PDP and GSDL: a new service discovery middleware to support spontaneous interactions in pervasive systems*

Celeste Campo, Mario Muñoz, José Carlos Perea, Andrés Marín, Carlos García-Rubio
Telematics Engineering Department. University Carlos III of Madrid
{celeste,munozm,jperea,amarin,cgr}@it.uc3m.es

Abstract

In pervasive computing environments, mobile devices communicate via wireless links without requiring any fixed infrastructure. These devices must be able to discover and share services dynamically. In this paper, we propose a new service discovery middleware specifically designed for this kind of environments. This middleware is composed of a service discovery protocol, Pervasive Discovery Protocol (PDP), and a service description language, Generic Service Description Language (GSDL). PDP is a fully distributed protocol that merges characteristics of both pull and push solutions; it reduces power consumption of the most limited devices. GSDL is an XML based markup language that uses a hierarchical service description designed taking into account the specific requirements of pervasive environments.

1. Introduction

Service discovery addresses the general problem in which some elements in a network, the clients, need to know the services offered by other elements in the network, the servers. This problem can be solved following a distributed approach or following a centralised approach. In the distributed approach no additional device is needed and the discovery of services can be done in two ways: *push mode*, in which servers send unsolicited advertisements, and clients listen to these advertisements selecting the services they are interested in; *pull mode*, in which clients request a service when they need it, and servers that offer the service answer the request. In the centralised approach, a new element, the directory, is introduced (be it a single physical host or a hierarchy of hosts). Servers register their services in the directory and clients send their requests to the directory. This way, the number of transmissions is minimized. The different protocols defined in the literature are based in one or more of these approaches.

*Thanks to UBISEC (IST STREP 506926) and EVERYWARE (MCyT No2003-08995-C02-01) projects.

In wired networks, there are several solutions proposed with different levels of acceptance, like IETF's SLP [8], UPnP's SSDP [7], SUN's Jini [12], UC Berkeley's SSDS [4] and Salutation [3]. However the heterogeneity and volatility of ubiquitous environments demand a novel approach to the problem of service discovery in the software of ubiquitous computing systems.

The paper is organized as follows. First, in section 2, we describe the new challenges for service discovery and description in pervasive environments. Then, in section 3, we introduce our proposals, the PDP and GSDL. In section 4 we present an implementation of our middleware using J2ME. Section 5 reviews the related work, and finally we draw some conclusions and future work in section 6.

2. Challenges for service discovery in pervasive environments

Pervasive computing environments impose new restrictions that must be taken into account when defining a middleware for service discovery. These restrictions can be divided in those that concern to the way services are discovered (i.e. the service discovery protocol) and those that concern to the way services are defined (i.e. the service description). Here, we will present both.

2.1. Challenges for service discovery protocol

There is a big consensus that legacy solutions for service discovery fail in addressing the requirements imposed for pervasive environments [9]. Here, we summarize some of the most outstanding challenges that must be addressed.

- Minimize network transmissions: One of the main sources of power consumption is network transmission [6]. This implies that one of the most important issues when designing a service discovery protocol is to minimize the number of transmissions, especially of the most limited devices.

- Do not rely on fixed infrastructure and adapt to highly changing environments: In pervasive computing networks, most or all devices are mobile. No fixed infrastructure, including a directory of services, should be required. However, in certain places, but not always, some devices will be fixed and will stay forever. A service discovery protocol must adapt well to all places.
- Maximize cooperation between devices: In environments saturated with limited devices, cooperation allows them to carry out more complex tasks. This concept, applied to mobile services discovery, means that devices within the same system must cooperate to discover services, for the sake of the common good.
- Take into account different application needs: Some applications look for any device offering a service in the network, e.g. any temperature sensor in the room, while others browse all the devices offering that service in the network, e.g. a word processor that searches for printers and wants to display all the available ones. Traditionally, service discovery protocols have not differentiated between both kinds of searches.

2.2. Challenges for service description

The proliferation of services in ubiquitous computing environments requires a scalable service description mechanism complex enough to capture every detail in every service. Here we summarize the challenges for service description in these environments.

- Simplicity: Many of the devices are limited devices in terms of processing power, memory and networking capabilities. Simplicity is therefore a need.
- Scalability: A service description mechanism adapted to pervasive environments should be able to capture every detail in every service that could be present in the environment. There is a certain trade off between simplicity and scalability which has to be taken into account.
- Backward and forward compatibility: It is important that new services are available not only for new applications but for legacy applications also. New applications, on the other hand, should be able to use legacy services.

3. Definition of a new service discovery middleware

In this section we present the definition and implementation of a new service discovery middleware which consists of a service discovery protocol, the Pervasive Discovery Protocol (PDP), and a service description language, the Generic Service Description Language (GSDL), especially designed to work in ubiquitous environments.

3.1. Definition of a new service discovery protocol

One of the key objectives of the PDP protocol is to minimize the number of transmissions necessary to discover services, and so the battery consumption, especially of the most mobile and limited devices. This is accomplished through the use of the availability time, a parameter that will be introduced bellow. PDP prioritizes the replies of the less limited devices, allowing the others to abort their answers. PDP also does away with the need for the central server, and it is a fully distributed protocol that merges characteristics of both pull and push solutions. Devices maintain a cache of services previously announced, that is also used for the answers. In PDP all messages are broadcast, and all devices cooperate by coordinating their replies and sharing the information in their caches. PDP takes into account the different applications needs, and this allows to further reducing the power consumption. In this section, we explain in more detail how PDP works.

Application scenario Let's assume that there is a ubiquitous environment, composed of D devices, each device offering S services and expecting to remain available in this network for T seconds. This time T is called the availability time and it is configured in the device, depending on its mobility characteristics.

Each device has a PDP User Agent (PDP_UA) and a PDP Service Agent (PDP_SA). The PDP_UA is a process working on behalf of the user to search information about services offered in the network. The Service Agent PDP (PDP_SA) is a process working to advertise the services offered by the device. The PDP_SA always includes the availability time T of the device in its announcements.

Each device has a cache containing a list of the services that have been heard from the network. Each element of the cache has two fields: the service description and the service expiration time. The service expiration time is the estimated time for the service to remain available. This time is calculated as the minimum of two values: the availability time of the local device, and the service announced availability time. Entries are removed from the cache when they timeout.

Protocol description PDP has two mandatory messages: `PDP_Service_Request`, which is used to send service requests, and `PDP_Service_Reply`, which is used to answer a `PDP_Service_Request`, announcing available services. Additionally, PDP has one optional message: `PDP_Service_Deregister`, which is used to inform that a service is no longer available.

When an application or the final user of the device needs a service of a certain type, it calls its PDP_UA. In order to support the needs of different applications, in PDP we have defined two kinds of queries:

- One query-one response (1/1): the application is interested in the service, not in which device offers it.
- One query-multiple responses (1/n): the application wants to discover all devices in the network offering the service. In this kind of query, we introduce a special type of service, named ALL, in order to allow an application to discover all available services of all types in the network.

Both types of query use the same message, `PDP_Service_Request`. A flag in the header of the message indicates if it is 1/1 or 1/n.

In one query-one response queries, the PDP_UA searches for a `service_type` in the list of local services and in its cache. If it is found, the PDP_UA gives the application the corresponding service description, without any network transmission. If it is not found, the PDP_UA broadcasts a `PDP_Service_Request` for that service, waiting `CONFIG_WAIT_RPLY` seconds for replies. If no reply arrives, the PDP_UA answers to the application that the service is not available in the network. If some reply arrives, the PDP_UA gives the application the service description received.

In one query-multiple responses queries, the PDP_UA makes a list of known services of the specified type, that is, a list of the ones locally offered or stored in its cache (all the services if the service type is ALL). Then, it sends a `PDP_Service_Request` including this list. It waits `CONFIG_WAIT_RPLY` seconds for replies and when this timer expires, it gives the application the list of known services plus, if any replies arrived, the service descriptions received.

PDP_UAs in all devices are continually listening to the network for all types of messages (requests and replies) and update their caches with the services announced in them. Moreover, the device's cache has a limited size. When a PDP_UA hears a new announcement but the cache is full, it deletes the service entry closer to expire.

The PDP_SA advertises services offered by the device. It has to process `PDP_Service_Request` messages and to generate the corresponding `PDP_Service_Reply`, if necessary.

In order to minimize the number of transmissions, the PDP_SA takes into account the type of query made by the remote PDP_UA. When a PDP_SA receives a `PDP_Service_Request 1/1`, it checks whether the requested service is one of its local services. In that case, a `PDP_Service_Reply` is scheduled for a random time, inversely proportional to the availability time of the device. During this time, if another reply to the same PDP request is heard, the reply is aborted as the remote PDP_UA will just pass the first service to the application and discard any other. If the timer expires and no reply has been heard, the reply is sent.

The algorithm awards the more static devices with more opportunities of answering requests. Therefore the algorithm gives higher priority to answers coming from devices with longer estimated availability.

When a PDP_SA receives a `PDP_Service_Request 1/n`, it checks whether the requested service is one of its local services, or if it is in the cache. If so, it generates a random waiting time, inversely proportional to the availability time of the device and the number of known services. During this time, the PDP_SA listens to the network for any `PDP_Service_Reply` of the same request. When the timer expires, if the PDP_SA knows about some additional devices offering this type of service that have not been announced yet, it sends its `PDP_Service_Reply`. So, the more time the device is able to offer the service and the bigger the cache, the higher the probability of answering first. We suppose the device with the highest availability time and the biggest cache is the one with the most accurate view of the world.

In certain cases, a device may detect when it is about to be switched off or to roam to other network. If so, the PDP_SA of the device has to send a `PDP_Service_Deregister`, listing all its local services, before switching off or roaming. When another device hears this message, it must remove the services listed from its cache. In other cases, when a device tries to access a service listed in its cache and the service is down, it may also use the `PDP_Service_Deregister` message to inform the rest of the network that this service is no longer available. The device that receives the message may delete the entry from the cache.

3.2. Definition of a new service description language

In this section we present a new service description language we have defined for pervasive computing environments that satisfies the requirements captured in the previous section. This language is called Generic Service Description Language (GSDL).

GSDL makes use of the service hierarchy described in [13]. This hierarchy defines general services common to all the environments in next generation networks in a first layer, specifies a second layer with more specific services that are shared by groups or families of devices and finally incorporates, in a third and successive layers, the specific services for each device (in fact, layers 1 and 2 are abstract layers whose functions are implemented in services in layers 3 and beyond). The services exported by each device are inserted in an inheritance tree in which the specific details in layer 3 are descendant from more generic services in layer 2 and so on. This allows applications in pervasive environments to use the services in other devices even if they are not familiar or have no previous knowledge of the details in layer 3 by means of a well known ancestor in layers 1 or 2. This service hierarchy is a good solution for both the simplicity and scalability requirements stated in the previous section since it allows, on the one hand, the definition of specific services and, on the other, their generic use by limited devices through well known services in layers 1 and 2. Moreover, this hierarchical service definition provides forward and backward compatibility of services and applications as long as layer 1 and 2 services are not modified.

It is important that the service description language is well integrated with the service access mechanisms. Pervasive computing environments are open and network oriented by nature. Several access protocols are defined for these open communication environments such as CORBA, RMI and Web Services. We have selected Web Services as the service access mechanism since it is well adapted to ubiquitous computing. Web Services are XML based and use the ubiquitous Internet protocols such as HTTP. Web Services are simple enough to be integrated in small devices. GSDL is defined to be a service description language which is well integrated with Web Services. In fact, GSDL is defined to be a complementary description to the service interface. Web Services use WSDL as the language to specify the interface for service access. Although WSDL is powerful enough to capture every aspect related to the service interface it does not specify any relationship between different services. This is the reason why we have designed GSDL since the definition of a hierarchical relationship among services provides a simple, scalable and forward and backward compatible way to define new services. GSDL captures only the hierarchical relationships for the described service and uses WSDL as the language for interface description.

GSDL is XML based and is specified using an XML Schema. The first elements in the Schema are graphically represented in Figure 1. For every specific service, GSDL describes the level 1 and level 2 services from which this specific service inherits. These level 1 and level 2 services are well known services and represent a generic way to access every specific service (in fact they are defined taking into account the different network scenarios in next generation networks, from which pervasive computing environments are an important part). The specific details of level 3 services are captured specifying the URL where the interface description in WSDL can be obtained.

In order to reduce the size of the PDP messages, we do not include the full GSDL description of the services. Instead of this, for a specific service, we include the layer 1 and layer 2 services codified in two byte (there are three services of level 1 and seven services of level 2) and the URL where the device can obtain the level 3 service descriptions.

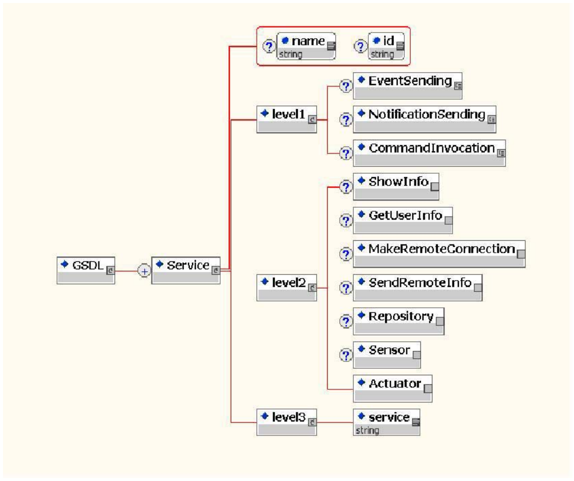

Figure 1. GSDL Schema graphical representation (generated with Turbo XML from Tibco)

4. Implementations issues

The middleware is implemented in J2ME Personal Profile. This library is composed of two packages (one implementing PDP and the other implementing GSDL) with a total file size of 39KB, and needs three external libraries to run. These external libraries (58KB), kSoap.jar, kObjects.jar and kXML.jar, have to do with SOAP and XML parsing mechanisms suitable for J2ME platform, developed under the Enhydra kXML-RPC projects.

5. Related work

In this section, we will present the main proposals that have appeared during the last years, and that, as our middleware, focus on providing a generic solution for service discovery in pervasive computing environments. We also discuss the main differences between them and our proposal.

DEAPspace Algorithm [14] proposes a push solution, in which all the devices hold a list of all known services, the so called ”world view”. Each device periodically broadcasts its ”world view” to its neighbors, which update their ”world view” accordingly. DEAPspace confines itself to a small network by assuming a single hop ad hoc network and uses broadcasts to send the messages. Its primary goal is to obtain a fast convergence of service information available in the network. DEAPspace uses service attributes as the format for service description.

Rendezvous [1] enables automatic discovery of computers, devices, and services on IP networks. Service discovery and description in Rendezvous are based on clasical DNS and in a distributed DNS, called Multicast DNS, for ad-hoc networks, consisting on a pull mode service discovery with multicast responses. Another proposal of distributed DNS, LLMNR [5], is under development within the DNSEXT group of the IETF.

Konark [10] is a middleware designed specifically for the discovery and delivery of services in multi-hop ad hoc networks. With regard to the service discovery mechanism, Konark supports both push and pull modes, with a cache in all devices. Konark has recently defined a new discovery mechanism, called Konark service gossip protocol [11]. This is based on a message exchange round, which is triggered by a service request message or by a service announcement message. With regard to service description, Konark defines an XML based service description language similar to WSDL.

Regarding how services are discovered, when compared with PDP, these proposals have the following lacks: (i) they do not take into account the different characteristics of the devices (battery, memory size), they handle all of them the same way; (ii) except Rendezvous, they are based in a push mode, announcements are transmitted even when no other device is in the network. This is a waste of battery in many situations; (iii) pull mode is supported, but it is not well integrated with the push mode. Applications must choose between consulting the cache or issuing a service request (pull); (iv) they do not take into account different application needs.

With regard to how services are described, when compared with GSDL, the description mechanisms used by these proposals have the following lacks: (i) they are not scalable enough, every service is defined independently of other related services; (ii) the service description based only on service URLs or service attributes requires also the specification of a service access mechanism. In open environments such as pervasive computing networks the service access mechanism should be very well integrated with the service description language.

6. Conclusions and future work

A good service discovery mechanism for ubiquitous environments must address the highly changing nature of these environments, no depending on any existing infrastructure. It must consider the power constraints of most of the devices. It must take into account that interactions with close devices will be likely and interactions with far devices will be unusual. It must exploit cooperation between devices to achieve the best performance with the least effort. Finally, it must address the different needs of applications, with some looking for any instance of a particular service and others looking for all instances of that service. PDP has been designed taking into account these needs.

The service description mechanism used in pervasive environments should provide a simple and yet scalable way to describe whatever services conceivable in this kind of environments. Moreover it should be very well integrated with the service access mechanism in a forward and backward compatible way. GSDL fulfils these requirements basing its service description on a hierarchical categorization of pervasive services which is integrated with Web Services for service access.

We are aware of the security issues of ubiquitous computing environments and we are working in a distributed trust model for them [2].

References

[1] http://developer.apple.com/macosx/rendezvous/.

[2] F. Almenárez, A. Marín, C. Campo, and C. García-Rubio. PTM: A Pervasive Trust Management Model for Dynamic Open Environments. In *PSPT 2004*, Boston, MA, USA, Aug. 2004.

[3] S. Consortium. Salutation Architecture: Overview, 1998.

[4] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Proc. Mobicom'99*, 1999.

[5] L. Esibov, B. Adoba, and D. Thaler. Linklocal Multicast Name Resolution (LLMNR). Internet-Draft (work in progress), Oct. 2004.

[6] L. M. Feeney and M. Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *IEEE INFOCOM*, 2001.

[7] Y. Y. Goland, T. Cai, P. Leach, and Y. Gu. Simple Service Discovery Protocol/1.0. Internet-Draft (work in progress), Apr. 1999. draft-cai-ssdp-v1-03.txt.

[8] E. Guttman, C. Perkins, J. Veizades, and M. Day. RFC 2608: Service Location Protocol, Version 2, June 1999.

[9] S. Helal. Standards for Service Discovery and Delivery. *IEEE Pervasive Computing*, pages 95–100, jul/sep 2002.

[10] S. Helal, N. Desai, and V. Verma. Konark-A Service Discovery and Delivery Protocol for Ad-hoc Networks. In *IEEE WCNC)*, New Orleans, Mar. 2003.

[11] S. Helal, N. Desai, V. Verma, and B. Arslan. Konark: A System and Protocols for Device Independent, Peer-to-Peer Discovery and Delivery of Mobile Services. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(6):682–696, Nov. 2003.

[12] S. Microsystems. Jini Architectural Overview., 1999.

[13] M. Muñoz and C. García-Rubio. A New Model for Service and Application Convergence In B3G/4G Networks. *IEEE Wireless Communications*, 11(5):6–12, Oct. 2004.

[14] M. Nidd. Service Discovery in DEAPspace. *IEEE Personal Communications*, Aug. 2001.