Distributed Directory Facilitator:
A proposal for the FIPA Ad-hoc First CFT

Celeste Campo
Universidad Carlos III de Madrid
`celeste@it.uc3m.es`

April 30, 2002

Contents

Part I
General data

1 Contact point

Name:	Celeste Campo
Postal address:	Universidad Carlos III de Madrid
	Avda. Universidad 30
	28911 Leganés (Madrid), Spain
Affiliation:	Depto. de Ingeniería Telemática
	Universidad Carlos III de Madrid (Spain)
Email address:	`celeste@it.uc3m.es`
Telephone number:	+34-91-624-5949
Fax number:	+34-91-624-8749

2 Completeness of the submission

This submission to the FIPA TC Ad-hoc First Call For Technology addresses in part the second requirement stated in section 2 of the CFT: "definition of mechanisms and protocols for agent platform fragments to build, release, join and leave compounds". Specifically, our proposal centers on the fragmentation of the Directory Facilitator (DF).

As suggested in section 6 of the CFT, a device without DF functionality must use a remote DF to advertise its own services and to find out other services provided by other agents on other devices in its surroundings.

The question is: how does the device without DF functionality know the address of the remote DF? Two solutions are possible:

- The address of the remote DF is statically (manually) configured in the device.

- The address is dynamically "discovered" by the device, for example with broadcast messages announcing or requesting a DF.

However, we think pervasive environments cannot be relied upon to have any single device permanently present in order to act as remote DF, because they are dynamic in nature. The second solution described above is better because the device dynamically discovers the nearest remote DF, but it has many problems. First, if the environment changes frequently, the device will be continually discovering a new remote DF. Second, maybe none of the devices present at any moment may be suitable to act as the remote DF.

Here we propose all the devices to have a DF fragment that works in a distributed fashion and dynamically discovers services in its environment. This is what we call a *Distributed DF fragment* and its cost is equivalent at the one of the second solution described above when the environment is very changing.

We have considered existing approaches for dynamic service discovery, such as SLP, Jini, Salutation and UPnP's SSDP. None of them adapts well to the characteristics of ad-hoc networks. So, we have defined a new service discovery protocol, called Pervasive Discovery Protocol (PDP), to be used in the Distributed DF.

In part II of the proposal, we present the concept of Distributed DF, review existing service discovery protocols and describe the PDP algorithm.

3 Overview

Recent advances in microelectronic and wireless tecnologies have fostered the proliferation of small devices with limited communication and processing power. They are what are known as "pervasive systems".

Personal Digital Assistants (PDAs) and mobile phones are the more "visible" of these kinds of devices, but there are many others that surround us, unobserved. For example, today most household appliances have embedded microprocessors. Each one of these small devices offers a specific service to the user, but thanks to their capacity for communication, in the near future they will be able to collaborate with each other to build up more complex services. In order to achieve this, devices in such "ad-hoc" networks should dynamically discover and share services between them when they are close enough. For example, sensors and air conditioning systems in intelligent buildings will talk with our PDAs to automatically adapt the environment to our needs or preferences. As Arthur C. Clarke has said – any sufficiently advanced technology is indistinguishable from magic. We are approaching the time when "open sesame" is a valid user command.

We think that agent technology will be of great help in pervasive systems development. Pervasive systems are inherently dynamic, with devices continually coming in and out. Agents are autonomous software entities that can interact with their environment, and therefore they adapt well to such continuous changes.

However, the use of Multi-Agent Systems (MAS) in pervasive environments poses important challenges, and it is necessary to adapt their design to meet these challenges. We are currently working on this.

In part II of this report, we will define the Distributed Directory Facilitator, that helps agents in search of services, offered by other agents or other systems in the ad-hoc network. Then, we will see that none of the service discovery protocols propose so far in the literature, adapts well to the case of pervasive systems. Finally, we will propose a new service discovery protocol, called PDP.

Part II
Proposed specification

4 Distributed Directory Facilitator

To implement the Directory Facilitator in a pervasive environment, we propose to use a Distributed DF formed by the joint work of Distributed DF fragments in devices that coincide in a given moment in the same ad-hoc network, with no need of any remote "full" DF. The fragments that form the Distributed DF in an ad-hoc network change as devices join and leave the network.

The Distributed DF fragment helps agents working on the device in search of services offered by other devices in the ad-hoc network. It is a kind of "yellow pages" service adapted to the peculiarities of pervasive computing enviroments. One can think of it as a kind of shop assistant. New entrants to the shop will want assistance in order to better locate items of interest, and the agent, because of its relative expertise in the environment, will generally be able to provide the appropriate directions.

To carry out its work, a Distributed DF fragment needs to implement a service discovery protocol. As we will see in the next section, service discovery protocols proposed for use in the Internet do not work well in pervasive systems in ad-hoc networks. We have defined a new protocol, the Pervasive Discovery Protocol, PDP (see Section 4), specially designed to work in this environment. The internal operation of the Distributed DF fragment will be based on this protocol.

It is important to point out here that, since the Distributed DF fragment uses the PDP, it will be able to interoperate with any other system that implements PDP, be it an agent platform or not (e.g., sensors, air-conditioning system, lighting control, etc.).

5 Service Discovery

Dynamic service discovering is not a new problem. There are several solutions proposed for fixed networks, with different levels of acceptance. We well now briefley review some of them: SLP, Jini, Salutation and UPnP's SSDP.

The Service Location Protocol (SLP) [2] is an Internet Engineering Task Force standard for enabling IP networks-based applications to automatically discover the location of a required service. The SLP defines three “agents”: User Agents (UA), that perform service discovery on behalf of client software, Service Agents (SA), that advertise the location and attributes on behalf of services, and Directory Agents (DA), that store information about the services announced in the network. SLP has two different modes of operation: when a DA is present, it collects all service information advertised by SAs, and UAs unicast their requests to the DA, and when there is not a DA, UAs repeatedly multicast the request they would have unicast to a DA. SAs listen for these multicast requests and unicast responses to the UA.

Jini [3] is a technology developed by Sun Microsystems. Its goal is to enable truly distributed computing by representing hardware and software as Java objects that can form themselves into communities, allowing objects to access services on a network in a flexible way. The service discovery in Jini is based on a directory service, similar to the Directory Agent in SLP, named the Jini Lookup Service (JLS). JLS is necessary to the functioning of Jini, and clients should always discover services using it, and never can do so directly.

Salutation [5] is an architecture for looking up, discovering, and accesing services and information. Its goal is to solve the problems of service discovery and utilization among a broad set of applications and equipment in an environment of widespread connectivity and mobility. The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a directory of applications, services and devices, generically called Networked Entities. The SLM allows networked entities to discover and to use the capabilities of the other networked entities.

Simple Service Discovery Protocol (SSDP) [1] was created as a ligthweight discovery protocol for Universal Plug-and-Play (UPnP) initiative, and defines a minimal protocol for multicast-based discovery. SSDP can work with or without its central directory sevice, called the Service Directory. When a service wants to join the network, first it sends an announcement message to notify its presence to the rest of the devices. This announcement may be sent by multicast, so all other devices will see it, and the Service Directory, if present, will record the announcement. Alternatively, the announcement may be sent by unicast directly to the Service Directory. When a client wants to discover a service, it may ask the Service Directory for it or it may send a multicast message asking for it.

The solutions described above can not be directly applied to the scenario we treat in this report, because they were designed for and are more suitable for (fixed) wired networks. We see two main problems in the solutions enumerated:

- First, many of them use a central server, that maintains the directory of services in the network. Pervasive environments cannot be relied upon to have any single device permanently present in order to act as central server, and furthermore, maybe none of the devices present at any moment may be suitable to act as the server.

- Second, the solutions that may work without a central server, are designed without considering the power constraints typical in wireless networks. They make an extensive use of multicast or broadcast transmissions almost costless in wired networks but are power hungry in wireless networks.

Accepting that alternatives to the centralized approach are required, we consider two alternative approaches to distributing service announcements:

- The "Push" solution, in which a device that offers a service sends unsolicited advertisements, and the other devices listen to these advertisements selecting those services they are interested on.

- The "Pull" solution, in which a device requests a service when it needs it, and devices that offer that service answers the request, perhaps with third devices taking note of the reply for future use.

In pervasive computing, it is very important to minimize the total number of transmissions, in order to reduce battery consume. It is also important to implement mechanisms to detect as soon as possible both the availability and unavailability of services produced when a device joins or leaves the network. These factors must be taken into account when selecting between a push solution or a pull solution.

The DEAPspace group of the IBM Research Zurich Lab. has proposed a solution to the problem of service discovering in pervasive systems without using a central server. The DEAPspace Algorithm [4] is a pure push solution, in which all devices hold a list of all known services, the so called "world view". Each device periodically broadcasts its "world view" to its neighbours, which update their "world view" accordingly.

We think the DEAPspace algorithm has the following problem: the "world view" of a device spreads from neighbour to neighbour, perhaps arriving to a device where some of those services are in fact not available.

In this report we propose a new service discovery algorithm, the Pervasive Discovery Protocol (PDP), wich merges characteristics of both pull and push solutions. This way we think a better performance can be achived.

6 Pervasive Discovery Protocol

The Pervasive Discovery Protocol (PDP) is intended to solve the problem of enumerating the services available in a local cell in a low power short-range wireless network, composed of devices with limited transmision power, memory, processing power, etc. The classical service discovery protocols use a centralized server that listens for broadcast or multicast announcements of available services at a known port address, and lists the relevant services in response to enquiries. The protocol we propose does away with the need for the central server. The kind of enviroments described above cannot be relied upon to have any single device permanently present in order to act us central server, and further, none of the devices present at any moment may be suitable to act as the server.

One of the key objectives of the PDP is to minimize battery use in all devices. This means that the number of transmisions necessary to discover services should be reduced as much as possible. A device announces its services only when other devices request the service. Service announcements are broadcast to all the devices in the network, all of which will get to know about the new service simultaneously at that moment, without having to actively query for it.

In the remainder of this section, we present the application scenario for PDP and some considerations to be taken into account, and then we will formally describe the algorithm used to implement it.

6.1 Application scenario

We will suppose that there are D devices, each one with I network interfaces. Each device offers S services and expects to remain available for T seconds. This time T is previously configured in the device, depending on its mobility characteristics. The Distributed DF fragment has a cache associated with each interface which contains a list of the services that have been heard from on this interface. Each element e of this list has two fields: the service description, *e.description*, and a time that it is calculated that the service will be available for, *e.timeout*. The calculation is exactly the minimum of two values: the time that the local device has promised to remain available, T, and the time that the server that offers the service announced that it would be available for.

Services are not associated with any specific interface and the availability time T of the device is always included in the announcements of its services.

For simplicity, we suppose that local services are stored in the cache associated with the loopback interface ($cache_0$).

6.2 Algorithm description

The PDP has two messages: **PDP request**, which is used to send service annoucements and **PDP reply**, which is used to answer a PDP request, announcing availables services.

Now, we will explain in detail how a Distributed DF fragment uses these primitives.

6.2.1 PDP request

When an application or the final user of the device needs a service, whether a specific service or any service offered by the enviroment, it requests the service from the Distributed DF fragment. The number of broadcast transmissions should be minimized, so:

- If a specific service S has been requested, the Distributed DF fragment searches for that service in all its caches. If it is not found, it broadcasts a PDP request for that service (table 1).
- If the request is for all available services in the network, the Distributed DF fragment updates its caches by sending a PDP request message through all the interfaces of the device (table 2).

6.2.2 PDP reply

The Distributed DF fragments in all devices are continually listening on each interface for all type of messages (PDP requests and PDP replies).

Table 1: PDP request S

```
for ( $i = 0$  to  $I$  ) {
  if ($S \in cache_i$) return $i$;
}
for ($i = 1$ to $I$ ) {
  remote_service = PDP request($S$);
  if ($\exists$ remote_service) {
    add_service(remote_service,  $cache_i$);
    return $i$;
   }
}
```

Table 2: PDP request ALL

```
for ($i = 1$ to $I$ ) {
  remote_services_list= PDP request($ALL$);
}
```

Table 3: PDP reply S

```
if ( $\exists\ e \in cache_0\ /\ S = e.description$) {
  t = generate_random_time($\frac{1}{T}$);
  wait(t);
  if ( not listened PDP reply)
    PDP reply(e);
}
```

When a PDP reply is received, announcing a service, the Distributed DF fragments update their caches accordingly.

When a PDP request for a specific service S (table 3) is received then the Distributed DF fragment:

- Checks whether the requested service, S, is one of its local services and therefore is stored in the loopback cache, or is not.
- If not, it generates a random time t, inversely proportional to the availability time of the device, T. So, the more time the device is able to offer the service, the higher the probability of the device answering first.
- During the interval t, the Distributed DF fragment listens to the network. If another reply to this PDP request arrives, it aborts its PDP reply, otherwise when the interval expires, it sends its PDP reply.

When a PDP request for all services ALL (table 4) is received then the Distributed DF fragment:

- Generates a random time t, inversely proportional to the availability time of the device, T, and to the number of elements stored in the cache of that interface. So, the more time the device is able to offer the service and the biggest the cache, the higher the probability of answering first. We suppose the device with the highest availability time and the bigger cache is the one with the most accurate view of the world.
- During the interval t, the Distributed DF fragment listens to the network. If another reply to this PDP request arrives, it aborts its PDP

Table 4: PDP reply *ALL*

```
t = generate_random_time( $\frac{1}{T*cache\_size}$ );
wait(t);
if ( not listened PDP reply)
  PDP reply($cache_0, cache_i$);
```

reply, otherwise when the interval expires, it sends its PDP reply listing the services in the loopback cache plus the services in the cache of that interface.

7 Additional Authors

Andrés Marín (amarin@it.uc3m.es), Carlos García-Rubio (cgr@it.uc3m.es), and Peter T. Breuer (ptb@it.uc3m.es).

References

[1] Y. Y. Goland, T. Cai, P. Leach, and Y. Gu. Simple service discovery protocol/1.0. Technical report, 1999.

[2] IETF Network Working Group. *Service Location Protocol*, 1997.

[3] S. Microsystems. Jini architectural overview. white paper. Technical report, 1999.

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

[5] I. Salutation Consortium. Salutation architecture overview. Technical report, 1998.