Directory Facilitator and Service Discovery Agent

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

July 19, 2002

Contents

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

Scope of the document

This document is the contribution of the University Carlos III of Madrid to the white-paper "Agents in Ad Hoc Environments" to be discussed at FIPA26 in Helsinki.

In this document we broach the problem of implementing yellow pages services in agent platforms for ad-hoc networks. We present a solution integrated with FIPA specifications, centred in the adaption of the Directory Facilitator to work in ad-hoc environments.

First, we will start with an introduction to the characteristics and requirements of pervasive systems and ad-hoc networks. Secondly, and following the proposed index for the mentioned white-paper, we briefly review some existing discovery technologies and their limitations in ad-hoc networks; we also propose a new service discovery protocol, the Pervasive Discovery Protocol (PDP). Thirdly, we propose a solution for service discovery based on existing service discovery mechanisms. Finally, we propose another solution at an agent level that uses multicast ACL messages improved with the use of caches.

1 Introduction

Recent advances in microelectronic and wireless technologies 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.

The most accepted definition of ad-hoc network is the following: it is a network that consists of mobile nodes that use wireless interfaces to send and receive data, with no central, permanent network infrastructure.

The dynamic topology of ad-hoc networks, the fluctuating quality and capacity of wireless links, and the limited capacity (memory, processing power, and battery life) of many of the devices connected to this kind of networks, pose important, new challenges that should be addressed when defining new proposals for this environment.

Next, we will enumerate some of the requirements of ad-hoc networks, different of those of legacy fixed or mobile networks:

- Distributed operation: In ad-hoc environments, nodes join and leave the network spontaneously, so network operation must be distributed between all the devices that form the network at any given time. This is one of the requirements that more differ from traditional networks. In traditional networks, some elements perform special tasks without which the global operation of the network is not possible. For example, in GSM, network operation rely on the existence of base stations. Terminal equipments connect to these base stations to obtain connectivity and to access services.

- High cost of the communications: Communications in ad-hoc networks

are expensive because they imply high battery consume in devices typically with limited battery power.

- Intransitive connectivity: In fixed networks, if A has connectivity with B, and B has connectivity with C, then A has connectivity with C. In ad-hoc networks this may not be true due to the short-range coverage of the underlying wireless protocols.

- Very changing environments: Nodes in ad-hoc networks continually change (join and leave the network).

- Broadcast domains: In ad-hoc networks, you can use broadcasts to reach all nodes in your range, that is, all the nodes you are able to communicate with. In fixed networks, broadcast domains are constrained to closer nodes (those within your local area network), and so broadcasts do not reach all nodes you are able to communicate with.

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. One of them is the implementation of yellow pages services in agent platforms for ad-hoc networks. Here we will present a solution integrated with FIPA specifications, centred in the adaption of the Directory Facilitator to work in ad-hoc environments.

2 Discovery technologies

2.1 Review of existing discovery technologies

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

2.1.1 Service Location Protocol (SLP)

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.

2.1.2 Jini

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.

2.1.3 Salutation

Salutation [5] is an architecture for looking up, discovering, and accessing services and information. Its goal is to solve the problems of service discovery and utilisation 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.

2.1.4 Simple Service Discovery Protocol (SSDP)

Simple Service Discovery Protocol (SSDP) [1] was created as a lightweight 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 service, 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.

2.2 Problems of existing discovery technologies in ad-hoc networks

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 centralised 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 minimise 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), which merges characteristics of both pull and push solutions. This way we think a better performance can be achieved.

2.3 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 transmission power, memory, processing power, etc. The classical service discovery protocols use a centralised 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 environments 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 minimise battery use in all devices. This means that the number of transmissions 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.

2.3.1 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 ($cache_0$).

2.3.2 Algorithm description

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

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

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 environment, it requests the service from the Distributed DF fragment. The number of broadcast transmissions should be minimised, 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).

Table 1: PDP request S

```
for ( i = 0  to  I  ) {
  if (S ∈ cache_i) return i;
}
for (i = 1 to I ) {
  remote_service = PDP request(S);
  if (∃ 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);
}
```

- 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).

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).

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 avail-

Table 3: PDP reply S

```
if ( ∃ e ∈ cache_0 / S = e.description) {
  t = generate_random_time(1/T);
  wait(t);
  if ( not listened PDP reply)
    PDP reply(e);
}
```

ability 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 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.

Table 4: PDP reply *ALL*

```
t = generate_random_time( 1/(T*cache_size) );
wait(t);
if ( not listened PDP reply)
  PDP reply(cache_0, cache_i);
```

3 Solution for service discovery based on the underlying service discovery mechanisms

Here we present a solution integrated with FIPA specifications, centred in the adaption of the Directory Facilitator to work with any service discovery protocol used by the underlying ad-hoc network technology (BT SDP, Jini, UPnP SSDP, PDP or whatever). This solution is based in the introduction of a new agent, the Service Discovery Agent.

3.1 Directory Facilitator in Ad-hoc networks

We have assumed the following prerequisites to adapt the DF to ad-hoc environments:

- A DF in an ad-hoc network must provide the same functionality than a DF in a traditional network, i.e., it must be a yellow pages service in which agents may register their services (to offer them to other agents), and search for services offered by other agents.
- A DF in an ad-hoc network must keep the same functions specified for the DF in FIPA00023. So, agents do not need to modify the way they interact with the DF. Therefore, it will have the following functions: `register`, `deregister`, `modify` and `search`.
- A DF must provide flexible service search mechanisms that include both local services (in the same platform) and remote services. In ad-hoc networks, remote services will be very changing because devices that offer them join and leave the network continually.

We centre now in the last point. We will argue the need to add a new functional block to the FIPA architecture to adapt it to ad-hoc networks.

In the traditional DF definition, the search of remote services is accomplished by using the concept of DF federation. DFs, besides registering services offered by local agents (native or not), they may also register other DFs (the so called federated DFs). This allows them to extend the search of services to the ones registered in these other DFs. The number of hops between federated DFs is limited by means of a parameter that restricts the depth of a search.

We will see now the disadvantages of DF federation in remote service discovery in ad-hoc networks:

- The search of an specific remote service always implies one communication with that remote system, because we just know that there is a DF in that platform, but not the services registered in it. This means at least one transmission is needed, with no success guarantee.
- If search conditions allow a multihop search, i.e., a search in a remote DF may spread to other remote DFs federated in it, then it is possible that services found in those remote DFs will not be reachable from the first system, and therefore they are not valid search results.

3.2 Service Discovery Agent

To overcome the problems stated above, we propose the introduction of a new agent, the **Service Discovery Agent (SDA)**, in the FIPA architecture for ad-hoc networks. This new agent will be mandatory, and it will have the following functionality:

- To register and to update, in the platform's DF, the list of remote services offered in the ad-hoc network. Therefore, the DF will have entries corresponding to remote services, not to remote (federated) DFs, and so the number of transmissions will be minimised [1].
- To propagate the search of remote services when solicited (because search conditions allow so).

[1] Transmissions will be minimised as long as the SDP (Service Discovery Protocol) the agent uses is adapted to work in ad-hoc environments and makes a minimum number of transmissions to discover remote services

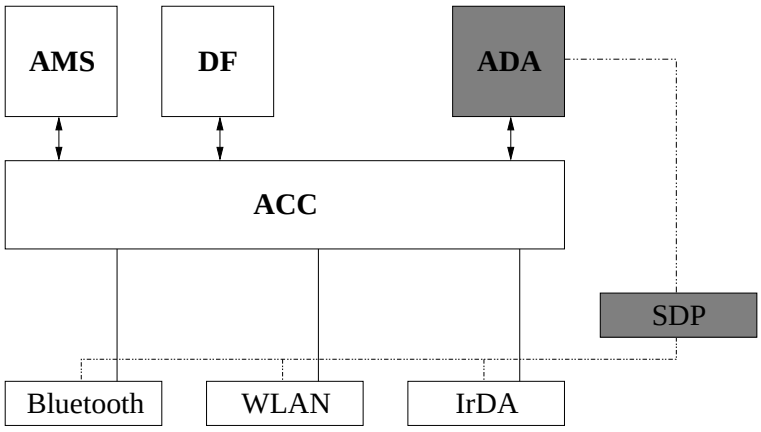

Figure 1: Proposed Architecture for the FIPA Ad-hoc

- To announce services registered in the DF using the associated SDP, when allowed so.

The SDA will use a Service Discovery Protocol (SDP) to discover remote services. In this sense, in order to guarantee an efficient solution for ad-hoc networks, a proper SDP should be selected that fits well into the restrictions exposed in the former section. In the second part of the document, we analyse some SDPs and propose a new SDP for ad-hoc networks.

3.3 Registration of the Service Discovery Agent with the Directory Facilitator

The SDA agent provides a service that is mandatory to be registered in the DF of the platform where it resides. This way, there will be always one entry corresponding to the SDA in DFs in ad-hoc networks, with the following reserved AID:

```
(agent-identifier
  :name sda@hap
  :addresses (sequence hap_transport_address))
```

This agent will register in the DF by setting the `:type` parameter of the `service-description` to the `sda` value.

When the DF processes a `search` and search propagation in the ad-hoc network is allowed (the restriction `max-depth = 1` may be reused for this)

the DF will delegate the search to the SDA.

Each time the SDA (through the SDP) knows about a new remote service in the ad-hoc network, it registers the service in the DF by a `register` request. Since the network is changing, this registers will have an associated `timeout`. The timeout will be managed by the SDA, so when the time expires, the SDA will do a `deregister` request to the DF. The SDA just has to store the corresponding `agent-identifier` and its associated `timeout`.

The SDA in its turn may provide the SDP with the local services registered in the DF. To do so, the SDA will make a `search` request in the DF.

4 Solution for service discovery at an agent level

Service discovery could also be done at an agent level without using the service discovery protocol possibly provided by the underlying network technology. Here we present our proposal in this sense.

Our solution here is also based in the use of a Service Discovery Agent (SDA), but now it will not work in cooperation with a service discovery protocol, but by sending and receiving special multicast ACL messages.

To minimise the number of transmissions we propose the way the SDA will work to be the same of our PDP protocol. That is, when the SDA wants to search a service in the ad-hoc network, it sends a multicast search message to all the SDAs in the network, requesting that service. The answer will be also multicasted, so the other SDAs will also learn from the answer and maintain a cache. As in PDP, just one SDA, the faster one (see the timer mechanism of PDP), will answer the request.

If FIPA TC Ad-Hoc finally adopts a solution like this, it will be necessary to deal with the definition of a multicast format for ACL messages. For example, to contact all the SDA agents present in an ad-hoc network, a multicast message with something like `sda@*` in the `receiver` field could be used.

5 Conclusions

In this paper we have proposed two possible solutions for FIPA platform and service discovery in ad-hoc networks. They both are based in the use of an agent, the Service Discovery Agent. In the first one, any underlying Service Discovery Protocol could be used. We reviewed existing discovery protocols and proposed a new one, the PDP protocol. In the second solution the SDA itself would discover the services through the use of multicast ACL messages in a way similar to the PDP protocol. These multicast messages need to be defined.

One of our premises was not to require modification of existing FIPA specifications, so we have maintained the DF as a mandatory component. If in the near future FIPA allows the DF to be optional, then the SDA of our proposal could assume the functions of the DF, and so no DF would be necessary and the implementation could have an smaller memory footprint.

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.