User Preference Based Service Discovery*

Jongwoo Sung, Dongman Lee, Daeyoung Kim[1]

Information and Communications University, Korea
{jwsung, dlee, kimd}@icu.ac.kr

Abstract. Existing service discovery protocols are designed to take less consideration of different service qualities. Consequently, after the client discovers the services using specific service type information, selecting the best qualifying one among identical services is left to the user or application developers. This paper proposes a quality of service discovery framework which enables a client to discover and select the qualifying service by considering different service qualities based on the weighted preference of the user. We implement and evaluate the prototype system performance and compare it with standard UPnP protocol.

Keywords: Quality of service discovery, user preference, UPnP.

1 Introduction

In a spontaneous network [1] where multiple services and devices cooperate with each other without any user involvement, discovering a qualifying service among different quality services is important. A service discovery protocol allows a service client to discover a qualifying service on a network consisting of heterogeneous devices and services. To discover a service in the network, UPnP (Universal

* [1] This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Advancement)(IITA-2006-C1090-0603-0047) and the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government(MOST) (No. R0A-2007-000-10038-0).

Plug and Play) [2] uses service types and service description while Jini [3] uses service interface and additional static attributes. SLP (Service Location Protocol) [4, 5] adopts string-based service attributes which come with query operators like AND, OR and EQUAL.

However, existing service discovery protocols take less consideration of the quality of a service: if there exist more than one service qualifying to a user's service request, they simply choose one of them based on either the order of appearance or randomly. Consequently, after a client discovers target services using a specific service type or through interface information, selecting the best qualifying one from the discovered services is left to the user or application developers.

Each service instance can be characterized by attributes consisting of a functional service type (cf. printer) and describing characteristics (cf. location) [6]. For service discovery of fine granularity, various service attributes as well as service type should be taken into consideration. However, such service discovery is not a simple problem for two main reasons. First, because services are described by the collection of heterogeneous attributes, it is difficult to compare them using simple string operations. Second, since qualified services are scattered across a network(s), comparison among them may be time consuming.

In this paper, we propose a new service discovery scheme which discovers and selects a service that most satisfies the user's quality requirement among qualified service candidates. The key features of the proposed scheme are as follows: 1) it does not depend on any central server for discovery; 2) to compare heterogeneous service qualities efficiently, a scoring algorithm exploiting the user preference table and service descriptions is introduced; and 3) it uses a response collision avoidance mechanism which removes unnecessary responses, keeping service responses from imploding the client.

2 Related Work

Service discovery is used to locate available services by service type; while service selection is concerned with a particular service among services of the same type [7]. In this sense, many established service discovery protocols such as UPnP, JinI and SLP are closer to service discovery than service selection. These discovery protocols find

multiple target services of a specified type, while the selection of one from the discovered candidates is left to the user or software developer.

There have been some researches that have enhanced existing service discovery protocols to have better gratuity. The researches in [6] and [7] aim to expand Jini lookup services in order to utilize a variety of service attributes. Researchers in [12] and [13] propose attributes-based extensions for easy service selection based on SLP, which allows the service list to be returned in a sorted order so that the client can easily select the most qualifying service from the candidates.

Quality of service and service discovery are considered together in [8], but that research focuses only on QoS negotiation. The research in [9] integrates network sensitivity into the service discovery process; while [10] describes QoS-aware service discovery for mobile ad-hoc networks. Context or preference service discovery are also proposed in [11, 21, 23].

3 Design considerations

For designing an efficient service discovery protocol which allows discovery and selection from identical services with different quality of service, the following should be considered:

3.1 Decentralized architecture

A centralized system fundamentally implies administrative setup and pre-knowledge of discovery environments. Unlike this, the proposed system design does not depend on any central systems. We assume that all services on the network spontaneously cooperate for providing an answer to a quality of service discovery query. Specifically, each device (service provider) calculates their conformance score based on user defined preferences by themselves.

3.2 Qualifying service selection

In order to compare one service quality with others it is required to normalize service characteristics. In addition, an effective way to express user preferences must be created. For this, the service

description table and the user preference table are provided for the service provider and the client, respectively. A user can set his preferences in the preference description table, and each preference is represented by a weighted significance value for scoring. Then a service score, which is calculated by the server itself, indicates the degree of service qualification against the user preference.

3.3 Network implosion avoidance

Multicast is a popular tool for service discovery when there is no centralized server. Multicast query-multicast response and multicast query-multiple unicast response can overrun the client to the point of implosion. Because this may degrade the overall performance of the system, proper service discovery has to alleviate this problem. We use multicast request-multicast response, and our scoring based jitter delay allows minimum network response messages to transfer.

4 Architecture

The proposed architecture uses an aggressive service discovery approach in which a client sends a service discovery query to service devices instead of waiting for services to announce their existence. The proposed quality of service discovery consists of three tightly coupled phases: 1) sending quality of service discovery queries which describe expected service attributes by a client; 2) service score calculation on each device; and 3) score-based service response. The following sub-sections explain each phase in more detail.

4.1 Quality of Service Discovery Query

To find the most qualifying service, a user needs to describe the expected service attributes which the client is looking for. In the similar way, available services can be described with multiple attributes and their value pairs by servers. User preference based quality of service discovery problem can be conceptually summarized; there are scattered services which have different featured attributes on the networks, and a client describes expected service attributes, which are sent to network

and compared to each services. Comparisons of available service's attributes and expected service's attributes are conceptually shown in Fig 1.

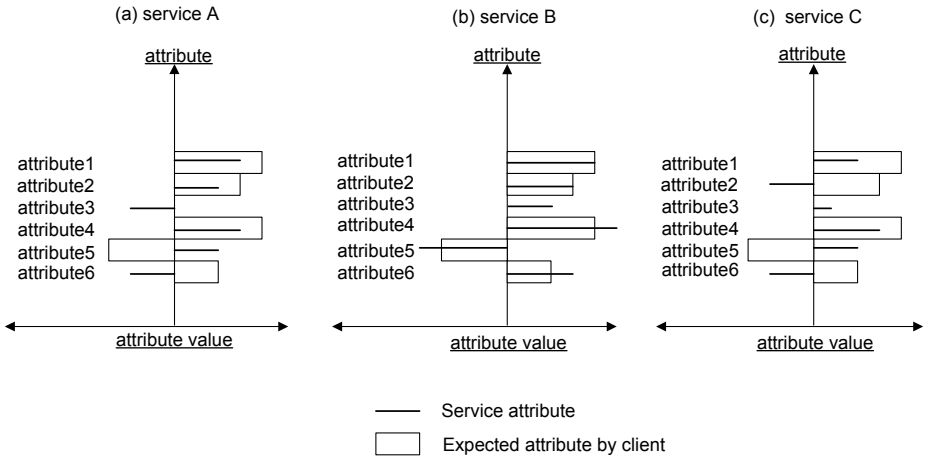

Fig. 1 Conceptual service selection process (Service A. and Service C. have several attributes which do not meet the expectation of client while service B. show best matching to client expectation)

The *preference description table* in Fig 2 is a list of **attributes**, **its value**, **priroity** and **arithmetic operators** maintained by a client.

Preference description table = a list of { service attributes, attribute value, significance, arithmetic operators }

Fig. 2. Favorite and preference description table

A property "**Arithmetic operators"** (>,<.=) explain attribute value condition (for example, delay time is lower than 3 min). Operations may be expanded to include richer and more complex ones as in [19] and [21]. The "**priority**" property expresses the significance of its attribute compared with other attributes. For the illustration purpose five different priority levels (Compulsory, Important, Normal, Trivial, Reject) are defined. Four levels fall into positive categories while the other (reject) level has negative connotations. We explain *preference description table* with a simple printer example in Fig. 3.

Attribute name	priority	Arithmetic	Value

		operator	
Service Type	Compulsory	EQ(=)	Printer
Usability (Waiting time)	Important	LT 10(>=)	10 min
Accessibility (location)	Important	EQ (=)	R507
Color support	Trivial	GT 256 (>=)	16 Bit
Pay printer	Reject	EQ(=)	No
Need authorization	Reject	EQ (=)	Yes

Fig. 3 Preference description table

In this example, six **attributes**, **value**, **priority** and **arithmetic operators** are defined to express expected user preferences.

After user sets categorized ***priority*** level to each attribute according its importance, it is necessary to change nominal attributes with a numeric value for calculations. Two strategies are used for this quantification. First, we assign high numeric value to more important attributes. Second, we assume that a client who request high priority attributes cannot be satisfied with less priority attributes even though all low priority attributes are qualified. This assumption can be explained with this example. Suppose that there is a service which does not meets "*service type* (e.g., printer)" **attribute** which is ranked as a nominal "*compulsory*" **priority** level. Then, there is the least possibility for a user to choose it.

We use the function of S(n) to denote n^{th} (n>=1) priority value with priority ordering from 1(Compulsory) to 5(Reject). Then Fig 4 explains a simple mechanism to calculate n^{th} (n>=1) priority value.

$$S(n) = \sum_{1}^{n} A(n-1) \times S(n-1) + 1, (n >= 1), A(0) = S(0) = 0$$

Fig. 4 Preference quantification

where *A(n)* is the number of **attributes** which have n^{th} **priority** level. The n^{th} **priority** value S(n) can be calculated by adding guard score "1" to sum of all low level attribute's **priority** value between 0 to

(n-1). The calculation process starts from low **priority** level (n=1), and it continued to high **priority** (n) based on lower **priority** value.

Negative **priority** level diminishes the chances of a service that meets the negative significance to be chosen. For example, if the printer service has an **attribute** of "pay printer" which is ranked in the "reject" **priority** level, it would be the last candidate regardless of other good **attributes**.

The *preference description table* may be constructed automatically with GUI based program and written in an XML format or simple text. A client sends a service discovery query including this *preference description table* via a multicast channel. Here, scoring calculation may be done by servers to alleviate the processing overhead on limited client devices.

4.2 Service conformance score calculation

Like the preference description table of the service discovery client, participating services maintain their own *service description table* which consists of **service attributes** and their **value** pair.

Once services on multicast channels receive multicast query messages from a client, they find a *preference description table in query messages*. To calculate *conformance score* initially it is set to zero. If an **attribute value**, which describes a service state in the *service description table*, meets an attribute condition in the *preference table*, a corresponding **priority** value is accumulated to the service score. For the simplicities purpose, we only use true or false judgment for each attribute attributes. The *conformance score* is achieved by Fig.5.

Given a priority value S=$\{s_1 \cdots s_n\}$ where attribute list of preference description table A=$\{a_1 \cdots a_n\}$, and a attribute list of service description table B=$\{b_1 \cdots b_n\}$, a conformance score C is defined as the sum of priority value where $A_n = B_n$.

$$c = \sum S_n \text{ such that } (A_n = B_n)$$

Fig. 5 conformance score calculation

As a result, a device with a high *conformance score* is considered as a more qualifying service compared to those with low scores.

4.3 Multicast response and service selection

Services return back a discovery response message including a *conformance score* to a client. If multiple response messages from services are sent to a client in a short period of time, they can implode the client [4,15,18,22]. To overcome this response implosion problem, some established service discovery protocols (e.g. UPnP) uses jitter delay, which all service have to wait random period of time before sending their responses.

Our response collision avoidance mechanism is also based on jitter delay, but we assign different jitter delay to services according to their calculated *conformance score*. The jitter calculation algorithm is shown in Fig. 6.

$$PerfectScore = \sum_{n=1}^{m} S(n) \times T(n)$$

$$DelayJitter = (\log_2^{PerfectScore - ConformanceScore})^n \times Scale$$

Fig. 6 Jitter calculations

where m is the number of different **priority** level (we use 4 because there are four priority levels from a trivial to a compulsory), S(n) is n^{th} priority value and T(n) is the n^{th} number of attributes. *PerfectScore* is the calculated score when a service has all **attributes** expected by a client. *DelayJitter* is calculated to give high delay to a low score matching service. Because a linear function gives same interval delay regardless of their *conformance score*, the total delay time may be impractically long. We use logarithms based jitter calculation so that relatively good qualified services can have enough delay difference to be distinguished while less qualified services have small delay difference among different services. As a result, it is assured that multiple qualifying services can have long jitter delay enough to distinguish it with responses from similar services, while less important services may pass over jitter delay quickly to reduce maximum delay

time. Variables *n* and *Scale* are used to characterize calculations in real implementations.

The service which has expired jitter delay sends respond messages to a client, and automatically, the first arriving service is the most qualifying service to the client's preference because it can be thought of as having the highest score according to our jitter calculation.

Throughout this process because all service responds to a client, it would result in a waste of precious network bandwidth and a client resource. To avoid this inefficiency we utilize multicast protocol for service discovery responses. This services which receive multicast response message from any other service stop to delay for responding., and simply cancel remaining process.

As a result of our service discovery process, the multicast request and multicast response mechanism alleviates the network implosion problem effectively, and our scored based delay Jitter response assures the right selection of the most qualifying service.

5 Implementation

We implement the proposed system, and compare its performance with the standard UPnP protocol. The testing environment consists of six desktop computers which have Pentium 4 CPU 3GHz and 1GB RAM and 100Mbps Ethernet. We implemented preference description tables and service description tables in client side and server side respectively. We programmed the preference quantification, conformance score calculation and jitter calculation mechanism as explained before.

The UPnP system is implemented based on Intel UPnP [17], and we modify both the UPnP device (services) and the UPnP control point (client). Most modifications are applied into the network layer in SSDP (Simple Service Discovery Protocol) codes of UPnP protocol stacks. For clarity purposes, we disabled periodic service notification announcements and ignoring cache control. Because originally UPnP does not differentiate the quality of services if they are of identical type, we modified the UPnP client to accept continuous responses even after it receive same service.

6 Performance Evaluation

In the experimental we made virtual printer example which have 25 attributes. We defined 5 significance levels from 1(compulsory) to 5(reject), and we set the identical number of attributes which have each significance level. We set n=1 and *scale*=1 for jitter calculation. Because performance of proposed service discovery depends on distributions of services, we assume services (service attributes) are uniformly distributed. We use only one service requestor for simple comparison purposes, and increase the number of devices from 0 to 100 with 5 steps.

We evaluate three factors: network traffic, time delay and accuracy. In our evaluation we compare prototype system's performance with standard UPnP. Because UPnP does not support quality of service discovery, we use UPnP's operations (GET/SET) for retrieving service descriptions from services. In UPnP, total network bandwidth is the sum of service discovery request/response message, communication overhead for service descriptions transfer from services. Similarly, service delay is the sum of service discovery request/response delay, communication delay for service descriptions and descriptions sorting delay.

Network traffic is measured by the network capturing tool, Etherreal [16]. The total request and response message sizes are monitored and the result of network bandwidth is shown in Fig. 9. The bandwidth of service discovery increases as the number of services increases.

As a result UPnP bandwidth will increase linearly, but our prototype shows more stable results.

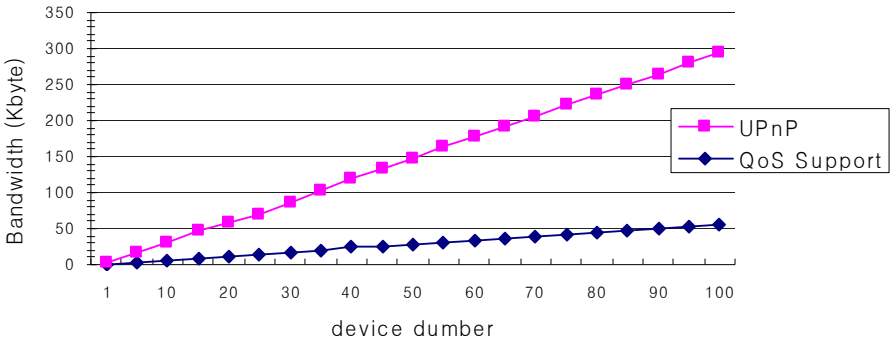

Fig. 9 Network bandwidth

Because the proposed service discovery model consists of one multicast request and one multicast response, the network bandwidth is always the same regardless of device numbers.

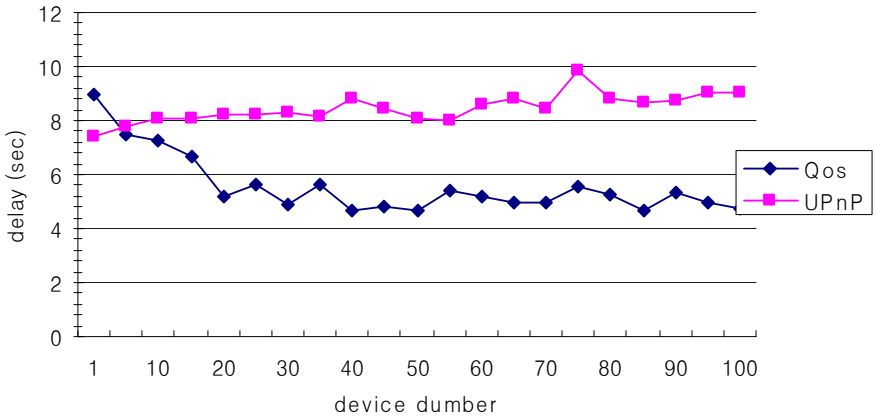

Fig. 10 Service response time delay

Service discovery delay is the total amount of time it takes from the client's request to the discovery of one qualifying service. Fig. 10 shows the time delay of the proposed system. UPnP has an MX(default is 3sec) value on its multicast request query header indicating the maximum waiting time. UPnP sends service discovery query two times.

On the other hand, the time delay of the proposed system shows fast response time. In addition, because the increased device numbers give high probability of good services overall response time is increased.

Because the client needs to communicate with multiple services at the same time, it is easy to experience network errors. We measure error rates (number of expected messages - actual message number)/ number of expected messages *100. In Fig 11, UPnP have more packet errors when number of device is increased while our system shown little error rate. This may come from packet collisions, and when devices are too crowded, our delay jitter scheme failed to successfully differentiate delay difference among services.

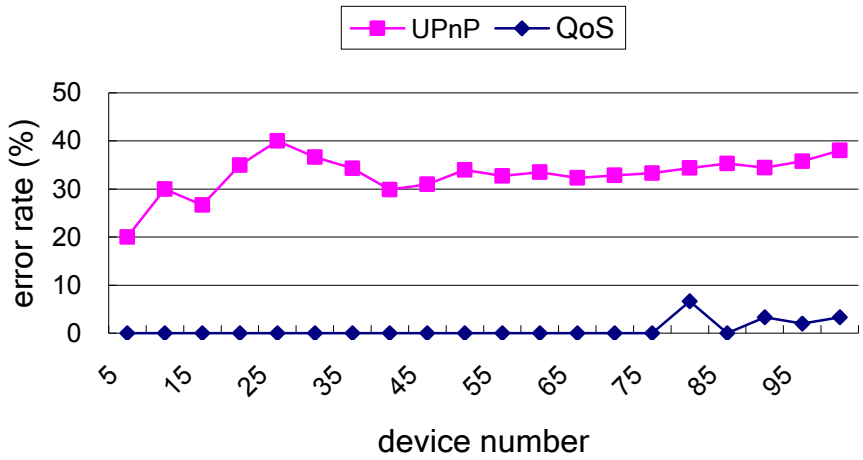

Fig. 11 service discovery response error rate

In our experimental a client can composite service with maximum 1290 score (when the client sets five attributes for each priorities), and worst case waiting for service discovery response are 10.33 seconds. Because we use weighted delay calculations according to a service's good qualities, the most qualifying service is assured to have at least 1 second delay difference with next qualifying service. This prohibits services are miss-chosen due to simultaneous responses by other good services. However, this long delay difference affects overall response time, we diminish delay gap between less qualifying services. This is because considering a few good services are more important than taking care of many less qualifying services. However, this may cause

the least qualifying service to have small delay (0.001 seconds) gap, which can lead response duplications problems.

6 Conclusion

In this paper, we have proposed a new service discovery based on the user's QoS requirement which finds the service based on preferred service quality. To minimize network traffic and client overhead, service description comparison processing is moved from the client side to service devices. The proposed service discovery consists of three tightly coupled phases: 1) preference advertising by the client; 2) candidate service score calculations at each device; and 3) qualifying service response.

To prove the feasibility of our system, we compare the performance of our system against standard UPnP. To the best of our knowledge, we have uniquely categorized some service discovery mechanisms in the point of quality of service discovery. We will study better jitter control to minimize duplicate multicast responses as our future work.

References

[1] Stephan Preu, Clemens H. Cap,"Overview of Spontaneous Networking - Evolving Concepts and Technologies," Future Services for Networked Devices (FuSeNetD) Workshop, Heidelberg, Germany, November 8-9, 1999.

[2] UPnP(Universal Plug and Play) forum, http://upnp.org/.

[3] Jini, http://jini.org/.

[4] E. Guttman. "Service location protocol: Automatic discovery of IP network services," IEEE Internet Computing, 3(4):71-- 80, 1999

[5] E. Guttman, C. Perkins , RFC2608, Service Location Protocol, Version2, June 1999.

[6] M.B Moller, B.N. Jorgensen, "Enhancing Jini's Lookup Service using XML-based Service Templates," IEEE tech-nology of Object-Oriented Language and Systems, March 2001, Pages:19-31.

[7] Choonhwa Lee; Helal, S, "Context attributes: an approach to enable context-awareness for service discovery," IEEE Symposium on Applications and the Internet, 27-31 Jan. 2003.

[8] Minyoung Sung and Gui-young Lee, “A Qos-enabled Service Discovery and Delivery Scheme for Home Networks,” IEEE International Conference on Local Computer Networks(LCN’03).
[9] A.-C. Huang and P. Steenkiste, “Network-Sensitive Service Discovery,” In USITS, 2003.
[10] Jinshan Liu, Valerie Issarny, “Qos-aware Service Location in Mobile Ad-Hoc Networks,” IEEE International conference on Mobile Data Management(MDM04), Jan 2004
[11] Ryota Egashira, Akihiro Enomote, Tatsuya Suda, “Distributed and Adaptive Discovery Using Preference,” Proc. of the Workshop on Service Oriented Computing in the IEEE SAINT Symposium, 2004
[12] W. Zhao, H. Schulzrinne, E. Guttman, C. Bisdikian, W. Jerome, Select and Sort Extensions for the Service Location Protocol (SLP), Internet experimental RFC 3421, November 2002.
[13] Weibin Zhao and Henning Schulzrinne, " Improving SLP efficiency and extendability by using global attributes and preference filters," in International Conference on Computer Communications and Networks (ICCCN'02), Miami, Florida, October 2002
[14] Y. Y. Goland and et.al, Simple Service Discovery Protocol /1.0, Internet Draft, draft-cai-ssdp-v1-03.txt, Work in Progress, Oct 1999.
[15] K. Mills, C. Dabrowski, C, ”Adaptive jitter control for UPnP M-search,” IEEE International Conference on Communications, Volume 2, 11-15 May 2003 Page(s):1008 - 1013 vol.2.
[16] www.ethereal.com/.
[17]Intel software for UPnP Technology,
http://www.intel.com/technology/upnp/.
[18] Chaiporn Jaikaeo, Chavalit Srisathapornphat, and Chien-Chung Shen, "Diagnosis of Sensor Networks," in IEEE International Conference on Communications (ICC 2001).
[19] Howes, T., "A String Representation of LDAP Search Filters," RFC 2254, December 1997.
[20] G. Thomson, M. Richmond, S. Terzis, and P. A. Nixon, "An Approach to Dynamic Context Discovery and Composition," Proceedings of Ubisys: System Support for Ubiquitous Computing Workshop, UbiComp, Seattle, Washington, 2003.