of firewalls in an enterprise). However in pervasive computing heterogeneity and the absence of boundaries are two inherent characteristics. Therefore strong security and privacy solutions are fundamental requirements that must be met by service discovery protocols in order to be usable in pervasive computing environments [50].

Interoperability One of the biggest challenges for service discovery protocols in a pervasive computing environment is their work domain. Service discovery protocols are primarily designed to work in a local area network or a single hop wireless communication network [50]. However the very definition of pervasive computing encompasses all kinds of networks that are connected and working everywhere. In a pervasive computing environment each device and service may be connected to one or more networks; service discovery protocol for discovering and controlling them have to be able to work with different networks and protocols. The available service discovery protocols have been designed to some degree to accommodate heterogeneity, but because each one is designed to connect some type of service or device, it is unlikely that they can work in an environment with all sort of platforms and devices. Different vendors support different service discovery protocols; this factor too decreases interoperability [50].

Another problem is semantic interoperability. Different vendors and discovery protocols are using different terminologies. For example in a service description of a video projector two commands may be labelled “ON” and “OFF,” but in the controller of the smart phone these two words may appear as “TURN ON” and “TURN

OFF." As another example some words may appear in different devices with the same spelling but different meanings [50].

Ontology engineering thus has a very important role in the increase of semantic interoperability. Designing a new service discovery protocol according to most popular and vendor supported protocols can also be a very realistic solution to increase interoperability.

Remote service discovery Most of the service discovery protocols have been designed for local networks and they are not able to discover remote services. Remote services are services not present on the local area network (LAN) but in another network or LAN.

Many services in a pervasive computing environment are physical-oriented services, meaning that their services are useful for the users in the same physical environment and not for distant users. As an example, a video projector or coffee machine controller can service only the users on premise. Still, many other services are not physically-oriented and they can be accessed and have to be accessed by users physically far away from them. Overall a pervasive computing environment must access some services that are far away from its physical environment, such as accessing the digital data in someone's home, remotely monitoring sensors, actuators and cameras present in a place for security or health care purposes, and so on. These needs and examples of remote service discovery will be further discussed in Chapter 1.5.

Computing anywhere is also the very definition of the concept of pervasive computing. While it is not possible to provide all services anywhere, remote access to any services makes sense and could be realized. For this purpose service discovery protocols must be able to discover services remotely in order to be able to work in a pervasive computing environment.

1.4 Contribution

Most service discovery protocols (such as UPnP) are designed to work only in a local area network (LAN) [5]. In other words, they can discover only services which are available on their LAN. As we mentioned in Section 1.3, there is a need to discover services located in different LANs (called remote services).

A combination of existing technologies and services can enable some level of remote access, such as remote file access or remote control of the console of a computer. However, seamless service discovery and control of remote UPnP services is currently not possible [4, 5, 17, 23].

The objective of this work is to enable different local service discovery networks (such as UPnP networks) to discover services in other similar networks (that is, discover remote services which are not available locally). We lay the basis of such remote service discovery by proposing a suitable architecture. We use UPnP as a prototype for service discovery protocols. In our architecture each local UPnP network is enhanced by a function called service mirror builder. A service mirror builder presents local services as remote services to other UPnP networks, and also builds mirrors of remote services in its local network. The process of finding a remote service is done with the help of the distributed peer to peer search protocol Gnutella.

A service mirror builder is seen as an UPnP enabled device in the local UPnP network. It is worth emphasizing that UPnP is just an example; the service mirror builder can be generally defined as a service discovery enabled device with respect to any service discovery protocol.

1.5 Motivation

Between other things pervasive computing means spatial heterogeneity: some places offer all the services a user needs and others only have a few services to offer. A combination of remote and local services is sometimes needed to satisfy the user's needs (such as establishing a remote connection for a video game player to join a multiplayer video game, as we will see below). The following scenarios motivate our quest for remote service discovery.

Realizing the smart home With the support of new technologies as well as wider development of existing ones, pervasive computing is being realized day by day. One small example of pervasive computing environment is a connected home or smart home. A smart home is a house in which most of the appliances, devices and services are connected with each other and form a local network. In other words, a connected home environment is a dwelling incorporating a communications network that connects the key devices such as sensors and actuators, electrical appliances and services, which allows them to be remotely controlled, monitored or accessed [17]. Since services in a smart home must be able to be controlled, monitored, or accessed remotely, so to realize a smart home we need to have a mechanism to access its services remotely.

Accessing home located devices Most of us desire seamless storage, access and consumption of digital content from and to any compatible digital device in a home or smart home. Advanced mobile terminal products, with increased multimedia and broadband connectivity potential also increase the need for accessing various home located devices [5]. Ideally, users should be able to access their residential services from anywhere using any type of terminal [23].

Use cases for remote service discovery include lighting, residential climate control, home theatre, audio entertainment systems, domestic security, domestic health care systems, and so on.

Customer support and continuing close presence Support of home appliances remotely by vendors is another important need. Vendors need to connect to their devices for various purposes such as to update their software or perform routine checks.

Security and health care companies in particular need to be in contact with their customers and their products continuously. The information from sensors, actuators and cameras can be monitored by security and health care companies to take action in case of any threat. These sensors, actuators, etc. can also be controlled by the service providers to be more efficient and able to satisfy all the security or health care needs. Finally the vendors can advertise their new features and devices to their customers and offer customers upgrades to their devices.

Multiplayer video games Multiplayer Video Games (MVG) are video games in which more than one person can play at the same time. In these kind of games people can play with each other in a form of partnership, competition, rivalry, or a combination thereof.

There are different kinds of MVGs. Among them Massively Multiplayer Online games (MMO or MMOG) and Massively Multiplayer Games (MMG) are very popular. MMOGs are graphical 2- or 3-D video games played online, allowing players to interact not only with the gaming software but with other players' avatars (self created digital character of the players) as well [42]. Massively Multiplayer Games are traditionally supported by a client-server architecture where every player as a client communicates with a central server. However, such a centralized architecture lacks flexibility and can put communication and computation stress on the servers [8]. To overcome these problems inherent to centralized solutions, peer to peer overlay networks are emerging as a promising architecture for MMGs [8].

Running MVGs with the help of remote service discovery is perhaps the best use cases to motivate our research contribution. For example, Bob has a MVG console connected to his home UPNP network. He has a new MVG on his console and he wants to play it with someone else. Finding another player like John who has the same game and is willing to play with someone else at the same time but at his home (a remote network), is one of the scenarios which can be easily realized by our proposed architecture. Furthermore this scenario can be realized in a fully distributed manner, without any need for a centralized coordinator to discover players.

1.6 Outline

The remainder of this work is organized as follows.

The next chapter contains of 2 parts. The first part is an introduction to the Universal Plug and Play service discovery protocol (UPnP) and the second part is an introduction to the Gnutella distributed search protocol. These two protocols have been chosen as representatives of their protocol categories to assist us in explaining our approach to remote service discovery.

Chapter 3 presents research related to our own (also addressing the enhancement of UPnP with remote service discovery capabilities). In addition we introduce briefly the Service Oriented Architecture (SOA), the Jini service discovery protocol, and UPnP bridges, as these could be considered previous research in our domain as well.

Chapter 4 is all about our new architecture, featuring the UPnP service discovery protocol enhanced with a function (called service mirror builder) which can connect to other networks and discover remote services using a Gnutella client software.

We conclude in Chapter 5.

Chapter 2

The UPnP and Gnutella Protocols

In this chapter we summarize the two protocols on which we base our new architecture for remote service discovery. These protocols are Universal Plug and Play (UPnP), a service discovery protocol for local services, and the Gnutella distributed (peer to peer) search protocol.

2.1 Universal Plug and Play

The automatic detection by the operating system of new devices connected to a computer is called Plug and Play (PnP). The operating system can discover new devices and configure them without any physical device configuration or human intervention.

Plug and Play was the basis for Universal Plug and Play or UPnP [44]. The idea of Universal Plug and Play is the automatic discovery and configuration of any new devices that connect to a computer network. UPnP supports zero configuration networking or Zeroconf, meaning that UPnP creates an IP network without any need of manual configuration or configuration servers.

UPnP uses the Internet protocol suite: TCP/IP, HTTP, SOAP and XML. Special features of the protocol include the following [36, 44].

- *Media and device independence:* Any network media or device which supports

IP can be a basis for the establishment of UPnP.

- *User Interface (UI) control:* Devices can have a UI written by XML which is readable by a browser

- *Operating System and programming language independence:* This feature together with media and device independence make UPnP just like the Internet.

UPnP has three major components:

- **Device:** a device contains one or more services.

- **Service:** a service performs actions and shows its state via state variables. A service in UPnP consists of a state table, a control server and an event server.

- **Control Point:** a control point is a system that discovers and then controls services and devices.

The functioning of UPnP then involves six steps, as follows. These steps are also summarized in Figure 2.1.

Addressing Each device must have a Dynamic Host Configuration Protocol (DHCP) client. When the device connects to the network for the first time it must search for a DHCP server. If there is a DHCP server, then the device receives an IP address in this way. Otherwise the device must assign an IP address to itself. This address assignment is called Auto-IP. In the UPnP Device Architecture version 1.1 [45], Auto-IP references IETF RFC 3927 [37]. After assignment the device must check whether this address is not being used by anybody else. This is accomplished by the device broadcasting some probe message; if the device receives any other message with the sender IP address matching the address being tested, a conflict has happened. On the

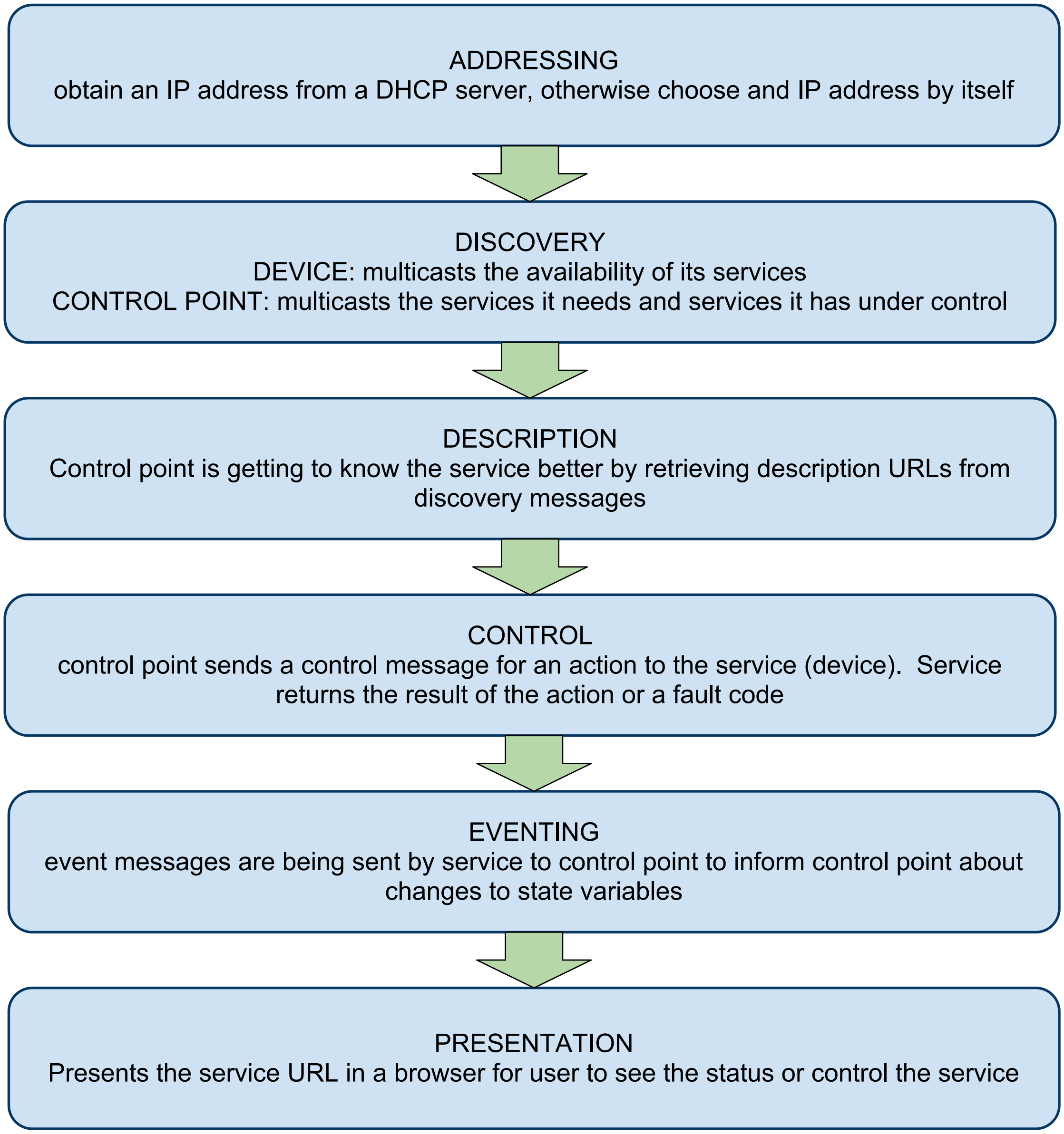
ADDRESSING
obtain an IP address from a DHCP server, otherwise choose and IP address by itself
DISCOVERY
DEVICE: multicasts the availability of its services
CONTROL POINT: multicasts the services it needs and services it has under control
DESCRIPTION
Control point is getting to know the service better by retrieving description URLs from discovery messages
CONTROL
control point sends a control message for an action to the service (device). Service returns the result of the action or a fault code
EVENTING
event messages are being sent by service to control point to inform control point about changes to state variables
PRESENTATION
Presents the service URL in a browser for user to see the status or control the service

Figure 2.1: UPnP service discovery and control

other hand, if a device receives a probe message with the same IP address as its own, it must send a response to the network, which will detect the conflict as explained before. A conflict implies that the address is already in use and then the device should change the address and check again. Even after the Auto-IP phase is complete, the device must periodically check for the presence of a DHCP server [36, 44]. Probing a new IP address, conflict detection, and address announcement are the three phases of Auto-IP as described in the IETF RFC 3927 [37].

Discovery Discovery is the process of discovering the capabilities of the devices on the network. It can take place in two ways.

When a new device gets an IP address and so is connected to the network, the device must multicast discovery messages, advertising its embedded devices and services. This process is called *discovery advertisement*. Any interested control point in the network can listen to these advertisements and then connect and control the originating devices or only some of their services.

The second way discovery can take place happens when a new control point is established in the network. Such a new control point multicasts a Simple Service Discovery Protocol (SSDP) discovery message [45], searching for available devices and services. All devices in the network must listen to these kinds of messages and respond to them whenever any of their services or embedded devices matches criteria in the SSDP messages. This process is called *discovery search* [36, 44].

Description Once discovery is complete and the control point knows about the existence of one device or service, it must also find out how to invoke that device or service. The respective control point retrieves the device description from the URL provided by the device in the discovery message. The UPnP description for a device is expressed in XML and includes vendor-specific information, manufacturer information, a list of any embedded devices or services, as well as URLs for control, eventing, and presentation [36, 44].

Control Now that the control point has a clear overview of the service and knows how to control it, it can send an action request. The control point sends a control message to the device according to the respective service control description. Control messages are expressed in XML. In response, the service will return action specific values or fault codes [36, 44].

Eventing Services keep control points informed by sending them event messages. Event messages contain the last update of changed state variables in the service. This process is called eventing [36, 44].

Presentation Some devices have URLs for presentation. Such an URL can be fetched and then presented in a browser by the control point. According to the device capabilities and URL presentation definition, a user can then see the status of the service and even control it [36, 44].

2.2 An Introduction to P2P File Sharing

A distributed network architecture may be called a peer to peer (P2P) network whenever the participants share a part of their own hardware resources (processing power, storage capacity, network link capacity, printers, etc.) with each other. These shared resources are necessary to provide the service and content offered by the network (e.g. file sharing or shared workspace for collaboration). Furthermore they are accessible by other peers directly, without passing through intermediate entities. The participants in such a network are thus resource (service and content) providers and at the same time resource (service and content) requesters (the "servent" concept) [40].

Peer to peer file sharing is a particular example of peer-to-peer network. In this case the peers are sharing some digital data or files. The peers are also called hosts, nodes or servents (having the functionality of both a server and a client). Each peer in a peer to peer file sharing network is implemented by a file sharing software client. This client uses some distributed search protocol to find other peers as well as the files that are being shared by them.

Different protocols for distributed search are being used by peer to peer file sharing programs. The two most prominent distributed search protocols are BitTorrent [12] and Gnutella [11].

2.3 Gnutella

Gnutella is a protocol for distributed search [11]. The participating servents (or nodes) can act as clients (meaning that they can issue queries and view search results), and also as servers (meaning that they accept queries from other servents, check their data sets to find any matches, and respond with applicable results).

Because of the distributed nature of Gnutella and its independency from any central servers, a Gnutella network is highly fault-tolerant. Indeed, a network can work continuously despite the fact that different servents go off-line and back on-line [11].

We describe in what follows the Gnutella protocol [11, 25, 35] and we also explain its functionality. The first time a servent wants to join a Gnutella network, its client software must *bootstrap* and thus find at least one other servent (node or peer) in the network. A bootstrap is thus the process of joining the network by discovering other servents [21]. It can happen automatically or manually:

1. **Out of band:** The user can inquire about another Gnutella servent using some out of band method such as Internet Relay Chat (IRC) or Web pages. Then the user communicates the received hosts to the Gnutella client software which tries to connect to them until one of them works.

2. **Gnutella Web caches**: Caches that include a pre-existing list of addresses of possibly working hosts may be shipped with the Gnutella client software or made available over the Web.

Once the servent finds at least one active peer in the Gnutella network, it can find more peers with watching Ping and Pong messages. A *ping* is used to actively discover hosts on the network. A servent that receives a ping message is expected to respond to it with a *pong* message. A pong is the response to a ping and includes information about the sending servent.

Two nodes in Gnutella network are neighbour if they are directly connected. A node that is connected to a Gnutella network informs periodically its neighbours through ping messages. These messages are not only replied to by pong messages but they are also propagated to the other interconnected servents. Therefore when a servent receives a ping message, it sends it to the nodes to which it is connected (typically servents are connected directly to 3 other nodes). Each servent can create an updated list of active servents in the Gnutella network by listening to the ping messages and the corresponding pong messages.

When a client wants to search for a file (or as we will see in Chapter 4 for a service), it sends a query to all its directly connected neighbour servents (except the one which delivered this query message). Then these neighbour servents forward the query to their neighbours and so on. This process repeats throughout the network.

A *query message* is the primary mechanism for searching the distributed network. If a servent receives a query and finds a match in its directory, it will respond to it with a *query-hit message*. A query-hit is the response to a query and contains enough information for the retrieval of the data matching the corresponding query.

To avoid flooding the network the query messages contain a TTL (Time To Live) field. It is possible that one query reaches a servent more than one time. To avoid serving a query more than once, each query is identified by a unique identification called *muid*. Before processing a query a servent checks the query's muid against a table of previous muids. If they have encountered the query muid before, then they simply drop the query message.

The query-hit can go back along the reverse path of the query to reach the servent which requested it, or it can be sent directly to the requester.

Chapter 3

Related Work

We review in this chapter previous attempts at remote service discovery. We believe that our architecture (Chapter 4) remedies the shortcomings of these solutions. We will offer a critical comparison in our conclusions (Chapter 5).

3.1 Remote Access to UPnP Devices Using the Atom Publishing Protocol

One architecture for remote service discovery in UPnP [4, 5] uses the Atom Publishing Protocol [20] and the Atom Syndication Format [33].

The network topology of this architecture consists of at least two network segments: one is called home network and the other external or remote network. These two networks are connected to each other through the Internet. The architecture assumes that there is an IP tunnelling mechanism such as a Virtual Private Network (VPN) between the two network segments.

The architecture introduces a new element called *UPnP Device Aggregator* which is acting as a proxy for the existing standard UPnP devices. *Enhanced UPnP Devices* or *Control Points* are then UPnP devices or control points which are compatible with this remote service discovery architecture.

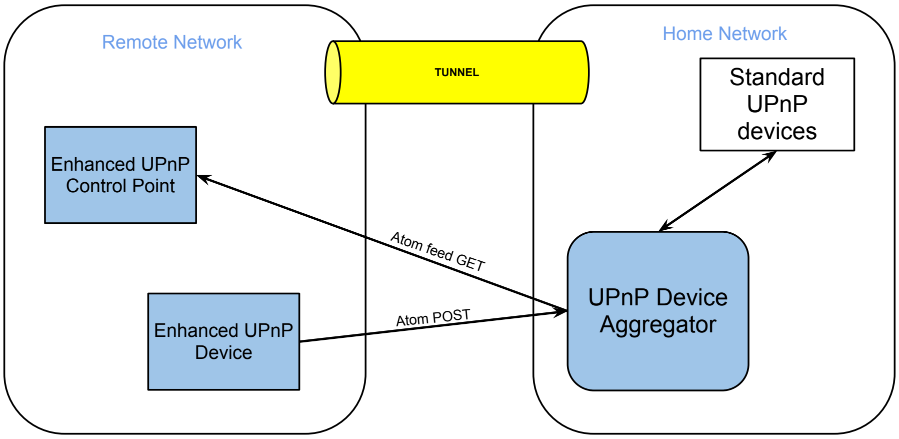

Figure 3.1: Atom-based architecture

The UPnP Device Aggregator aggregates information about the services and devices in the local network and presents them to the enhanced UPnP control points in the remote network. In fact this aggregated information is assembled as an Atom feed which can then be retrieved by an Enhanced UPnP control point through the Atom feed GET command.

Additionally, a UPnP Device Aggregator can receive information from remote Enhanced UPnP Devices and present them to the local control points. This information can be received by the UPnP Device Aggregator through the HTTP POST method. Figure 3.1 shows this architecture.

3.2 Remote Service Discovery and Control for Ubiquitous Environments

An architecture for remote service discovery and control based on presence service [13] was also proposed [23]. The presence service is developed by the Internet Engineering

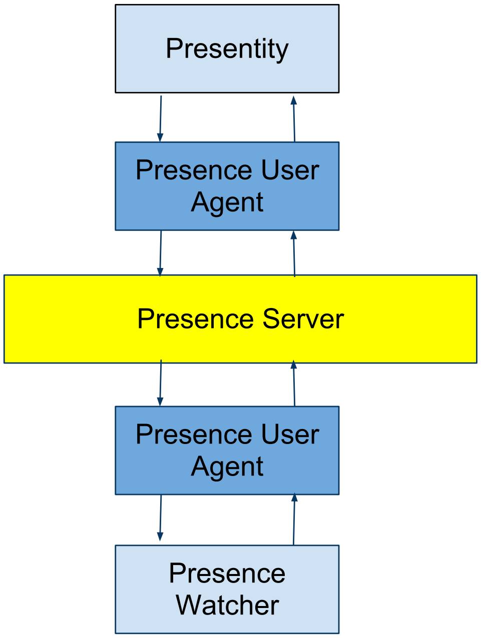

Figure 3.2: Presence service design (as used for remote service discovery and control in ubiquitous environments)

Task Force [13]. The architecture of this service includes four major elements: presence user agent, presence service, presence watcher and presence entity (presentity) [23].

The presentity can be anything that can have a presence state (be present or absent). Examples include a human, a group of humans, an office, etc.. Presence information is a status indicator that specifies availability and willingness to communicate. Presence information or presence states are sent to a presence service, which is a network service that records and distributes presence information. The presence information is mainly used in instant messaging and voice over IP [14, 23, 46]. For example, users of an instant messaging service (such as Yahoo! Messenger or Google Talk) are presentities, and their presence information is their status (online, offline,

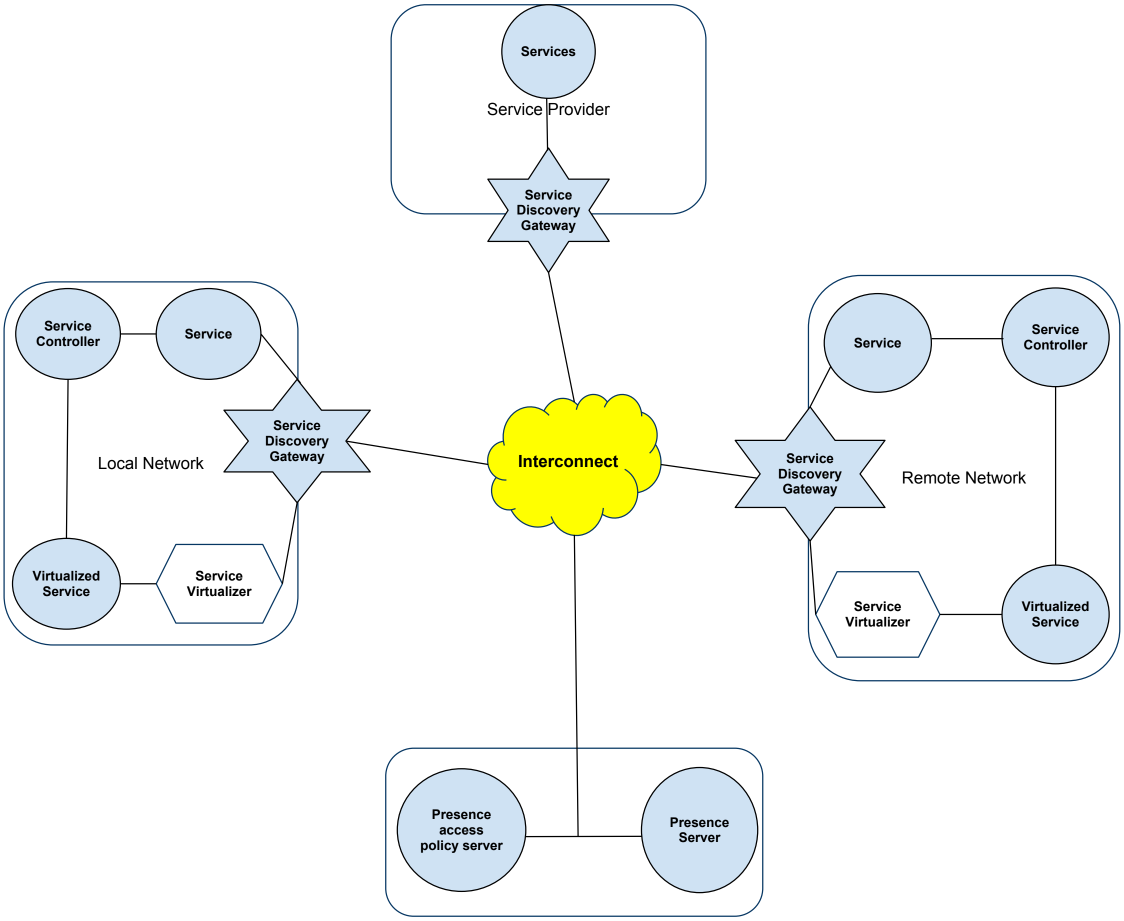

Figure 3.3: Presence service architecture

busy, etc.). The presence user agents are any devices or entities that can communicate presence information to a presence server or service (thus "subscribing" to that service) and can process notify requests from the presence server. A presence watcher is a presentity that requests presence information of a presentity from a presence service. Figure 3.2 shows the presence service design.

In the remote service discovery architecture based on presence service [23] there are two new functions called *service discovery gateway* and *service virtualizer*. Each service is seen as a presentity. The service discovery gateways register local services as presentities in a presence server. They can also retrieve other presentities from the presence server and present them to the service virtualizer. The service virtualizer uses this presence information to virtualize a local service in the local network. That is, a service virtualizer presents a remote service as a local one. Figure 3.3 shows this architecture.

3.3 Content Sharing and Transparent UPnP Interaction Between UPnP Gateways

Dynamic Overlay Topology Optimizing Content Search (DOTOCS) [27] enables flexible content searches among UPnP gateways. DOTOCS aims to establish an optimized peer to peer overlay network among UPnP gateways. The emphasis is on an optimized unstructured peer to peer protocol which puts nodes with similar contents next to each other.

DOTOCS uses a communication protocol between UPnP local networks described elsewhere (transparent interaction solution [34]): The communication between two connected UPnP local networks across the Internet is accomplished using the Web service technology. A local gateway encapsulates Simple Service Discovery Protocol (SSDP) messages into Simple Object Access Protocol (SOAP) messages and transmit them to another gateway over the global network. A Web service at the destination UPnP gateway extracts the SSDP message and replaces the original IP address (which is not valid in this local network) with the IP address of the gateway itself. The gateway then multicasts this discovery search message in the local UPnP network. If any device responds to that message (meaning that the device has the service demanded by the SSDP message), then the gateway encapsulates that message into another SOAP message and sends it back to the first network. This way one local

UPnP network can discover remote services from a different UPnP network.

3.4 UPnP bridges

UPnP bridges are applications that enable the presentation of non-UPnP devices (that is, devices that do not support UPnP) to a UPnP network. Example of bridge applications include the *IEEE1394-UPnP bridge* [28], the *Jini-UPnP bridge* [39], or the *HNCP-UPnP bridge* [26]. These applications try to create a virtual presence for devices and services which are not able to support UPnP.

While UPnP bridges are not directly related to our work, it is worth mentioning that this kind of applications could be a part of the local UPnP networks used in our investigation. These applications are mostly used to connect to the local network those physically available devices that are not able to connect directly via UPnP (such as IEEE1394 devices [28]). These bridges can also be used to add all services from another network (with a different service discovery protocol) to the local network (examples include the Jini-UPnP bridge [39]). Therefore UPnP bridges are relevant in that they can be used locally just like any other UPnP device but also because they offer an (albeit severely limited) solution to our problem (remote service discovery). Given the obviously limited nature of this remote service discovery solution we will not consider UPnP bridges in our discussion any further.

3.5 Service Oriented Architecture

In this section we discuss the Jini service discovery protocol and its architecture which is a particular instance of Service Oriented Architecture (SOA) [30]. Although this topic is not directly related to our work, our investigation as well as some of the related investigations outlined above are intimately related to SOA. We will also use

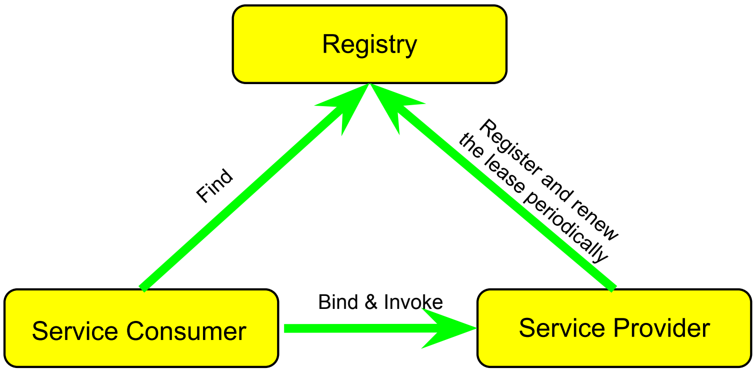

Figure 3.4: Service oriented architecture (the find-bind-execute paradigm)

the context of SOA in our concluding remarks.

Sun defined Service Oriented Architecture(SOA) in the late 1990s to describe Jini, which is an environment for dynamic discovery and use of services over a network [30]. Jini is in effect a distributed service-oriented architecture [9].

An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how) [30]. The service oriented architecture has three major components: registry, service consumer, and service provider. Service providers first register their services in a service registry; this process is called *register*. Service consumers can then find the needed services from the registry through a process called *find*. A registry then sends information to the service consumer so that the consumer is able to invoke the respective service. The service consumer then binds to the service provider and starts invoking the requested service. This process is called *bind and invoke*. An important mechanism in service oriented architecture is checking for the availability of services. For this purpose the registry allocates a lease to each service.

Service providers have to renew their leases to continuously affirm the availability of their services. Whenever the registry does not receive a renewal of the service lease, it will remove the service from the list of available services [30]. Figure 3.4 shows these steps in the service oriented architecture.

Chapter 4

A New, Distributed Architecture for Remote Service Discovery

One of the challenges that service discovery protocols are facing is remote service discovery. This is a very important need especially in the context of computing anytime, anywhere.

Recall that remote services are services not present in the current physical location of the controller (control point) but should be available to the controller upon request. In other words, service discovery protocols should be able to step out of their local domain in order to find services and in turn serve the users' needs. Additionally, a control point may reside in a pervasive computing environment with heterogeneous protocols and networks; even if some otherwise available services in the local domain could not be accessed because of heterogeneity in protocols, networks, ontologies, and so on, the controller may still be able to access services within its capabilities but far from its physical location. In other words, sometimes service discovery protocols could not see all the available services in their domain, but if they could just bridge to neighbour networks (with the same protocols and ontologies) they could accomplish their tasks.

Remote service discovery is especially important in pervasive computing, which consists of devices that surround people to help and assist them in their everyday lives. We already discussed in Section 1.5 some use cases which show the need for remote service discovery in a pervasive computing environment.

We are proposing in this chapter a new architecture that accomplishes remote service discovery in a fully distributed manner, that is, without the need of any centralized, coordinating entity. Our architecture allows the discovery of services in local and remote domains, and offers a solution for automatic discovery and control of remote services.

4.1 An Introduction to the Architecture

Our architecture features UPnP or other similar (local) service discovery protocol. One important example of pervasive computing is the smart home or connected home. In a smart home environment there are many appliances and many service discovery protocols being used in consumer space [23] including UPnP, Bluetooth service discovery and Apple Bonjour. Because of the popularity of UPnP we use it as a prototype in our architecture, but we in fact try not to depend on any particular service discovery protocol.

Our architecture is outlined in Figure 4.1. There are 5 local networks in the figure, labelled from 1 to 5. Each of these local networks offers local services, devices, and control points. These devices, services, and control points are connected with each other locally through UPnP. In each local network there is one special function (which can also be seen as a UPnP enabled device) called *service mirror builder*. This special function will help with remote service discovery. We will discuss the UPnP network and the service mirror builder in the next section.

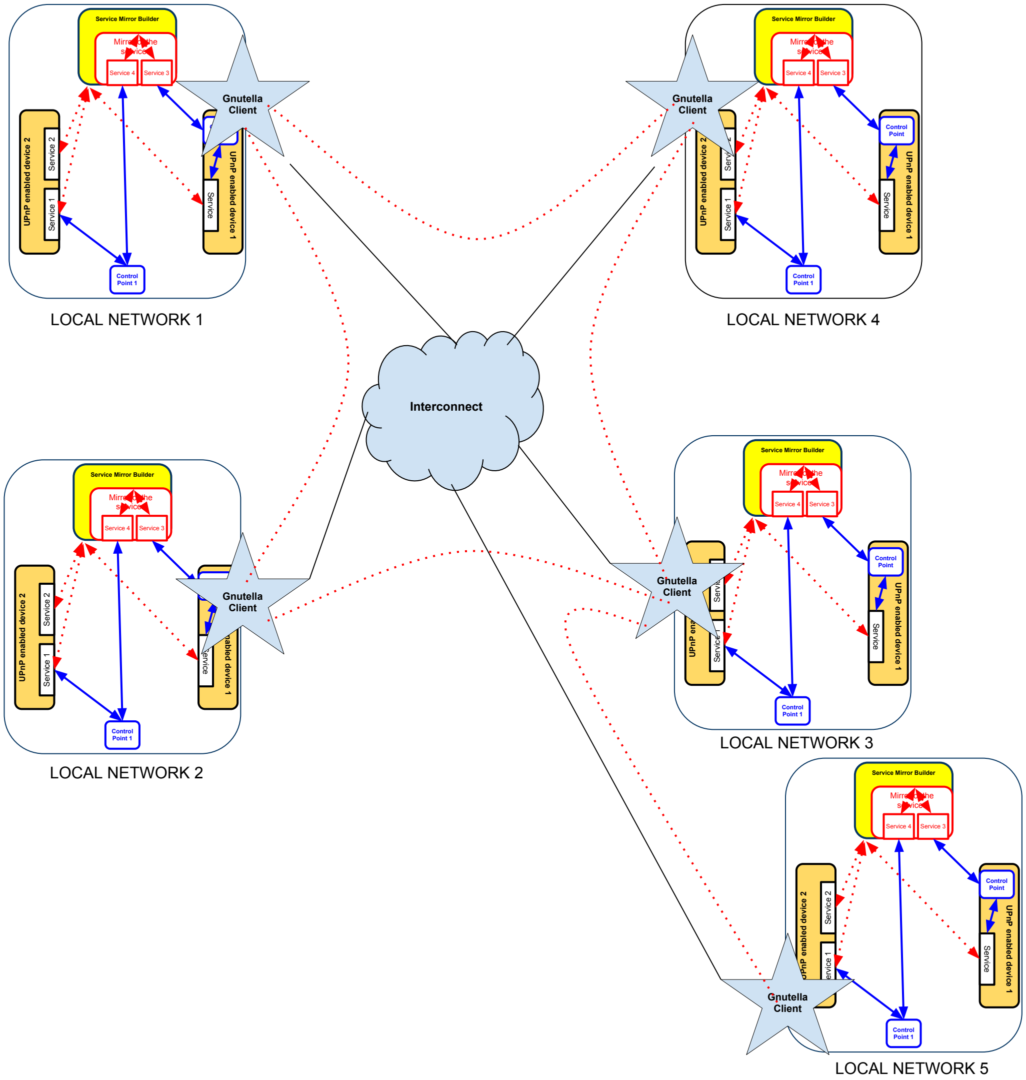

Figure 4.1: A distributed architecture for remote service discovery (doted lines show the Gnutella network overlay; each local network has the structure shown in Figure 4.2)

In addition, each local network runs a Gnutella client software, shown as a 5-point star in Figure 4.1. These Gnutella clients are specialized clients that share local services to the outside world and find services requested by their service mirror builder. The local networks establish a Gnutella network between them. We use dotted lines in the figure to show the overlay of the Gnutella network. In our example client 1 is connected to clients 2 and 4; client 2 is connected to clients 1 and 3; client 3 is connected to clients 2, 4 and 5; client 4 is connected to clients 1 and 3; and client 5 is connected only to client 3.

4.2 The Local Network

Refer to Figure 4.2 for a closer look at one of the local networks (and assume for demonstration purposes that this is the local network 1). A local network contains a number of (local) devices, services, and control points. Every local network contains a unique service mirror builder. The network is an IP based network with all of these devices connected through UPnP (the UPnP protocol with its six steps is described in Chapter 2).

Since UPnP uses the TCP/IP and Web models, it has the following features:

- *Media and device independency:* UPnP can run on any media including phone lines, power lines, Ethernet, RF, and IEEE 1394 seamlessly and in a peer to peer manner [31].

- *Operating System and programming language independency:* Like the Internet, UPnP is not dependent on any particular operating system installed on devices or any particular programming language used to write services.

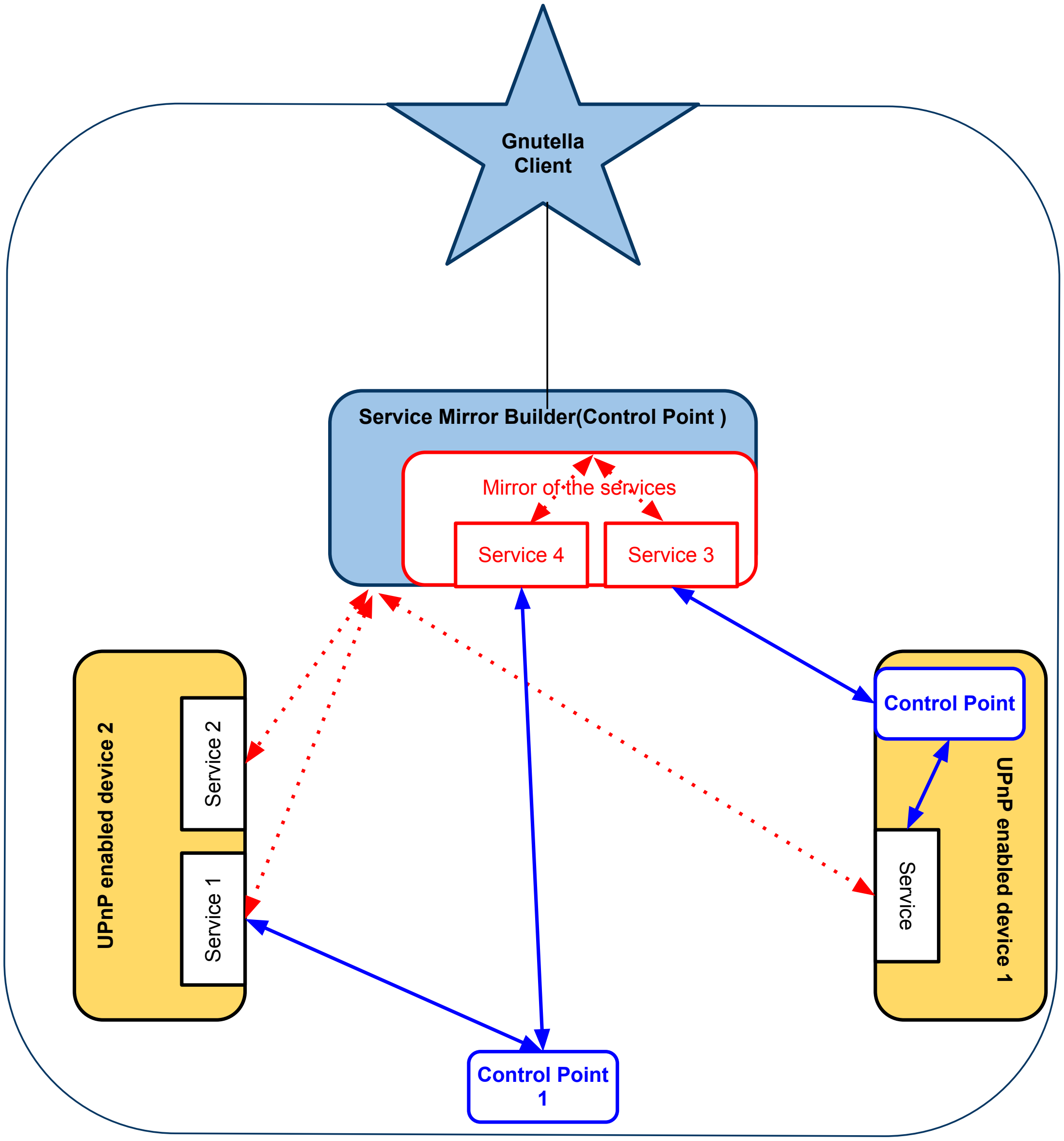

Figure 4.2: Local network structure

- *User Interface control:* Each device and service can have a User Interface described in XML

Addressing Addressing is accomplished using the standard UPnP protocol. All devices and services first search for a DHCP server; if no DHCP response is received, then they must wait and again multicast a request for IP address from the DHCP server. As a failover mechanism, device must assign an IP address to itself whenever no DHCP server can be reached. This address is multicast to the network, and the process is repeated whenever another device is found to have the same self-assigned IP address. Multicasting is accomplished using UDP, while TCP is used for all the remaining steps.

The sample local network in Figure 4.2 features four components: UPnP-enabled device 1 (just device 1 or even D1 for short), UPnP-enabled device 2 (device 2 or D2), one control point (control point 1) and one service mirror builder (SMB for short). The service mirror builder typically resides on the smart environment gateway (such as a connected home gateway). All of these components have unique IP addresses.

Discovery-advertisement Now all the devices, control points, and the service mirror builder are connected to the network and have unique IP addresses. The subsequent discovery dialogue is depicted in Figures 4.3, 4.4, 4.5, and 4.6. Initially device 1 has not introduced its service to other control points except the service mirror builder, and its control point has discovered a mirror of a remote service (service 3). Device 2 is a UPnP device with 2 embedded services (service 1 and service 2) which are similarly not known to the others. Now device 2 must inform all the available control points in the network about its services; it does so by multicasting a message and thus advertising services 1 and 2.

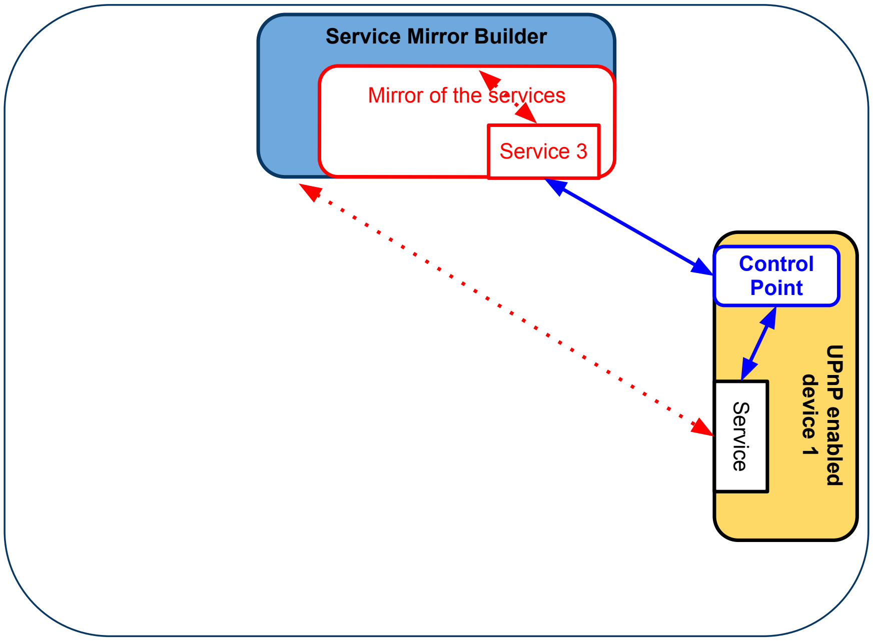

Figure 4.3: SMB and a device connected in the UPnP network

These advertisement messages have a timestamp to indicate for how long this advertisement is valid. Before this time expires, the device must renew the advertisement; otherwise all the control points will infer that the respective service is no longer available on the network.

In local network 1 used in our example, the multicast message will be received by the service mirror builder and also by device 1. The control point in device 1 is not interested in (or not capable to control) either service 1 or service 2 and so it ignores this message. However, the service mirror builder must be aware of all the available services in the local network, so it cannot ignore any multicast message. The service mirror builder uses this information for remote service discovery, which will be discussed later. In local network 1 the service mirror builder is (obviously) interested in service 1 and service 2. It then sends a message to device 1 to retrieve

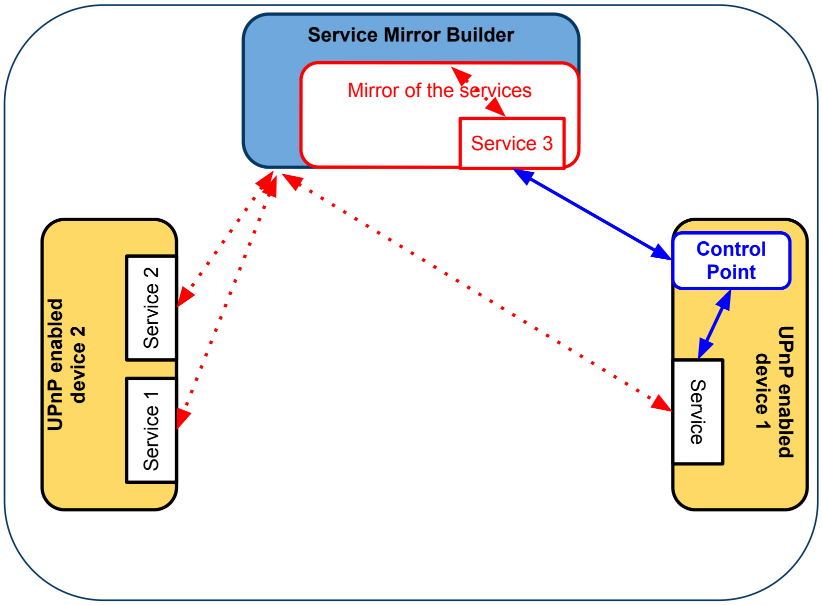

Figure 4.4: SMB requests descriptions of Services 1 and 2

the description of the two services as discussed in the next step below (Description). Figure 4.4 shows this step.

When the service mirror builder receives a request for a service (which is not locally available), it tries to request it from a remote network. Once such a service is found, a mirror of that service is made available in the local network. In Figure 4.2 the mirrors of the remote services are shown in the service mirror builder box.

Discovery-Search Another possibility of discovery is when a new control point is added to the network. In Figure 4.5 device 1, device 2, and the service mirror builder have all discovered each other. Control point 1 is then added to the network, has its own IP address, but has not discovered any services to control yet. In such a case the newly added control point multicasts a Simple Service Discovery Protocol (SSDP) discovery message [45] sent using HTTPMU [19, 36], thus searching for available

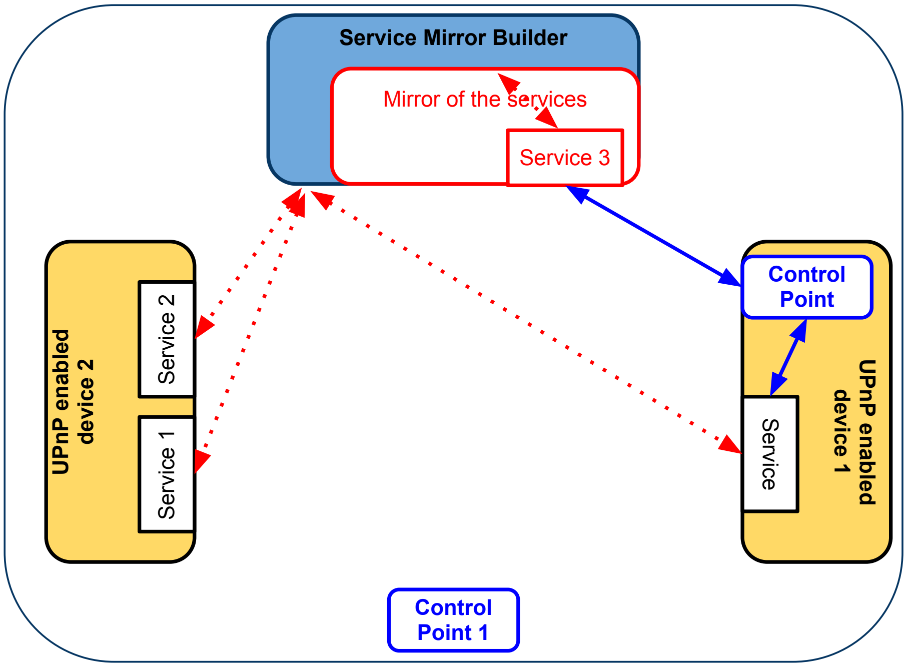

Figure 4.5: SMB, D1, D2 have discovered each others' services (control point 1 is now added to the network)

services and devices in the network. The search message (which is encapsulated in the SSDP request) contains vendor-specific information which includes device or service types and identifiers.

All the devices in the network must listen to these SSDP messages and respond whenever any of their services match the criteria specified by the SSDP message. The response message is a unicast message sent via UDP with SSDP headers and contains the same information as the advertisement message. The service mirror builder listens to all these messages and for each such a message it checks whether the requested service is in the list of available local services. If it is, then the service mirror builder drops the message; otherwise, it proceeds to discovering the respective service remotely.

In Figure4.5, service 1 in device 2 is matched with the request of control point 1.

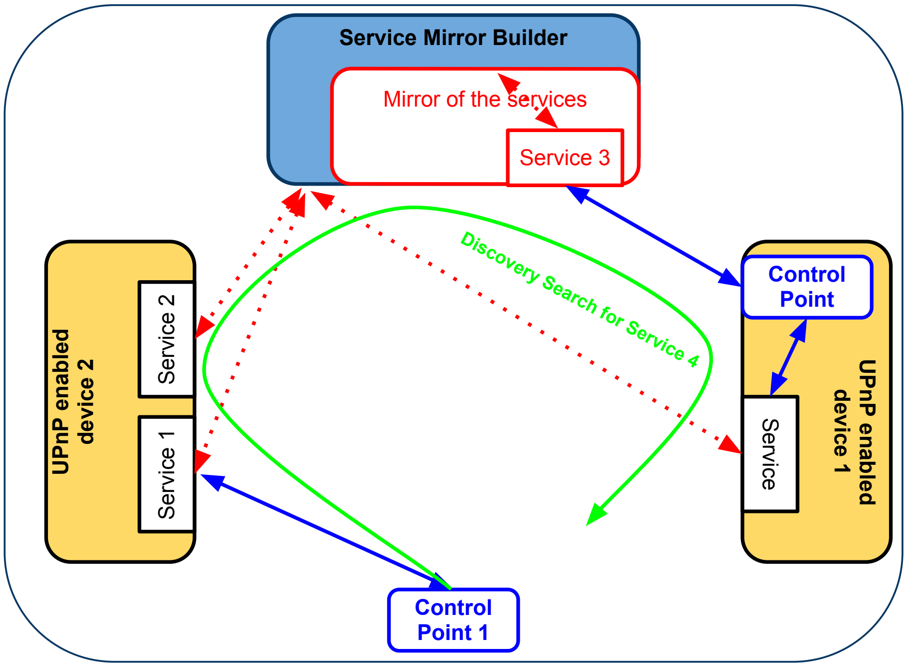

Figure 4.6: Searching for a service which is not available locally (control point 1 multicasts a discovery search message for service 4 which is not available locally)

Therefore device 2 unicasts a response message to control point 1. Now that control point 1 has discovered one of its needed services, it will ask for a description. Once the description is received, control point 1 can control service 1 in device 2. Assume now that control point 1 multicasts a discovery search message requesting a service which is not locally available (say, service 4). Figure4.6 shows this step. The service mirror builder will recognize that this service is not locally available, and so it sends a query for that service to the local Gnutella client. The Gnutella client will then propagate that query to the Gnutella network. We will show in detail how the remote discovery is accomplished in the next section (Section 4.3).

Description After the discovery step (which makes the control points aware of the available services), the control points must know how to use these available services.

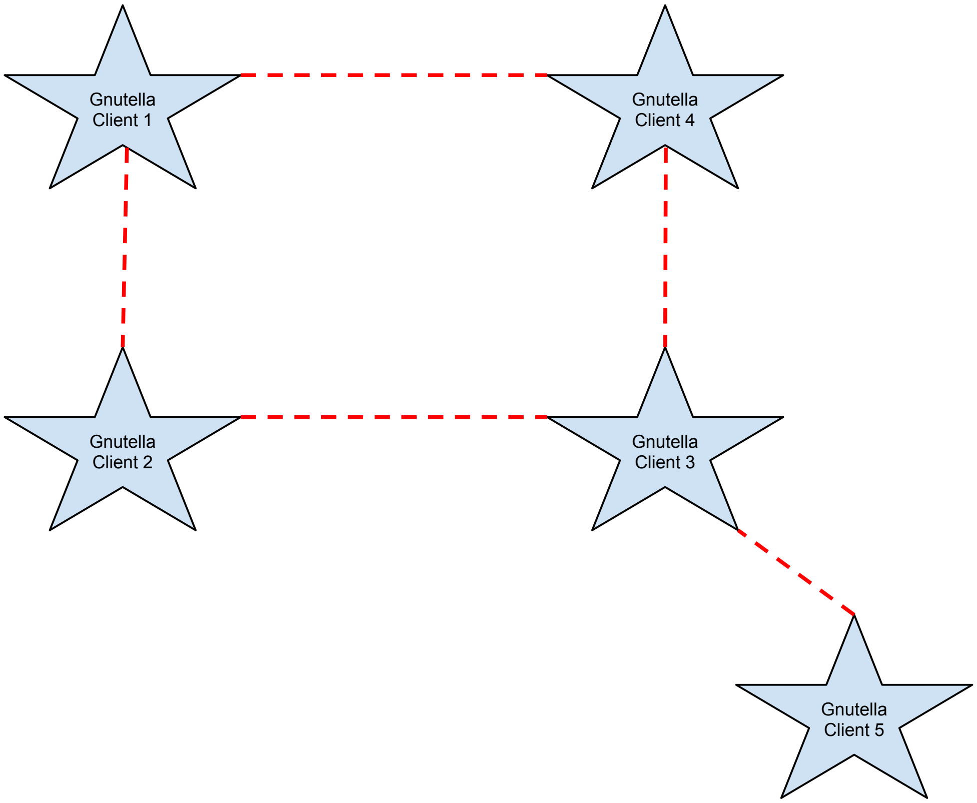

Figure 4.7: The overlay Gnutella network (five connected servents or local networks)

The process that accomplishes this is called Description. Advertising messages circulated during discovery contain URLs from which the control points can retrieve the description of the respective devices. The control points then issue HTTP GET requests to the devices to obtain the descriptions. In our sample network from Figure 4.2 control point 1 will request a description for service 1 in device 2.

The UPnP description for a device or a service is an XML document which contains vendor specific information, definition of all embedded devices, and an URL for the presentation of the device [36].

Control, Eventing and Presentation Once a control point has the device or service description it can invoke actions on that service and get result values in return. Invoking an action in UPnP is a particular instance of Remote Procedure Call [36, 41]. Control points send control messages (formatted in XML) which are encapsulated in a specific format using SOAP and then transmitted using HTTP over TCP/IP [36, 44].

The major focus of this research contribution is service discovery so we will not elaborate further on service control, eventing and presentation.

4.3 The Gnutella Protocol and Remote Service Discovery

Two major characteristics of pervasive computing are distributedness and mobility. In such an environment we want to connect nodes in a distributed manner and without any dependency to a central server (such as the presence server from Section 3.2). We therefore chose in our architecture Gnutella as the connecting protocol, since Gnutella is a strongly decentralized peer to peer system [25].

Servents in Gnutella can share any type of resources [25]. In our design servents are sharing their local services with remote servents. Figure 4.1 on page 34 shows this architecture and Figure 4.7 shows the overlaying Gnutella network. We now explain the remote service discovery process in our architecture.

The first step toward remote service discovery is the establishment of a Gnutella network. This is accomplished as follows.

1. In Figures 4.1 and 4.7 we have 5 local networks. Each local network has a Gnutella client which is specialized in sharing the services aggregated by the service mirror builder. We assume that local networks 2, 3, 4, and 5 are already connected to the Gnutella network. Local network 1 then can use one of the

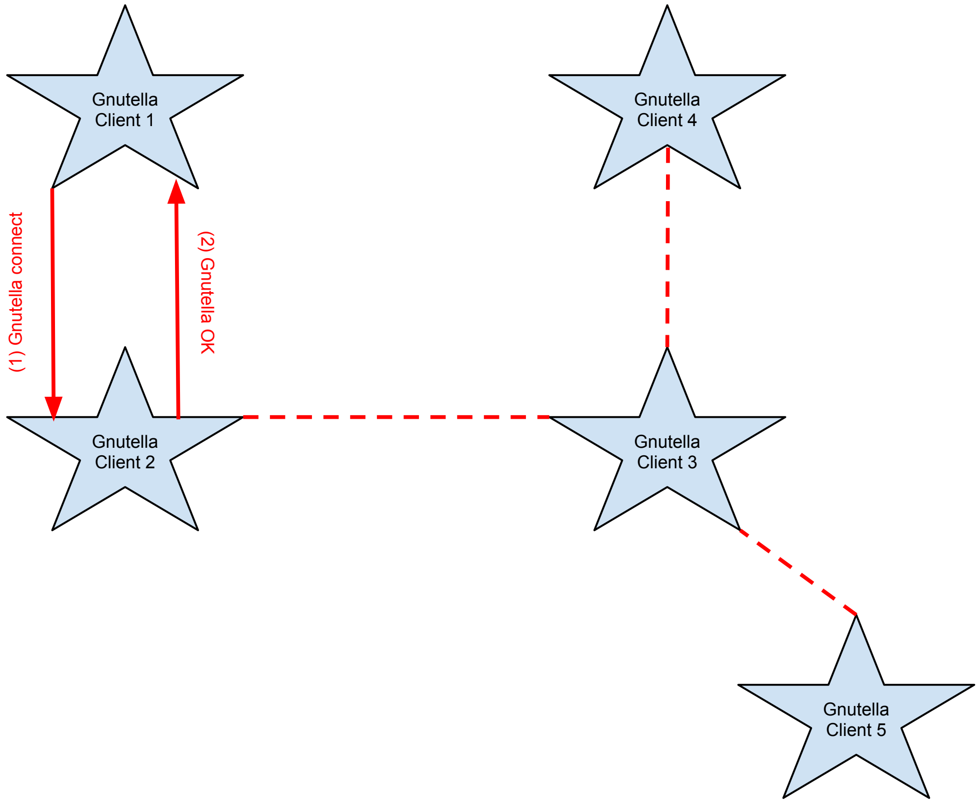

Figure 4.8: Local network 1 joins the Gnutella network

techniques discussed in Chapter 2 to join the Gnutella network. Suppose that the address of network 2 is thus given.

2. Local network 1 then tries to connect to local network 2. Local network 1 sends a "Gnutella connect" to local network 2 in order to join the Gnutella network. Local network 2 responds with a "Gnutella OK" to the local network 1. Local network 1 is now part of the Gnutella network. Figure 4.8 shows this step.

3. Gnutella servents periodically Ping their neighbours with their information. In figure 4.9 local network 4 Pings its information to its connected neighbour, local

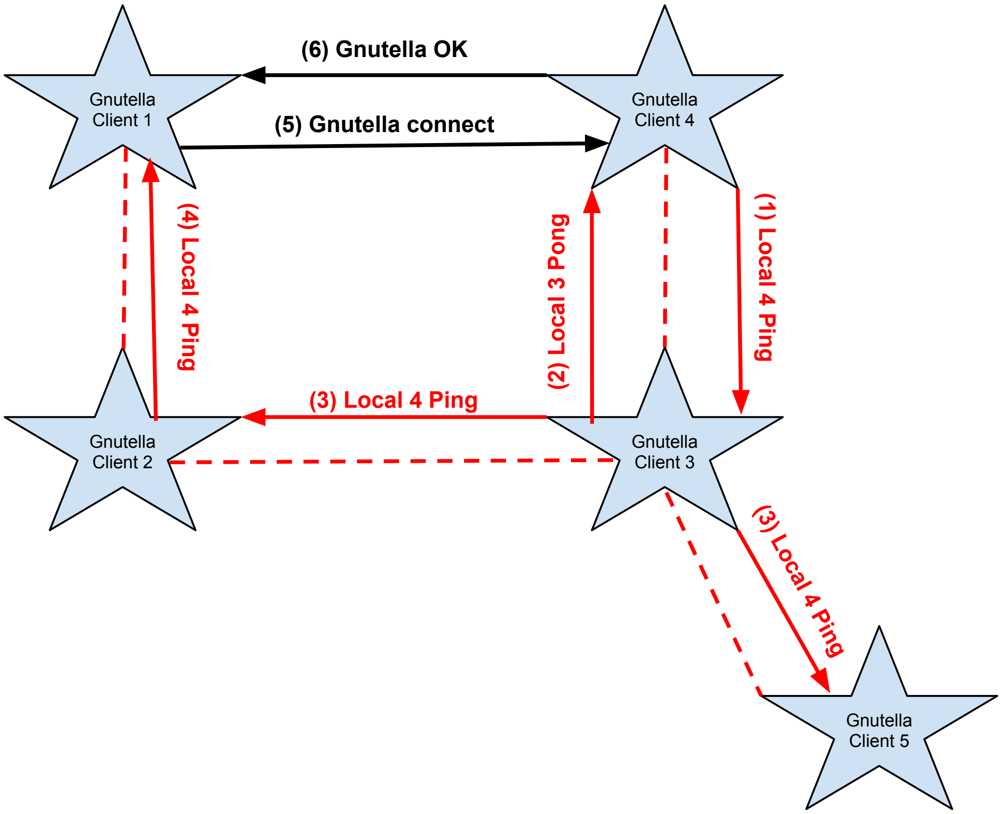

Figure 4.9: Local network 1 finds Local network 4 and connects to it via Ping messages

network 3. Local network 3 replies to local network 4 with a Pong message and propagates this Ping to its direct connected neighbours local network 5 and local network 2. Local network 2 and 5 do the same. Finally local network 1 receives Ping message from local network 4. Local network 1 uses the address of local network 4 from the Ping message and through ”Gnutella connect” connects to the local network 4. Figure 4.9 shows this step.

4. Now we have a Gnutella network with 5 nodes which are connected to each other like in Figure 4.7.

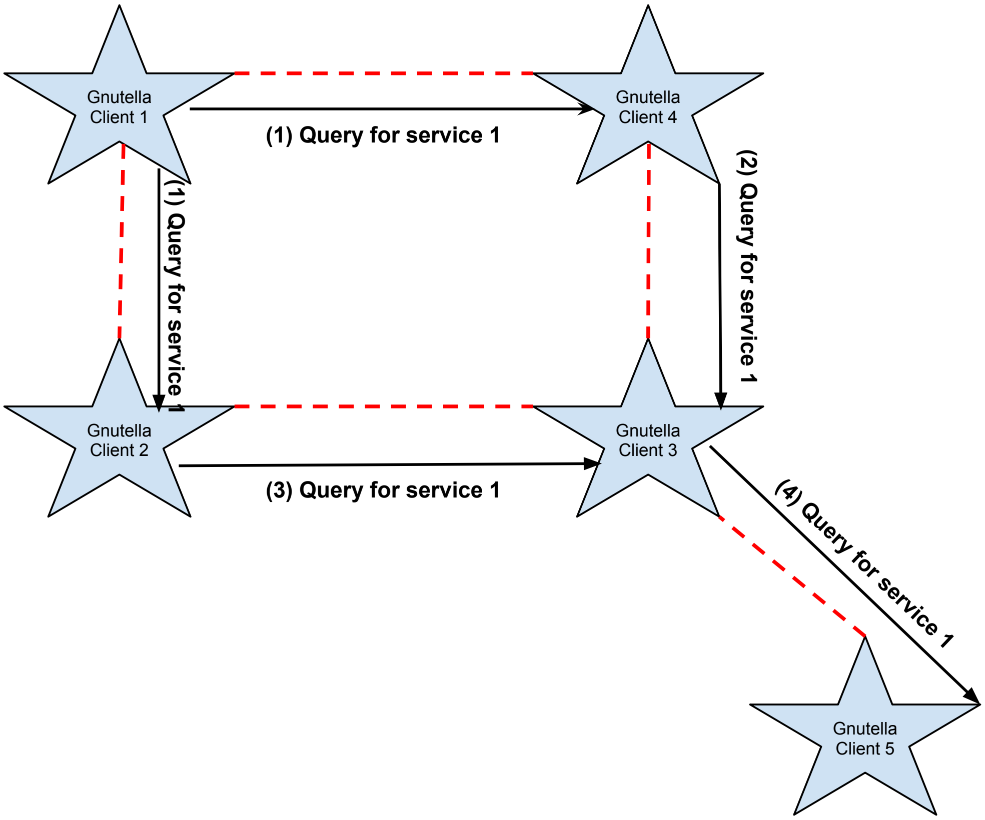

Figure 4.10: Finding a remote service (a query message for a service is sent from Gnutella client 1 to the Gnutella network)

Remote service discovery in the new architecture Now that both the local networks and the Gnutella network are established, remote service discovery can begin. Such an event happens whenever a control point requests a service but this service is not locally available. The service mirror builder then activates and tries to remotely discover it.

Each service mirror builder has a cached description of all of the available local services. When a control point requests a service, the service mirror builder checks in its local service directory and if there is such a service just ignores the query (since the control point can locally discover and control that service). However, if that service is not in the local directory, the service mirror builder proceeds to discover it remotely, as follows.

1. The service mirror builder sends a request for the service to the Gnutella client. The Gnutella client issues a query message asking for the requested service and sends this query to its directly connected servents. In Figure 4.10 local network 1 performs such a query. Gnutella client 1 then sends the appropriate query message to its directly connected neighbours, namely local networks 2 and 4.

   The Gnutella clients in local networks 2 and 4 first send the request to their service mirror builders to see whether the requested service is locally available. Suppose that both answers are negative; then Gnutella clients 2 and 4 send the query to their direct connected neighbours except to local network 1 (which originated the query). The query thus goes in local network 3. Gnutella client 3 receives this query message two times but discards the second instance after a muid check. This continues until the query reaches a network that can provide the requested service (local network 5 in our example).

2. Gnutella client 5 checks the query against its service mirror builder and receives a positive answer. Therefore Gnutella client 5 sends a query-hit to the requester (Gnutella client 1). This query-hit can be sent along the reverse path or directly to the Gnutella client 1. Figure 4.11 shows this action.

3. Now that the service has been discovered, node 5 can send a service description and other information back to the node 1. This information will be delivered to the service mirror builder of that node (1). This service mirror builder then creates a mirror of the service in the local network 1 (in our example service 4 which will be controlled by control point 1). From the point of view of the

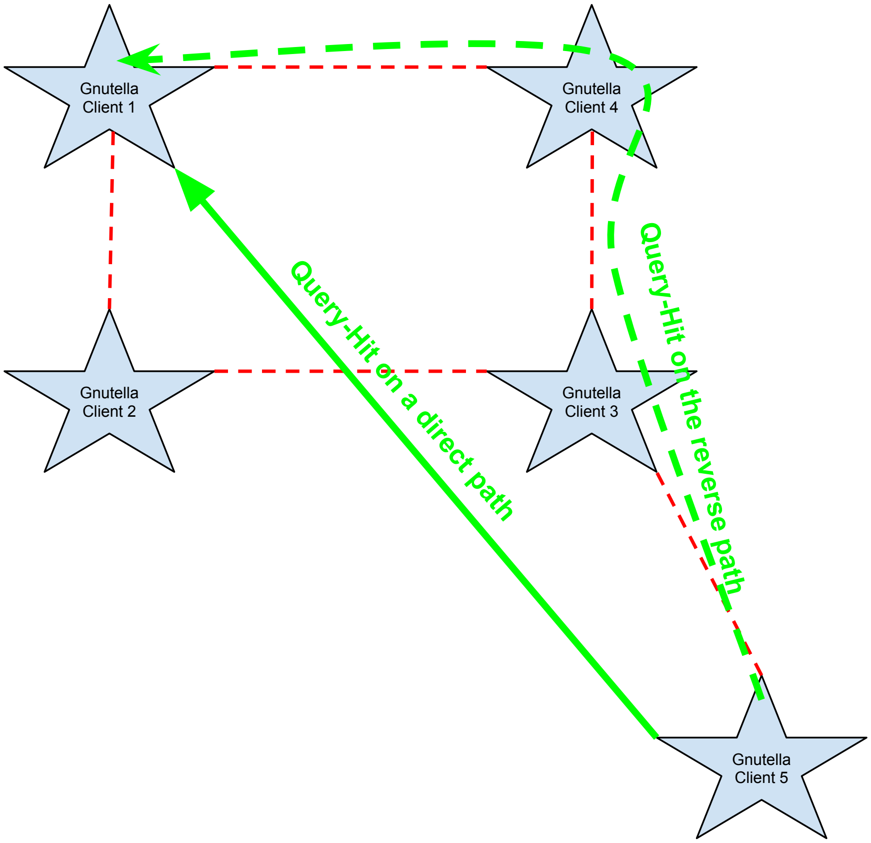

Figure 4.11: Query-hit response (a query-hit can reach Gnutella client 1 on the reverse path or directly)

control points in local network 1 the service looks just likes a local one and can be controlled in the usual way.

4.4 Gnutella and UPnP Messages in the New Architecture

This section shows the possibility of using the Gnutella distributed search protocol to search for services in remote networks (remote services). We do this by discussing the Gnutella and UPnP message structure and the modifications that are needed in our architecture.

In our architecture all local network components communicate and work with each other under UPnP protocol standards. All six steps in UPnP (addressing, discovery, description, control, eventing and presentation) are being done as per the UPnP protocol.

As far as the remote connections are concerned, all servents are working under the Gnutella standards and specification. All Gnutella connect, Gnutella OK, ping and pong messages are exactly according to the available Gnutella protocol. The only differences happen in the Gnutella query and Gnutella query-hit messages (since the original query and query-hit messages are used for requesting for and responding with shared files). We are then explaining the structure of these messages in more detail along with some recommendation for changing them to work in the new architecture for the purpose of service discovery instead of file sharing.

All of the Gnutella protocol messages, including query and query-hit, include a header with the following byte structure [29]:

Bytes	Description
0-15	Message ID/GUID (Globally Unique ID)
16	Payload Type
17	TTL (Time To Live)
18	Hops
19-22	Payload Length

This message header structure will remain unchanged in our architecture. The payload type in the header indicates the type of the message. The following are possible values [29]:

0x00 = Ping	0x01 = Pong	0x02 = Bye
0x40 = Push	0x80 = Query	0x81 = Query-Hit

Other Gnutella message types can be used as long as all the participating servents support them [29].

Payload length shows the size of the payload. The whole Gnutella message (according to the available specification) should not be more than 4 kB. Immediately following the message header is a payload which can be one of the following messages: ping, pong, query, query-hit and push [29].

The query message Since query messages are broadcast to many nodes, servents normally send query messages that are smaller than 256 bytes; however, query messages can be as large as 4 kB. A query message has the following structure [29]:

Bytes	Description
0-1	Minimum speed
2	Search criteria
Rest	Optional extension block

The rest field of a query message is used for the original query which in our case is a query for a remote service. The allowed extension types in the rest field can be specified using the Gnutella Generic Extension Protocol (GGEP), Hash/URN Gnutella Extensions (HUGE), and XML [29]. The Gnutella Generic Extension Protocol (GGEP) allows arbitrary extensions in a Gnutella message; a GGEP block is a framework for other extensions [29].

In a UPnP network service discovery is accomplished using Simple Service Discovery Protocol (SSDP) [45]. All SSDP messages are sent using the HTTP protocol version 1.1 [18]. The HTTP 1.1 and Gnutella protocols are both application layer protocols. The fundamental data in a SSDP discovery search or discovery advertisement message (in a UPnP network) contains a few essential specifics about the device or one of its services (e.g., its type, universally unique identifier, and so on) [45]. All this information can be readily encoded in a GGEP extension by the service mirror builders and then sent to the Gnutella network agent. Then the Gnutella network agent can put this GGEP-formatted information in the Rest part of a Gnutella query message and send it to the Gnutella network.

The query-hit message A query-hit message has the following fields [29]:

Bytes	Description
0	Number of Hits
1-2	Port
3-6	IP Address
7-10	Speed
11-	Result Set

The result set is used for the response to the query and has the following structure [29]:

Bytes	Description:
0-3	File Index
4-7	File Size
8-	File Name; terminated by a null (i.e. 0x00)
x	Extensions Block; terminated by a null byte

The first three fields of the result set are defined specifically to hold information about a requested file (or portion thereof); this happens because Gnutella is mainly used for peer to peer file sharing. In our approach it is possible to redefine all of these fields. In order to prevent increased complexity and the extra work needed to define a new specification, we recommend that these fields be filled with some default and fixed labels. In other words these fields of the result set are simply ignored.

GGEP, HUGE, and plain text metadata are all allowed in the extension block. We recommend that the response messages from service mirror builders be formatted in a GGEP extension and sent back to the network in the extensions field of the query-hit message.

Chapter 5

Conclusions

Service discovery plays an important role in pervasive computing. At the same time pervasive computing creates many challenges for service discovery protocols, which now need to work properly in a heterogeneous and dynamic environment. One other challenge is the problem of remote service discovery.

We described in Chapter 3 two architectures that enable the service discovery protocols and in particular UPnP to discover remote services. Similar to their attempts, we introduced a new approach that is decentralized and fully distributed. We therefore believe that our approach offers better compatibility with pervasive computing environments.

The core part of the new architecture is the new function in a UPnP network called service mirror builder and its cooperation with a specialized Gnutella client software to discover remote services and then present these remote services as local ones. Conversely, a service mirror builder can also control local services to serve them as remote services for other, remote service mirror builders.

The service mirror builder can communicate with the specialized Gnutella client software. From the point of view of the local network however the service mirror builder is just a UPnP-enabled device: It can control other services (whenever the respective service is offered outside the local network) but can also offer services (the mirrors of the remote services); overall, it is just a normal service discovery-enabled device. We used UPnP for illustration purposes, but the service mirror builder can be defined based on *any* service discovery protocol (Bluetooth, Apple Bonjour, etc.) as defined in Section 1.2. Our solution is in fact general and not dependent on any particular service discovery protocol.

The very definition of pervasive computing is distributed and mobile computing [1]. In this dissertation we propose the Gnutella protocol as a distributed search protocol for discovering remote services. The very design of a Gnutella network as a decentralized and distributed protocol moves this remote service discovery architecture one step ahead toward truly distributed computing. In other words, Gnutella helps this remote service discovery architecture to be more compatible and adapted to the pervasive computing environment, as opposed to both the Atom base and Presence service solutions, which have a centralized architecture.

We used the basic Gnutella protocol in our architecture. During the years many changes and refinements have been added to the Gnutella protocol. Some refinements and new techniques like Ultrapeers and Leaves, Distributed Hash Table, Query Routing Protocol (QRP), and so on helped to reduce the traffic in Gnutella network and have increased the efficiency of the protocol. These refinements can be trivially added to our architecture.

5.1 A Critical Comparison with Related Work

The main focus of the DOTOCS solution (Section 3.3) is the optimization of the peer to peer search mechanism. Its authors believe that this search mechanism is best suited for content sharing (audio, video and digital data) among UPnP gateways.

For the actual communication between UPnP gateways they refer to a transparent interaction solution [34] which enables two connected local UPnP networks to discover and control each others' services. Overall, we note that in the DOTOCS case neither the functionality and cooperation of its transparent solution, nor the peer to peer search mechanism are discussed in any detailed way. We also note that DOTOCS only aims to enable media content sharing. As a consequence, we will not include DOTOCS in our comparison; we will instead compare our approach with DOTOCS' underlying transparent interaction solution [34] (named "transparent interaction solution" henceforth).

A good evaluation of the previous work outlined in Sections 3.1 and 3.2 can be found in Chapter IV of [23]. Partially based on this evaluation we now compare our architecture with these two previous approaches as well as the transparent interaction solution (from the previous paragraph).

- *Compatibility with the available trends and technologies:*

  Atom-based solution. Each local network needs to be enhanced with a UPnP Device Aggregator to be able to discover remote services. Each remote network must also be enhanced with Enhanced control points and devices to be able to serve a service to a remote UPnP Device Aggregator.

  Presence service-based solution. Each local network must have a service discovery gateway and a service virtualizer. In addition, all local networks need to be connected to one or several presence servers by means of presence service applications.

  Transparent interaction solution. Each local network has to be enhanced with a specialized UPnP gateway which encapsulates/extracts messages into/from SOAP messages.

Gnutella-based solution. Each local network must be enhanced with a service mirror builder and a Gnutella agent software. There is no need for any central server, and remote service discovery is accomplished as soon as a Gnutella servent connects to the Gnutella network (in cooperation with the respective service mirror builder).

- *Scalability:*

Atom-based solution. The main shortcoming of this solution is the need for VPN. Indeed, VPN does not scale well, as it requires careful administration of IP addresses and subnetworks [23]. Additionally, there is a need in pervasive computing for general solutions that are able to support heterogeneity. VPN also limits the architecture to some domains within the VPN network.

Presence service-based solution. We do not believe that this architecture has any scalability limitation. A presence service application can be used for instant messaging with support for millions of users. The only potential problem is the availability of presence servers in geographically remote areas. Servers geographically very far from the presence server may not be able to serve the clients very well due to possibly frequent disconnections.

Transparent interaction solution. Local networks are connected to the Internet and implement Network Address Port Translation (NAPT). Scalability between local networks is manageable. However, each gateway multicasts in its local UPnP network any received discovery message (regardless whether the demanded service in that discovery message is locally available or not). This creates substantial traffic in the local network, most of it useless. Therefore the transparent interaction solution reduces the scalability of the local UPnP networks.

Gnutella-based solution. If only the basic Gnutella protocol is used the architecture does not scale well because of the fully distributed nature of the Gnutella architecture that requires the exchange of many messages of different types. However, the network traffic can be easily lowered by using the recent enhancements of the protocol (such as Ultrapeers, or servents that perform some extra functions such as indexing [35]), case in which our architecture scales very well.

It is worth noting that as opposed to the transparent interaction solution, our protocol does not multicast remote requests to the local network (for indeed the service mirror builder has already discovered the locally available services), so the local UPnP network will not be loaded with spurious messages.

- *Remote service discovery for pervasive computing:* All four approaches support remote service discovery for pervasive computing.

- *Distributed architecture:*

Atom-based solution. The main focus of this research is remotely accessing a home network from another network. Therefore all remote service discovery requests are addressed to the home network. This architecture can thus be considered centralized or partially centralized: there are some service coordinators (named UPnP Device Aggregators) to register and cache services [17] .

Presence service-based solution. This architecture is also partially centralized. Presence servers serve as service coordinators to cache and register services. Remote service providers and remote service requesters must first find a presence server to register or request a service. Although presence servers (as service coordinators) provide service visibility, the benefit does not come without cost and complexity [16, 17].

Transparent interaction solution. The main focus of this research is remote service discovery between two connected UPnP networks and so their architecture cannot be considered distributed.

Gnutella-based solution. Our architecture is fully distributed: no centralized coordinator is necessary. Each service mirror builder in cooperation with a Gnutella agent software can provide remote service to others, run itself as a service discovery server, and responds to remote service discovery requests for its own services. In other words all servents in this architecture are equal in performance, and are both clients and servers at the same time.

If Ultrapeers are used, then the architecture changes to hybrid. A hybrid approach allows fully distributed service discovery servers to co-exist with some coordinators [17]. Ultrapeers in this case are servents that perform some extra functions such as indexing [35]. We note that robustness is maintained in an Ultrapeers structure, as typically any servent can become an Ultrapeer if it satisfies certain criteria and so the loss of an Ultrapeer does not compromise the network.

- *Security:*

  Atom-based solution. Security depends on the existence of a secure VPN tunnel which is used for the connection between remote networks and the home network.

  Presence service-based solution. Security is accomplished by applying access control policies at the level of service presence servers.

  Transparent interaction solution. Security and privacy are not considered.

  Gnutella-based solution. The focus of this work is service discovery. Security is more important in the service control phase which follows service discovery and is outside the scope of this work. Although the scope of this work does not allow for security issues, it is worth mentioning that authentication mechanisms based on Web Services Security (WS-Security, WSS) [17] can offer a good solution for allowing only authenticated users to use remote services.

Bibliography

[1] B. Abdualrazak, Y. Malik, and H.-I. Yang, *A taxonomy driven approach towards evaluating pervasive computing system*, in Aging Friendly Technology for Health and Independence, vol. 6159 of Lecture Notes in Computer Science, Springer, 2010, pp. 32–42.

[2] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, 2008.

[3] Apple Inc., *Open Source Development Resources.* http://developer.apple.com/opensource.

[4] P. Belimpasakis, *Seamless User-Generated Content Sharing in the Extended Home*, PhD thesis, Julkaisu-Tampere University of Technology, 2009.

[5] P. Belimpasakis and V. Stirbu, *Remote access to universal plug and play (UPnP) devices utilizing the Atom publishing protocol*, in International Conference on Networking and Services, IEEE Computer Society, 2007, p. 59.

[6] M. Bell, *Service-Oriented Modeling: Service Analysis, Design, and Architecture*, Wiley, 2008, ch. Introduction to service-oriented modeling.

[7] Bluetooth Special Interest Group (SIG), *Specification of the Bluetooth System Version 1.1*, 2001. http://www.tscm.com/BluetoothSpec.pdf.

[8] E. Buyukkaya, M. Abdallah, and R. Cavagna, *VoroGame: A hybrid P2P architecture for massively multiplayer games*, in 6th IEEE Consumer Communications and Networking Conference (CCNC), IEEE, 2009, pp. 1–5.

[9] S. Chen, D. Katsaros, A. Nanopoulos, and Y. Manolopoulos, *Wireless Information Highways*, IRM Press, 2005.

[10] Y. Chen, *Service oriented architecture*, Journal of Healthcare Information Management, 24 (2006), pp. 45–52.

[11] Clip2 Distributed Search Services, *The Gnutella Protocol Specification Version 0.4*. http://www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf.

[12] B. Cohen, *The BitTorrent Protocol Specification*, 2008. http://www.bittorrent.org/beps/bep_0003. html.

[13] M. Day, J. Rosenberg, and H. Sugano, *A Model for Presence and Instant Messaging*, Internet Engineering Task Force, 2000. RFC 2778.

[14] M. Debbabi and M. Rahman, *The war of presence and instant messaging: right protocols and apis*, in 1st IEEE Consumer Communications and Networking Conference (CCNC), 2004, pp. 341–346.

[15] *Domotique et informatique mobile (DOMUS).* Université de Sherbrooke; http://domus.usherbrooke.ca.

[16] P. Engelstad, Y. Zheng, and J. Tore, *Service discovery and name resolution architectures for on-demand MANETs*, in 23rd International Conference on Distributed Computing Systems, IEEE Computer Society, 2003, pp. 736–742.

[17] W. Feng, *Remote service provision for connected homes*, PhD thesis, De Montfort University, 2010.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *The Hypertext transfer protocol—HTTP/1.1*, Internet Engineering Task Force, 1999. RFC 2616.

[19] Y. Y. Goland and J. Schlimmer, *Multicast and unicast UDP HTTP messages*, 2000. UPnP Forum Technical Committee Draft: http://www.upnp.org/resources/specifications.asp.

[20] J. Gregorio and B. de Hora, *The Atom Publishing Protocol*, Internet Engineering Task Force, 2006. RFC 5023.

[21] Gtk-Gnutella, *Gnutella Bootstrapping*. http://gtk-gnutella.sourceforge.net/en/?page=bootstrap.

[22] E. Guttman, C. Perkins, J. Veizades, and M. Day, *Service Location Protocol*, Internet Engineering Task Force, 1999. RFC 2608.

[23] A. Häber, *Remote Service Discovery and Control for Ubiquitous Service Environments in Next-Generation Networks*, PhD thesis, University of Agder, 2010.

[24] T. Hodes, S. Czerwinski, B. Zhao, A. Joseph, and R. Katz, *An architecture for secure wide-area service discovery*, Wireless Networks, 8 (2002), pp. 213–230.

[25] D. Ilie, *Gnutella Network Traffic-Measurements and Characteristics*, Master's thesis, Blekinge Tekniska Högskola, 2006.

[26] J. Jeon, J. M. Lee, K. J. Myoung, K. R. Lee, W. H. Kwon, and B. Ko, *Design and implementation of the HNCP-UPnP bridge using a virtual device*, in International Symposium on Power-Line Communications and Its Applications, 2004, pp. 357–361.

[27] E. Kawamoto, K. Kadowaki, T. Koita, and K. Sato, *Content sharing among UPnP gateways on unstructured P2P network using dynamic overlay topology optimization*, in 6th IEEE Consumer Communications and Networking Conference (CCNC), IEEE, 2009, pp. 1–5.

[28] D. Kim, J. Park, P. Yevgen, K. Moon, and Y. Lee, *IEEE 1394/UPnP software bridge*, IEEE Transactions on Consumer Electronics, 51 (2005), pp. 319–323.

[29] T. Klingberg and R. Manfredi, *Gnutella 0.6*, Network Working Group, 2002.

[30] Q. Mahmoud, *Service-Oriented Architecture (SOA) and Web Services: The road to Enterprise Application Integration (EAI)*, Oracle Corporation, 2005. http://www.oracle.com/technetwork/articles/javase/soa-142870.html.

[31] Microsoft Developer Network, *Overview of UPnP Architecture*, 2010. http://msdn.microsoft.com/en-us/library/aa382261%28v=vs.85%29.aspx.

[32] M. Nidd, *Service discovery in DEAPspace*, Personal Communications, IEEE, 8 (2001), pp. 39–45.

[33] M. Nottingham and R. Sayre, *The Atom Syndication Format*, Internet Engineering Task Force, 2005. RFC 4287.

[34] M. Ogawa, H. Hayakawa, T. Koita, and K. Sato, *Transparent UPnP interactions over global network*, in Proceedings of SPIE, vol. 6794, 2007, p. 67944P.

[35] A. Oram, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, O'Reilly Media, 2001.

[36] *Understanding Universal Plug and Play.* White paper: http://www.upnp.org/download/UPNP_understandingUPNP.doc.

[37] B. A. S. Cheshire and E. Guttman, *Dynamic Configuration of IPv4 Link-Local Addresses*, Internet Engineering Task Force, 2005. RFC 3927.

[38] The Salutation Consortium Inc., *Salutation Architecture Specification.* 1999.

[39] A. Sameh and R. El-Kharboutly, *Modeling Jini-UPnP bridge using rapide ADL*, in IEEE/ACS International Conference on Pervasive Services, IEEE Computer Society, 2004, pp. 237–237.

[40] R. Schollmeier, *A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications*, in 1st International Conference on Peer-to-Peer Computing, aug 2001, pp. 101 –102.

[41] R. Srinivasan, *RPC: Remote Procedure Call Protocol Specification Version 2*, Internet Engineering Task Force, 1995. RFC 1831.

[42] C. A. Steinkuehler, *Learning in massively multiplayer online games*, in 6th international conference on Learning sciences, 2004, pp. 521–528.

[43] Sun Microsystems Inc., *Jini Technology Core Platform Specification Version 1.2*, 2001. http://www-csag.ucsd.edu/teaching/cse291s03/Readings/core1_2.pdf.

[44] *UPnP forum.* http://www.upnp.org.

[45] UPnP forum, *UPnP Device Architecture 1.1*, 2008. http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

[46] S. Vaughan-Nichols, *Presence technology: More than just instant messaging*, Computer, 36 (2003), pp. 11–13.

[47] M. Weiser, *Some computer science issues in ubiquitous computing*, Communications of the ACM, 36 (1993), pp. 75–84.

[48] H. Wong, *Developing Jini applications using J2ME technology*, Addison-Wesley, 2002.

[49] J. York and P. C. Pendharkar, *Human-computer interaction issues for mobile computing in a variable work context*, International Journal of Human-Computer Studies, 60 (2004), pp. 771–797.

[50] F. Zhu, M. Mutka, and L. Ni, *Service discovery in pervasive computing environments*, Pervasive Computing, 4 (2005), pp. 81–90.