Universal Adaptor: A Novel Approach to Supporting Multi-protocol Service Discovery in Pervasive Computing

Joanna Izabela Siebert[1], Jiannong Cao[1], Yu Zhou[1,2], Miaomiao Wang[1,3], Vaskar Raychoudhury[1],

[1] Department of Computing,
The Hong Kong Polytechnic University,
Kowloon, Hong Kong

[2] Department of Computer Science and Technology,
Nanjing University,
Nanjing, China

[3] The University of Science and Technology of China,
Hefei, Anhui, China

{csjsiebert, csjcao, csyuzhou, csmmwang, csvray}@comp.polyu.edu.hk,

Abstract. Service discovery is an important and challenging issue in pervasive computing. To date, many service discovery protocols have been proposed and new ones are under development. However, pervasive computing involves applications running in heterogeneous environments and application developers must cope with diversity of network infrastructures, middleware platforms, and hardware devices. There is a need for integrating or bridging these service discovery schemes in order to support the discovery of the services available in different environments. In this paper we study how to provide service discovery across different environments supported by different service discovery systems, which may use standard protocols as well as tailor-made mechanisms. We propose the Universal Adaptor (UA) approach, which consists of two major components: the Universal Adaptor Primitives (UAP) and the Universal Adaptor Mapping (UAM). UAP is the universal set of primitives used by the user to discover the services across different environments, while UAM provides the mapping between the Universal Adaptor Primitives to the primitives used in various service discovery systems. We have developed a prototype of the proposed approach and we describe the implementation issues and the experiment results.

Keywords: Pervasive computing, Service discovery, Heterogeneous environments

1. Introduction:

Along with the advances in pervasive computing technologies, users are no longer constrained to only a single computing environment but can work across different environments, meeting with a diversity of software, hardware devices, and network infrastructures. Heterogeneity of the environments brings significant problems that application developers must cope with. For instance, devices must be able to discover and share services among each other. However, manual configuration may require special skills together with long time to set up. Therefore, service discovery is an important, challenging task in pervasive computing.

Many service discovery protocols have been proposed. Examples include UPnP, SLP, and Jini [1, 2, 3]. They allow automatic detection of hardware devices, software, and other resources in a computing environment along with services offered by the various kinds of entities. Existing service discovery protocols differ in the way service discovery is performed, and no single protocol is suitable for all the environments. Due to the differences in the service discovery approaches implemented, a user working in one environment may not be able to search for the services available in another environment that suits the user's need. To support service discovery across different environments, new techniques are needed to integrate or bridge the service discovery protocols used in a diversity of environments.

Works can be found on providing interoperability of service discovery protocols [11, 12, 13, 14, 15, 16]. In this paper, we study how to provide service discovery across different environments supported by different service discovery systems, which may use standard protocols such as SLP and Jini, as well as tailor-made mechanisms that support multiple protocols within an environment, such as ReMMoC [13].We propose the Universal Adaptor (UA) approach, which consists of two major components: the Universal Adaptor Primitives (UAP) and the Universal Adaptor Mapping (UAM). UAP is the universal set of primitives used by the user to discover the services across different environments, while UAM provides the mapping between the Universal Adaptor Primitives to the primitives used in various service discovery systems. The Uniform Adaptor can be implemented in any environment, independent of the service discovery system used in that environment. This approach enables users to discover available services with no knowledge of the service discovery system adopted in an environment. We have developed a prototype of the proposed approach and we describe the implementation issues and the experiment results.

Comparing with existing approaches, Universal Adaptor provides a simple and flexible solution. It provides only a single set of APIs and supports not only all existing but also future service discovery systems. It is lightweight and easy to implement in diverse infrastructures and to use by users.

The rest of the paper is organized as follows. In Section 2, we present the related work. In Section 3, we present architecture overview of proposed system. Section 4 shows the design of Uniform Adaptor Primitives. Section 5 describes how to realize the Uniform Adaptor Mapping. In section 6, we describe the implementation of a prototype of the proposed UA approach. We conclude the paper in Section 7 with the discussion of our future work.

2. Related work

Many service discovery protocols have been proposed for different applications in various environments [5, 3, 1, 4, 2]. Survey and comparisons between the existing service discovery protocols can be found in [6, 7, 8, 9, 10]. New service discovery protocols targeting at pervasive environments are also emerging [16, 18, 19]. In this section, we focus on the works that support interoperability of the service discovery protocols. Several approaches on supporting multi-protocol service discovery have been proposed [11, 12, 13, 14, 15, 16].

All the existing service discovery mechanisms realize the concept of client/server application. Clients are entities that need some functionality (service) and servers are entities existing in the environment and offering this functionality. Service discovery is the framework for connecting clients with services. Existing works towards achieving multi-protocol service discovery use a middleware approach but differ in where the interoperability support is provided. The middleware can be on the service side, client side, or intermediate entity.

An example of providing the support on the service side is INDISS [14] designed for home networks. INDISS provides parsers and composers, which decompose a request from the source service discovery protocol into a set of events and then compose them into message understood by target service discovery protocol.

INDISS can also be deployed on the client side. Another example of middleware on the client side is ReMMoC [13]. In ReMMoC, the client side framework provides the mappings between all the supported service discovery mechanisms. Abstraction of discovery protocols to the generic service discovery is achieved by using a generic API or doing discovery transparently to the client. In this approach, all possible mappings need to be provided at the client side.

Service side support can also be based on service proxies [16]. When a new service appears or disappears in one of the environments, the framework detects it and creates or removes its proxies from other environments. However, this approach requires dynamic service proxies to be implemented for each environment, which increases developer's workload.

Another way of providing interoperability support is to provide an intermediate entity [12, 11]. In Open Service Gateway Initiative OSGi [12] all the connections and communications between the devices are brokered by a Java-based platform. An internal service registry exists in the framework. Interoperability is achieved by providing an API to map the given service discovery protocol to OSGi and vice versa. Supporting new service discovery protocols requires defining new APIs for them. Gateway functionality is also utilized by protocol adapters in the FuegoCore Service broker [11] designed for mobile computing environments. The service broker registers the mappings between its internal template and the external templates used by different service discovery protocols. Extending the FuegoCore service broker to new service discovery protocols requires creating and deploying additional service discovery protocol adapters. INDISS [14], mentioned above, can be deployed as an intermediate entity as well. The intermediate entity approach requires broker to integrate all the adapters into one system. In a network with a large number of service discovery protocols the framework may not be scalable.

Another approach is providing service that discovers services across different environments [15]. In MUSDAC, it registers itself in all the environments, so clients can use whatever protocol to discover it. However, clients must have the knowledge about MUSDAC and the process of discovering the service has high processing requirements.

The centralized approach has the scalability problem. In the future pervasive environment, an unlimited number of service discovery mechanisms will emerge. Providing all possible translations in one middleware would result in a very heavy system. Our work reported in this paper is based on the analysis of the following requirements on the interoperability system for pervasive computing:

- no change should be imposed on the existing service discovery mechanisms
- no change should be imposed on the services registered in domains
- no functionality of the environment should be compromised
- the system should be lightweight, scalable, and extendable.
- support both standard and tailor-made service discovery mechanisms

Our approach addresses all of these requirements. In the following sections we describe design of the system.

3. System Model and Architecture

The aim of our study is to support interoperability between existing as well as yet unknown service discovery systems.

In this section, we describe the system model of the proposed UA approach. We define an *Environment* as a Service Discovery Domain, where a native service discovery system is able to find a specified resources if they are available in the domain. A native service discovery system can take the form of an existing service discovery protocol, e.g SLP, Jini. Moreover, it can be a tailor-made mechanism, e.g. ReMMoC which supports the interoperability of service discovery within an environment.

Services in an environment can be provided by hardware devices, software, and other entities. Examples of software services can be weather forecast, stock quotes, and language translation. Hardware devices can provide services directly or via another device. For example, the printing service can be provided by a Jini enabled printer, or by a printer that is attached to computer running Jini software.

Figure 1 shows the major entities in the UA system. There are two major components: The Universal Adaptor Primitives (UAP) and the Universal Adaptor Mapping (UAM). UAP acts as uniform interface to the client. The client makes use of UAP to discover and access services. The main function of UAM, on the other hand, is to provide the mapping from UAP to the specific primitives of service discovery protocols (SDP).

The Universal Adaptor (UA) is installed in each environment. On the client side, the UA provides Universal Adaptor Primitives (UAP) for the communication between

the client and UA. On the service side, UA is a component tailored to the environment, mapping the UA primitives to those of the native service discovery system and performing service discovery on behalf of the client.

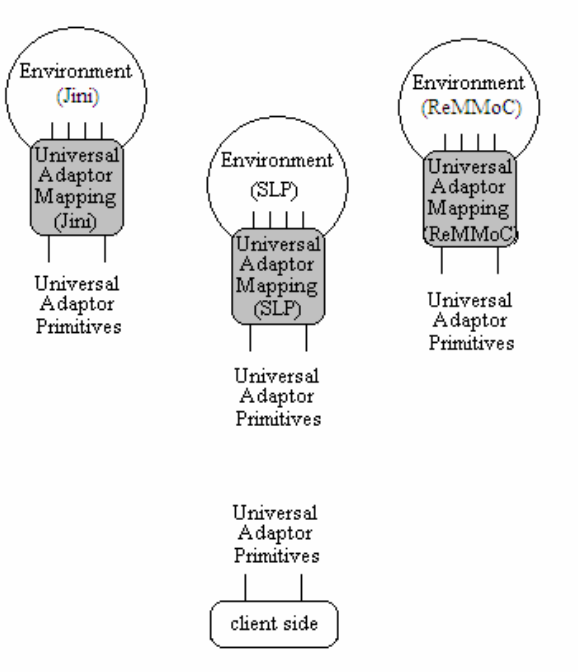

Figure 1. The system model

4. Universal Adaptor Primitives (UAP)

We studied existing service discovery protocols and observed that all the service discovery systems provide the same functionality for end users and differ only in the underlying models and operations. We can abstract the common characteristics of existing approaches and to provide a universal set of primitives that enable service discovery across diverse environments.

The format of the proposed primitives is the following:

[Return_Values] Primitive_Name [Parameters]

Primitive_Name represents the functionality requested by the client; *Parameters* is a set of attributes required in the discovery process; *Return_Values* is set of values returned to the client.

The discovery process starts when the client expresses his interest in a particular service. Next, the discovery system searches for the service and returns result to the client. The common feature of all the existing service discovery mechanisms is that

they support the process of looking for service and enable the access to the service. Therefore, we propose two universal primitives: *Discovery* and *Access*. Specific UA Primitives with parameters are shown in Figure 2.

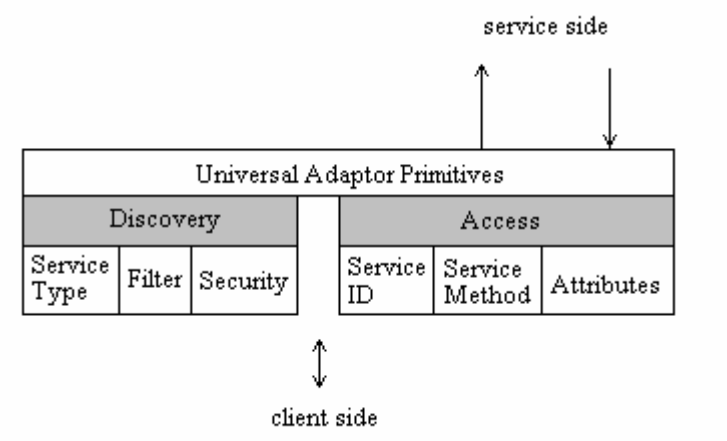

Figure 2. Universal Adaptor Primitives

The Discovery primitive is of the following format:

[RV_ST, RV_F, RV_S] Discovery (Service Type, Filter, Security).

A client requests the specific service by providing its type in the *ServiceType* parameter. The attributes of the service are specified by the client in the *Filter* parameter. If the authentication of the client for using the service is required, the client provides the required information in the *Security* parameter. The Discovery primitive provides three return values. *RV_ST* is the return value containing list of the discovered services of the requested type. Each of the discovered services has a unique ID assigned by the UA. *RV_F* provides a list of attributes for each of the discovered services. For example, for a printing service, an attribute can be specified to indicate whether the printer supports colour printing or only black&white. *RV_S* is a return value indicating the authentication status.

After the successful discovery of the requested service, the client selects one of the returned services and access the service by using the Access primitive, which is of the following format:

[RV_status] Access (ServiceID, Service Method, Attributes.)

The client provides the parameter *ServiceID*, the ID of the selected service. The *ServiceMethod* parameter specifies the specific method provided by the service. The *Attributes* parameter is a set of attributes specific to the particular service. *RV_status* is the return value indicating the status of accessing the service, e.g., success or fail.

5. Universal Adaptor Mapping (UAM)

In this section, we describe UAM, which performs the mapping from UAP to the SDP specific primitives used in the target environment. As we mentioned before, there is a number of currently existing service discover mechanisms, including protocols and interoperability systems. Moreover, new mechanisms are under development. Therefore, the many-to-many approach providing a mapping between the primitives of each pair of these mechanisms would result in a heavy system with difficulties in adapting to the future service discovery mechanisms. Our approach is a one-to-many approach, providing the tailored mapping between the UAP to each native protocol used in the target environment.

For the structure of components of the UAM, please refer to Figure 3. More specifically, the functions of UAM components are described in Table 1.

Table 1. Function of UAM components

Component	Function
ESD Description	Rules specifying the characteristics of the service discovery protocol used in the particular environment, in the form of the primitives extracted from the native protocol
Translator	Module that translates UA primitives into native primitives, and outputs an algorithm for service discovery using the native primitives
ESDA	Agent that uses the algorithm generated by Translator to perform service discovery in the target environment. Depending on the native service discovery protocol, ESDA can register for service advertisements or performs active service discovery

Recall that to discover or access a service, the client uses the UA primitives described in previous section. *Translator* is a component responsible of mapping the UA primitives to the primitives of the native protocol used in the Environment. The translation process takes place each time when a client sends a query.

Translator performs mapping based on the *ESD Description* (Environment Service Discovery Description), which is set of rules specifying the characteristics of the service discovery protocol used in the particular environment. Because different environments adopted diverse models at the low level, the rules used by *ESD Description* are tailored to each environment and are described in the form of the

primitives extracted from the native service discovery protocol. *Translator* outputs an algorithm for the service discovery using the native primitives.

ESDA (Environment Service Discovery Agent) is representing the client in the process of discovery in the target environment. It uses the algorithm generated by *Translator*. From the server's point of view, *ESDA* is the same as any other client in the environment using the native protocol for service discovery. All the communications take place between the service discovery protocol and *ESDA*. Another function of *ESDA* is to provide the client with the return values from environment, e.g. information about the attributes.

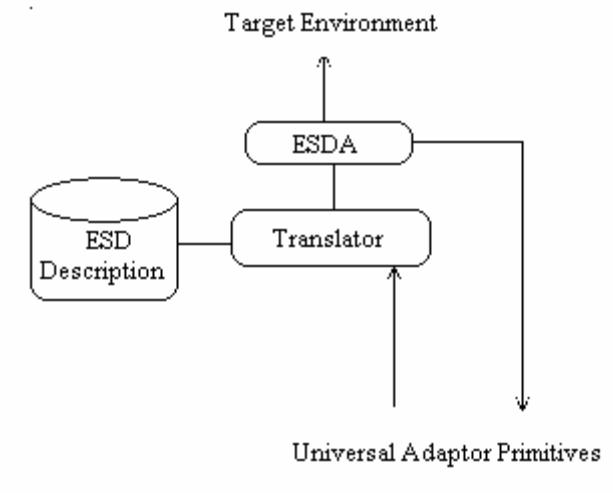

Figure 3. Universal Adaptor Mapping

6. Prototype Implementation and Experience

We have developed a prototype of our proposed UA approach. The system has two service discovery environments. One uses Jini and the other uses SLP as the native service discovery protocols. For Jini, we use Jini2_1 [20] which is an implementation of the Jini technology from Sun Microsystems. For SLP, we use a pure Java implementation of SLP – jSLP [21].

For each environment, a tailored UA is implemented. The implementation of UA for each environment consists of two parts – client side UA and service (environment) side UA, both are written in Java. Client side UA is the same for both Jini specific implementation and SLP specific implementation. On the service side, the Java UA listens on a port for the incoming requests. It parses the request, and uses reflection mechanism to invoke the corresponding service in the local environment, then passes the result back to the client.

As an example, we implemented a weather information service using Jini and SLP separately in two environments. We measured the performance of our UA system, in terms of the overhead in time comparing with the native service discovery protocols.

Experiments are conducted with a desktop computer with CPU Pentium D 2.8GHz, 1G Memory and a laptop computer with CPU Pentium Mobile 1.6GHz, 512M Memory connecting to the LAN with 100Mbps Ethernet and 11Mbps 802.11b protocol respectively.

We first measured the time for service discovery in the two environments using native service discovery clients (both for Jini and SLP). Next, we compared that with the discovery time using JiniUA and SLPUA respectively. The results of the experiments are presented in Table 2.

Table 2. Discovery overhead of UA

	Jini Environment	**SLP Environment**
Service discovery time without UA [ms]	638	152
Service discovery time with UA [ms]	743	197
Overhead [%]	16	30

In the Jini environment, the average service discovery time is 638 milliseconds, and in the case of using UA, the average time is 743 milliseconds. In the SLP environment, the average service discovery time is 152 milliseconds and the average time in the case with UA support is 197 milliseconds. The overhead that UA imposes on service discovery time is mainly introduced by the socket setup and request parse.

We also measured the time for service access in the Jini and SLP environments and compared them with the access time using UA. Results are presented in Table 3.

Table 3. Access overhead of UA

	Jini Environment	**SLP Environment**
Service access time without UA [ms]	832	223
Service access time with UA [ms]	926	268
Overhead [%]	11	20

In the Jini environment, the average service access time is 832 milliseconds, and in UA client, the average time is 926 milliseconds. In the SLP environment, the average service invoking time is 223 milliseconds, and the average time in the case with UA is 268 milliseconds. Similar to the discovery time experiments, the overhead that UA imposes service access time is related to the socket setup and request parse.

In summary, the experiment results show that the discovery and access time overhead imposed by UA is very small.

8. Conclusion

This paper presents Universal Adaptor, a novel approach towards supporting multi-protocol service discovery in pervasive computing. UA consists of Universal Adaptor Primitives and Universal Adaptor Mapping. It provides mapping from UAP to protocol specific primitives. The client makes use of UAP to discover and access services. UAM performs mapping from UAP to service specific SDP primitives. From the point of view of the service side, Universal Adaptor is a tailored component that uses native SDP and performs service discovery on behalf of the client.

Our contribution to the interoperability of service discovery is providing solution that in a lightweight manner bridges not only all existing but future service discovery systems, including standard ones, as well as service discovery mechanisms that support multiple protocols within the domain. Our implementation has shown that Universal Adaptor is a simple and flexible solution. It is easy in sense of writing the code, implementing in diverse infrastructures and using by client.

We will conduct more experiments to investigate the performance of the UA approach. In our future work, there are many issues to investigate. First, we plan to use mobile agent to find services in environments on behalf of users. Second, so far, we have assumed that the client will use UA primitives for service discovery and access, and not considered how to support clients using existing protocols. We will develop the mechanisms to provide transparent UA multi-protocol service invocation to these clients. Finally, we will investigate the possibility to let the client take the Universal Adaptor and dynamically install it to the target environment.

ACKNOWLEDGEMENT

This work is supported by Hong Kong University Research Grant Council under the CERG grant PolyU 5183/04E

9. References

1. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright, “Simple Service Discovery Protocol 1.0,” IETF, Internet-Draft Version 03, Oct. 1999.
2. E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service Location Protocol, Version 2,” IETF, RFC 2608, June 1999.
3. “Jini Technology Core Platform Specification Version1.2,” Dec. 2001.
4. “Salutation Architecture 2.1,” Salutation Consortium, Specification, 1999.
5. “Specification of the Bluetooth System,” Bluetooth Forum, Specification, Feb. 2001.
6. C. Bettstetter and C. Renner, “A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol,” In Proceedings of the EUNICE Open European Summer School, Twente, Netherlands, Sept. 2000.
7. R. Marin-Perianu, P. Hartel, and H. Scholten, "A Classification of Service Discovery Protocols". Technical Report TR-CTIT-05-25, Centre for Telematics and Information Technology, University of Twente, Enschede. ISSN 1381-3625, June 2005.

8. S. Helal, “Standards for Service Discovery and Delivery”, IEEE Pervasive Computing 1, 3 (Jul. 2002), 95-100.
9. F. Zhu, M. W. Mutka, and L. M. Ni, “Service Discovery in Pervasive Computing Environments,” IEEE Pervasive Computing 4, 4 (Oct. 2005), 81-90.
10. G. G. Richard, “Service Advertisement and Discovery: Enabling Universal Device Cooperation,” IEEE Internet Computing 4, 5 (Sep. 2000).
11. T. Koponen, and T. Virtanen, “A Service Discovery: A Service Broker Approach,” In Proceedings of the 37th Annual Hawaii international Conference on System Sciences (Hicss'04) - Track 9 - Volume 9 (January 05 - 08, 2004). HICSS. IEEE Computer Society, Washington, DC, 90284.2.
12. P. Dobrev, D. Famolari, C. Kurzke and B. A. Miller, “Device and service discovery in home networks with OSGi,” IEEE Communications Magazine 40, 8 (Aug 2002): 86-92.
13. P. Grace, G.S. Blair, and S. Samuel, “A reflective framework for discovery and interaction in heterogeneous mobile environments,” SIGMOBILE Mob. Comput. Commun. Rev. 9, 1 (Jan. 2005).
14. Y. D. Bromberg, and V. Issarny, “INDISS: Interoperable Discovery System for Networked Services,” In Proceedings of Middleware 2005. Grenoble, France November 2005.
15. P. G. Raverdy, V. Issarny, R. Chibout, and A. de La Chapelle, “A Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments,” In Proceedings of MOBIQUITOUS – The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services. July 2006, San Jose, CA, USA.
16. S. Kang, S. Ryu, N. Kim, Y. Lee, D. Lee, and K. Moon, “An Architecture for Interoperability of Service Discovery Protocols Using Dynamic Service Proxies”, LNCS 3391, pp 786-795, 2005.
17. C. A. Flores-Cortés, G.S. Blair, and P. Grace, “A multi-protocol framework for ad-hoc service discovery,” In Proceedings of the 4th international Workshop on Middleware For Pervasive and Ad-Hoc Computing (MPAC 2006) (Melbourne, Australia, November 27 - December 01, 2006). vol. 182. ACM Press, New York, NY.
18. M. Yu, A. Taleb-Bendiab, D. Reilly, and Wail Omar, "Multi-Standard Service Interoperation Protocol Through Polyarchical Middleware", In Processing of 4th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNet2003), Liverpool, U.K., June, 2003.
19. S. H. Kang, S. Ryu, N. Kim, Y. Lee, D. Lee, and K. D. Moon, “An Architecture for Interoperability of Service Discovery Protocols Using Dynamic Service Proxies,” ICOIN 2005: 786-795.
20. http://java.sun.com/developer/products/jini/installation_jini1_2_1.html
21. http://jslp.sourceforge.net/