Service Discovery in Pervasive Multi-Agent Systems

Celeste Campo
Dept. Telematic Engineering
Universidad Carlos III de Madrid
Avda. Universidad 30
Leganés (Madrid), Spain
celeste@it.uc3m.es

ABSTRACT

Agent technology will be of great help in pervasive computing. However, the use of Multi-Agent Systems in pervasive environments poses important challenges. One of them is to allow agents in different devices in an ad-hoc network to share services between them when they are close enough. In this paper, we will define a system agent, the Service Discovery Agent, that helps other agents in search of services offered by other agents or other systems in the network. We will see that none of the service discovery protocols proposed so far in the literature adapts well to the case of pervasive systems, and we will propose a new service discovery protocol, called PDP.

Keywords

Pervasive Computing, Multi-Agent Systems, Service Discovery Protocols, Pervasive Discovery Protocol

1. INTRODUCTION

Recent advances in micro-electronic 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. 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 and going. Agents are autonomous software entities that can interact with their environment, and therefore they adapt well to such frequent 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. One key element in MAS design is the "agent platform". An agent platform provides a runtime environment and a basic set of services that allows agents to do their tasks. The FIPA [2] standard classifies these services into three components:

- The Agent Management System (AMS): this manages the life cycle of the agents, the local resources of the platform and the communication channels, and it also provides a "white-pages" service that allow agents to locate each other by name.
- The Directory Facilitator: this is a "yellow-pages" service which identifies which agent provides what service.
- The Agent Communication Channel: this manages the interchange of messages between agents on the same or different platforms.

In this paper we present the problems associated with building a Directory Facilitator when the Multi-Agent System is intended to work in a pervasive system. The paper is organised as follows: in Section 2 we will present the "Service Discovery Agent", our implementation of the Directory Facilitator. We will see that the Service Discovery Agent needs to implement a service discovery protocol. In Section 3 we will review the main service discovery protocols proposed so far in the literature. We will see that none of them adapts well to the case of pervasive systems. In Section 4 we will propose a new service discovery protocol, PDP. Finally, in Section 5 we sum up the main contributions made in our paper and discuss some of the work we intend to do in the future.

Workshop on Ubiquitous Agents on embedded, wearable, and mobile devices
2002 Bolonia, Italy

2. SERVICE DISCOVERY AGENT

To implement the Directory Facilitator, we propose an agent, that we call the Service Discovery Agent (SDA). This is a special kind of stationary agent that provides resources for other agents on the same platform. Some authors call this kind of agent "middle agents" or "system agents" [1].

The SDA is a system agent that helps other agents working on the same platform as it goes in search of services offered by other devices in the network. It is a kind of "yellow pages" agent adapted to the peculiarities of pervasive computing environments. 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.

Other authors have proposed system agents which provide yellow pages services using some of the dynamic service discovery protocols which we will review in the next section. Specifically, [5] describes an implementation using SSDP, for its use in fixed local area networks. In our work we restrict ourselves to a much more limited environment: a pervasive system, i.e., an ad-hoc network of limited devices.

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 SDA will be based on this protocol.

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

3. SERVICE DISCOVERY

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

The Service Location Protocol (SLP) [4] is an Internet Engineering Task Force standard for enabling IP network-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 the UAs unicast their requests to the DA, and when there is no DA, the UAs repeatedly multicast the request they would have unicast to a DA. SAs listen for these multicast requests and unicast their responses to the UA.

Jini [6] 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. 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 [8] 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.

Simple Service Discovery Protocol (SSDP) [3] was created as a lightweight discovery protocol for the Universal Plug-and-Play (UPnP) initiative, and it 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.

The solutions described above can not be directly applied to the scenario we treat in this paper, 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 which are 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 in.

- The "Pull" solution, in which a device requests a service when it needs it, and devices that offer that service answer 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 discovery in pervasive systems without using a central server. The DEAPspace Algorithm [7] 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 at a device where some of those services are in fact not available.

In this paper 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.

4. 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 as 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.

4.1 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 Service Discovery Agent 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 device has promised to remain available, T, and the time that the server announced that the service would be available for. Entries remove themselves from a cache when *e.timeout* elapses.

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);
}
```

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 ($cache_0$).

4.2 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 Service Discovery Agent uses these primitives.

4.2.1 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 environment, it requests the service from the SDA. The number of broadcast transmissions should be minimised, so:

- If a specific service has been requested, the SDA 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 SDA updates its caches by sending a PDP request message through all the interfaces of the device (Table 2).

4.2.2 PDP reply

The SDAs in all devices are continually listening on each interface for all types of messages (PDP requests and PDP replies).

When a PDP reply is received, announcing a service, the SDAs update their caches accordingly.

When a PDP request for a specific service S (Table 3) is received, the SDA:

- Checks whether the requested service, S, is one of its local services and therefore is stored in the loopback cache, or is not.

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);
}
```

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);
```

- If it is, 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 SDA 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 SDA:

- 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 bigger 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 SDA 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.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have demonstrated the need to adapt some services in a MAS infrastructure to work in pervasive environments, in particular the yellow-pages service. We have defined a system agent, the Service Discovery Agent, that implements this service, using a new service discovery protocol, the PDP.

This paper describes our initial work in an important open issue in MAS in ad-hoc networks: service discovery. Much work in this area remains to be done, some of which we intend to address in the future:

- We want to measure the efficiency of the PDP algorithm by implementing it in a network simulator, and analysing how well it adapts to different mobility scenarios.
- We will compare the performance of the PDP algorithm with the classical solutions.
- We will improve the PDP algorithm, e.g. :
  - By adding a new message "PDP bye" to be used when a device is switched off or it roams to other network. This will be only possible in certain network technologies.
  - By adding new parameters, besides the availability time which weight the generation of the random time a device waits before sending a "PDP reply", e.g. the QoS, the cost of the service, etc.
- We will define the format of PDP messages and service descriptions.
- We will improve the security aspects of the protocol.

6. ADDITIONAL AUTHORS

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

7. REFERENCES

[1] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the Internet. In *IJCAI-97 International Joint Conference on Artificial Intelligence*, 1997.

[2] Foundation for Intelligent Physical Agents. http://www.fipa.org.

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

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

[5] B. K. Langley, M. Paolucci, and K. Sycara. Discovery of infastructure in multi-agent systems. In *Agents 2001 Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, 2001.

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

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

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