Service discovery in a DOM-based middleware architecture

Zhaomin Xu, Ming Cai, Jinxiang Dong
Institute of Artificial Intelligence, Zhejiang University, Hangzhou, P.R. China, 310027
xzm@zju.edu.cn;cm@zju.edu.cn;djx@zju.edu.cn

Abstract

Heterogeneity and uncertainty are two main characteristics of pervasive computing environment and CSCW. CSCW can be viewed as a special case of pervasive computing. Middleware is a good approach to accommodate heterogeneous and dynamically changing devices for CSCW and pervasive computing. In this paper, we propose the service discovery method adopted in our DOM-based middleware architecture, which aims to support as many types of pervasive environments as possible. Services in our middleware architecture are dynamically structured and can be adapted to the environments' changes or users' requirements. Our middleware architecture can also be used to support service collaboration in CSCW.

Keywords: Middleware architecture, Pervasive computing, Service discovery, Service description.

1. Introduction

The essence of the vision about pervasive computing was the creation of environments saturated with computing and communication capability, yet gracefully integrated with human users. Pervasive computing represents a major evolutionary step in a line of work dating back to the mid-1970s. Two distinct earlier steps in this evolution are distributed systems and mobile computing [1].

The requirements of pervasive computing, which is also called ubiquitous computing, covers almost all the requirements of Computer-Supported Cooperative Work (CSCW), such as multi-device collaboration, modeled collaboration mode and flexible coupling, multiple-user devise, etc. A successful model for collaborative ubiquitous computing applications must combine the results of all involved research areas, including Human-Computer Interaction (HCI), Ubiquitous Computing (UbiComp), Computer-Supported Cooperative Work (CSCW) and Software development techniques [2]. Research results of mobile collaboration and service collaboration in pervasive computing are especially useful to CSCW.

Service discovery is essential for pervasive computing environments to gracefully integrate networked computing devices, and it is also important for service collaboration in CSCW. Service discovery protocols are designed to minimize administrative overhead and increase usability. They can also save pervasive system designers from trying to foresee and code all possible interactions and states among devices and programs at design time. By adding a layer of indirection, service discovery protocols simplify pervasive system design [3].

Heterogeneity and uncertainty are two main characteristics of pervasive computing environments and CSCW. CSCW can be viewed as a special case of pervasive computing. Middleware are services provided by a layer in between the operating system and the applications. It usually requires only minimal changes to existing applications and Oss [4]. Middleware is a good approach to accommodate heterogeneous and dynamically changing devices for pervasive computing and CSCW.

In this paper, we propose the service discovery method adopted in our DOM-based middleware architecture. DOM (Dynamic Object Model) is made up of several smaller patterns. The most important is Type Object, which separates an Entity from an Entity Type [5]. Services in our middleware architecture are dynamically structured and can be adapted to the environments' changes or users' requirements. Our service discovery method aims to support as many types of pervasive environments as possible. Our middleware architecture can also be used to support service collaboration in CSCW.

The rest of this paper is structured as follows: Section 2 discusses related work to this paper. Section 3 introduces our DOM-based middleware architecture. Section 4 discusses our service discovery method, including service description, service structure, service registration and discovery. Section 5 shows an example of service discovery in our middleware architecture. Finally, we make a conclusion of this paper and describe our future work.

2. Related work

1-4244-0963-2/07/$25.00 ©2007 IEEE.

Over the past few years, many organizations have designed and developed service discovery protocols. All these protocols support service discovery in ambient computing environments in terms of network topology or location. Each one addresses a different mix of issues, but most are designed for home or enterprise environments and thus don't always apply to pervasive computing beyond these confines [3].

In paper [3], a taxonomy of existing protocols has been developed as a basis for analyzing discovery approaches and identifying problems and open issues relative to service discovery in pervasive computing environments. The authors argue that service discovery protocols and the underlying computing infrastructure must have more intelligence.

Authors of paper [6] propose a new service discovery protocol, the Pervasive Discovery Protocol (PDP), which has been designed to fulfill the requirements of wireless ad hoc networks comprised of limited devices. PDP doesn't include any central server, thus requiring no infrastructure. Devices multicast their services only when there is a service request in the network. All devices within transmission range listen to these announcements and store them in a cache of known services. Devices answer a service request with all the known services of the requested type. The algorithm adopted in PDP awards fixed devices of less limited resources with more opportunities of answering requests, thus giving higher priority to answers coming from devices with longer estimated availability. This reduces the consumption of the most limited devices, taking advantage of broadcasts and multicasts in wireless networks.

VSD (Service Discovery based on Volunteers) proposed in paper [7], a service discovery architecture for multi-hop wireless ad hoc networks, exploits node heterogeneity in terms of mobility and capability. It copes with uncertainty by duplicating service information through overlapped clusters. In VSD, relatively stable and capable nodes called volunteers perform directory services in the system.

We integrate PDP and VSD into a new discovery protocol named VPDP (Pervasive Discovery Protocol based on Volunteers) in our middleware architecture to accommodate heterogeneity and uncertainty of pervasive computing environments. We aim to support as many types of networks as possible, so broadcasts and multicasts are used with caution. Volunteers in our middleware architecture are middleware nodes with less limited resources (Here 'middleware nodes' refers to network devices that have our middleware built in and can communicate with each other, such as PDA, network servers, personal computers, etc.). But each client (a service provider or a service requestor) may have different number of local volunteers and each volunteer may maintain different number of the same directory entries for different clients.

In the middleware architecture of the Gator Tech Smart House [8], a sensor platform effectively converts any sensor or actuator in the physical layer to a software service that can be programmed or composed into other services. In our DOM-based middleware architecture, devices (sensors, actuators or other devices) are abstracted as Type Objects managed by the Object Manager in the Object Layer [9], and each service corresponds to a group of devices.

3. Middleware architecture

We have proposed our middleware architecture (Figure 1) and introduced DOM (Dynamic Object Model [5]) in our previous paper [9]. There are four layers in our middleware architecture: Environment Layer, Object Layer, Service Layer and Task Layer. Environment Layer is a basic abstraction of the real world, which consists of three parts: Resource Manager, Constraint Manager and Remote Resource Manager. Object Layer is the most important layer in our middleware architecture, and DOM is implemented in this layer. Service Layer is designed to construct or reconfigure services for the Task Layer. Task Layer is designed to construct tasks, which are application units that can fulfill user's requests.

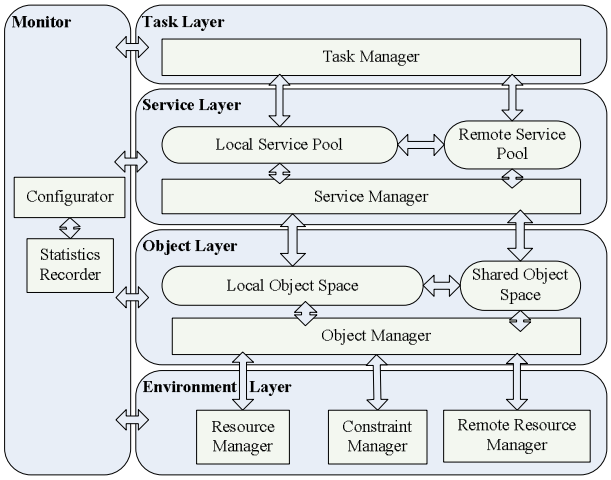

Figure 1. Middleware architecture based on Dynamic Object Model

In our middleware architecture, devices are abstracted as Type Objects managed by the Object Manager. Each service may contain more than one Type Objects and these Type Objects are considered to be in the same group. Tasks are formed by service composition. Information about devices is generated by and stored in the Resource Manager in the Environment Layer. Device information includes device properties, device functions, device requirements, etc.

We haven't realized any mechanism to fetch device information from the real world so far. So we assume that device information of each middleware node has been generated and is stored in a device information file, which is in XML format. The Resource Manager just analyzes the device information file to get device information. This assumption is reasonable because in reality device information is usually directly fetched from the operating system that our middleware lies on. In the future, the device information file may be managed by the Resource Manager, which will read device information from the underlying operating system (like the Sensor Platform [8] does), or by system administrators. System administrators can provide their own device information just by modifying the device information file if they will. We may provide device management tools to help them.

4. Service discovery

4.1. Service description

When a service requestor (SR) requests for a service, it has to describe the service it wants. Similarly, when a service provider (SP) advertises a service, it has to describe the service it has so that other nodes know how to use it. There are many approaches to describe a service. OWL, which is based on eXtensible Markup Language (XML) and Resource Description Frameworks, is a good approach that can be chosen to define an ontology to describe services [10]. But for simplicity we haven't chosen OWL. As the prototype of our middleware architecture evolves, we may use it in the near future.

There are two types of service description in our middleware architecture: Service Description for Service Requestor (SDSR) and Service Description for Service Provider (SDSP). A SDSR (Figure 2) normally contains a subset of the fields in a SDSP (Figure 3), but it can be extended to include any information needed to describe a service. Some fields are predefined and every service description must provide them. Services are grouped and subgrouped by their properties. Top level groups are divided by services' 'type' property. Function Description (Figure 4) in the SDSP is used to describe service functions. It is not very convenient for clients to use services now. They have to understand the meaning of the service input and output. We will work on to improve it. Maybe OWL will help.

Type	Quality	Availability	IO Type
Extended Description			

Figure 2. Format of Service Description for Service Requestor (SDSR)

Type	Quality	Availability	IO Type
Address	Constraints	Prerequisites	
Other Attributes-based Description			
Function Description			

Figure 3. Format of Service Description for Service Provider (SDSP)

Name	Prerequisites
Input Format Description	Input Constraints
Output Format Description	Output Constraints

Figure 4. Format of Function Description

All the fields of SDSR and most of the fields of SDSP are attributes-based descriptions. The content of every field should include the field name and the field value in two-tuple format like <'type', 'printer'>. The value of every field is based on natural language. For example, the value of 'type' can be 'printer', 'camera', 'projector', etc. We'll construct a knowledge base to group field values of the same meaning so that our middleware architecture will be more adaptable, intelligent and convenient.

4.2. Service structure

In our middleware architecture, Type Objects of the same type on a middleware node are grouped into a single service. Devices are abstracted as Type Objects which describe and implement devices' functions. This means that a service takes control of devices of the same type on the same middleware node. A service can serve different clients in the same time. It will choose which Type Object to use according to clients' requests. Some devices can execute a number of tasks in parallel, so a Type Object may be related to several contexts. Different contexts define different running environments of a service. When service migration happens, all the contexts should be carefully dealt with.

Properties provide static information of a service, contexts provide dynamic information that will change over time, and Type Objects define the service's behavior. A context is not only changed by the environment but also changed by the service itself. Figure 5 depicts the whole service structure.

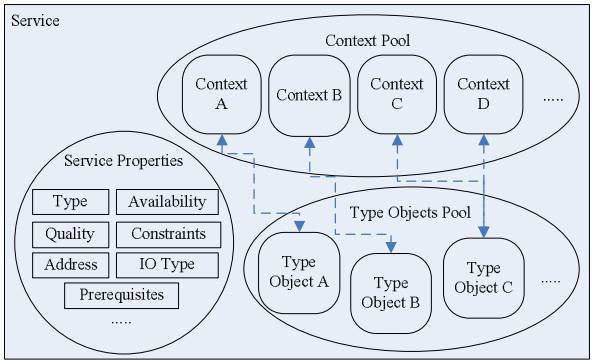

Figure 5. Service structure

Services can be composed into a task. A task may contain its own context and it must define the relationship between services. But the structure of a task is quite similar to a service. So in this paper we mainly discuss the service structure of our middleware. As stated in our previous paper [9], tasks can be generated automatically by our middleware system. But they can also be defined by system programmers. In the future, we'll provide tools for system programmers to define services and tasks.

4.3. Volunteers selection

In paper [7], volunteers in VSD are elected within a one-hop range using broadcasts. In our middleware architecture, volunteers are elected from middleware nodes within the range of a certain number of hops using multicasts. The number of hops (denoted as 'H') is defined as a system parameter which can be changed by system administrators or by the middleware system automatically. The election method of volunteers in our middleware architecture is similar to the method described in paper [7]. We have just made a few modifications to its parameters.

In our middleware architecture, when a node repeats the solicit process after it can't register with k volunteers within a given time period, it will try to register with max $\{(k/2+1), k_{min}\}$ volunteers. 'k_{min}' is a system parameter that defines the minimum number of volunteers a client (a service provider or a service requestor) should register with. It also defines the minimum number of volunteers that maintains the same directory entries for clients. Although at the start all nodes try to register with k volunteers, they may end up with different number of local volunteers that they have registered with. In other words, each client may have different number of local volunteers and each volunteer may maintain different number of directory entries for different clients.

In VSD, each node sets its own retrial times (denoted as 'ω', a integer value indicating the number of times a node should try to register with a certain number of volunteers) by considering its willingness, degree of mobility and amount of resource [7]. The lower value of ω the higher chance a node can take to be a volunteer. The opinion is consistent with that of PDP about service reply. In our middleware architecture, each node turns on or restarts with the system parameter ω set to a default value.

The value of ω will be changed over time by a evaluation function, which depends on the total running time in hours (TRTh, a value between 1 and 1000) of the node since it turns on or restarts, changed times of the network address (CTNA), and the number of services (n_s) on the node (in our middleware architecture the number of service types is equal to the number of services , so we just use the notation 'n_s' defined in paper [7]). The evaluation function is invoked periodically (for example every ten minutes) or when some events happen (such as network address changed, new services mounted on the node). TRTh and CTNA indicate the degree of mobility of a node. The number of services indicates the amount of resources on the node. The willingness of a node is hard to evaluate. We use the system parameter user willingness (UW, a value between 0 and 1, high value indicates high willingness) and n_s to evaluate a node's willingness.

Currently, the evaluation function used in our middleware architecture is defined as below. If UW equals 0, ω will be set to 9999 and the node state will be changed to 'CLIENT', which means that the node will never be a volunteer. If UW equals 1, ω will be set to 0 and the node state will be changed to 'VOLUNTEER', which means that the node will just be a volunteer. The first part of the function indicates the willingness of a node. The second part indicates the mobility of the node. The third part indicates the amount of resource on the node. Low value of these parts indicates high chance that the node can take to be a volunteer.

$$\omega = \frac{1}{UW \times (n_s + 1)} \times \frac{CTNA + 1}{TRTh / 7} \times \frac{6}{n_s + 1}$$

4.4. Service registration and discovery

We have integrated PDP and VSD into a new discovery protocol named VPDP (Pervasive Discovery Protocol based on Volunteers) in our middleware architecture. A SR is itself a user agent.

A SR can send its service request messages if it has at least k_{min} local volunteers. If doesn't, it will join a multicast group and tries to find local volunteers within H-hop range ('H' is introduced in previous section). If there are more than k_{min} responses from volunteers, the SR stores information about the first k_{min} volunteers in its local volunteers list. If it still has less than k_{min} local volunteers, it can send its service requests if there is at least one volunteer in its local volunteers list.

There are two types of queries as stated in paper [6] in our middleware architecture: one query-one response (1/1) and one query-multiple responses (1/n). In one query – one response queries, the SR selects a volunteer in its local volunteers list in a round-robin fashion and sends a service request to it. If the SR gets a service response, it can then directly interact with the SP. Otherwise if it can't get any response within a certain amount of time (denoted as 'TWspr', time to wait for SP responses), it will send the service request to another volunteer in its local volunteers list. If all the volunteers have been tried and the SR still can't get any service response, the discovery process fails.

On receiving a service request, a volunteer will lookup services in its service directory. The volunteer will send the service request to the first matched SP. If the SP accepts the request, it will send an acknowledgement to both the SP and the volunteer. The service discovery process ends successfully. Else if the SP doesn't respond within a certain amount of time (denoted as 'TWspa', time to wait for SP acknowledgement), the volunteer will try to find another matched SP in its service directory. If the volunteer can't find any matched SP, it will forward the service request to its neighbor volunteers (a volunteer knows its neighbor volunteer in service registration process). The discovery process will continue in this way until a matched SP is found or TWspr has expired or all neighbor volunteers have been tried.

In one query – multiple response queries, the service discovery process is almost the same as that described in paper [7]. So is the service registration process. When a SP registers its services, it will let the volunteer know about the other local volunteers in its local volunteers list. This registration process makes volunteers know about other volunteers and form a logical overlay network.

4.5. Service matching

Service matching is a big problem in service discovery. Designers must balance between discovery speed and accuracy. In paper [11], resources are described with hierarchies of attribute-value pairs and they are split into strands which service matching is based on. In Paper [12], service descriptions and service requests are described at an abstract level in terms of the Inputs, Outputs, Preconditions and Effects. Service matching is then based on these terms. Paper [12] also introduces three types of match: exact match, subsumption and plug-in match.

Service matching in our middleware architecture is also attribute-based (or property-based). We just compare property fields of the service request with those of the service description. When a SP receives a service request, it will also compare the predefined fields with those of its type objects (see section 4.2) to make sure that it can fulfill the SR's requirements.

5. An example

We have developed a prototype of our middleware architecture using Eclipse 3 and J2SDK 1.4.2. As mentioned in section 3, we assume that device information of each middleware node has been generated and is stored in a device information file, in XML format. The file usually contains a few device descriptions as shown in figure 6. Elements contained in each device description are used to generate a Type Object, which will be used to realize communication between services and devices.

```
<device id='Printer A'>
      <type>printer</type>
      <description>laser printer</description>
      <manufacturer>toshiba</manufacturer>
      <serial>toshiba-qkqs-48748</serial>
      <quality>wonderful</quality>
      <availability>always</availability>
      <IO-type>input</ IO-type>
      <extended-properties>none</extended-properties>

      <functions>printer_a_functions.xml</functions>
</device>
…
```

Figure 6. Device description

Values of some system parameters now defined in our prototype system are listed in the table bellow.

Table 1. Values of system parameters and default values of some system variables

Parameters	Values	Description
MULTICAST_ADDRESS	224.7.7.7	Multicast address used to join multicast group.
PORT	7777	Default port for multicast.
H	3	Hop range of volunteer election and service discovery
k	5	Number of volunteers a node should try to maintain.
k_{min}	1	Minimum number of volunteers a node should maintain.
TWspr	10000	Time to wait for SP responses, in milliseconds.
TWspa	3000	Time to wait for SP acknowledgement (millisecond).
TTLr	3	Time to live field of the request message (hop count).
ω	1	Retrial times in volunteer election.
TRTh	0	Total running time in hours of current node (1 - 1000).
CTNA	0	Changed times of the network address of current node.

n_s	0	Number of services on current node.
UW	1	User willingness (0 - 1).

Figure 7 shows information about services of a SP. Figure 8 shows the services' information that a SR has received from the service provider, with IP addresses concealed. The experiment is taken in our campus network, which is composed of many Local Area Networks.

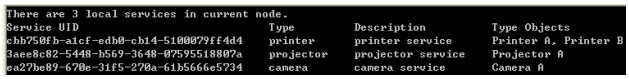

Figure 7. Services' information of a service provider

Figure 8. Services' information that a service requestor has received

6. Conclusions and future work

Middleware is a good approach to accommodate heterogeneous and dynamically changing devices for pervasive computing and CSCW. In this paper, we have discussed the service discovery method adopted in our middleware architecture. We have integrated PDP and VSD into a new service discovery protocol named VPDP to accommodate heterogeneity and uncertainty. Services in our middleware architecture are dynamically structured and include three parts: service properties, contexts and type objects. Currently, service descriptions in our middleware architecture are very simple. We'll probably use OWL in the near future.

As the prototype of our middleware architecture is under development, the example shown in this paper is rather simple. We'll keep on improving it and make experiments to prove that our middleware architecture is efficient in pervasive computing environments. In the future, we will concentrate on another hard problem in pervasive environments: service migration (or service roaming). We'll show that our middleware architecture is very adaptable to services' changes.

7. References

[1] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges". *IEEE Personal Communications*, Aug. 2001, 8(4), 10 – 17.

[2] Peter Tandler, "The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments", *Journal of Systems and Software*, January 2004, 69(3), 267-296.

[3] Feng Zhu, Matt W. Mutka and Lionel M. Ni, "Service discovery in pervasive computing environments", *IEEE Pervasive Computing*, 2005, 4(4), 81 – 90.

[4] Tim Kindberg and Armando Fox, "System software for ubiquitous computing", *IEEE Pervasive Computing*, 2002, 1(1), 70 - 81.

[5] Dirk Riehle, Michel Tilman and Ralph Johnson, "Dynamic Object Model", *In Proceedings of the 2000 Conference on Pattern Languages of Programs (PLoP 2000)*, *Washington University Technical Report number WUCS-00-29*, Washington University, 2000.

[6] Celeste Campo, Carlos García-Rubio, Andrés Marín López and Florina Almenárez, "PDP: A lightweight discovery protocol for local-scope interactions in wireless ad hoc networks", *Computer Networks*, December 2006, 50(17), 3264-3283.

[7] M. J. Kim, M. Kumar and B. A. Shirazi, "Service Discovery using Volunteer Nodes for Pervasive Environments", *In Proceedings of International Conference on Pervasive Services 2005 (ICPS '05)*, July 11-14, 2005, 188 – 197.

[8] Sumi Helal, William Mann and Hicham,El-Zabadani, Jeffrey King, Youssef Kaddoura, Erwin Jansen. "The Gator Tech Smart House: a programmable pervasive space", *Computer*, March 2005, 38(3), 50 – 60.

[9] Zhaomin Xu, Ming Cai and Jinxiang Dong, "A middleware architecture based on Dynamic Object Model for pervasive computing", *PerComChina (PCC)*, 2006, 92-97.

[10] Dipanjan Chakraborty, Anupam Joshi and Yelena Yesha, "Toward Distributed Service Discovery in Pervasive Computing Environments", *IEEE Transactions on Mobile Computing*, Feb. 2006, 5(2), 97 – 112.

[11] Magdalena Balazinska, Hari Balakrishnan and David Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery", *In Pervasive 2002 – The First International Conference on Pervasive Computing*, *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, August 2002, Vol.2414, 195-210.

[12] P. Fergus, M. Merabti, M. B. Hanneghan, A. Taleb-Bendiab and A. Mingkhwan, "A semantic framework for self-adaptive networked appliances", *Consumer Communications and Networking Conference 2005 (CCNC 2005)*, *Second IEEE*, Jan 3-6, 2005, 229 – 234.