Design and Implementation of a Service Discovery Architecture in Pervasive Systems

Vincenzo Suraci[1], Tiziano Inzerilli[2], Silvano Mignanti[3],
University of Rome "La Sapienza", D.I.S.
[1]vincenzo.suraci@dis.uniroma1.it [2]inzerilli@dis.uniroma1.it, [3]silvano.mignanti@dis.uniroma1.it,

Abstract - **The availability of cheap wireless technologies, which can be interconnected through the Internet, has paved the way to a large fruition of (web) services and the convergence of the Internet with mobile handheld devices. A potential market opportunity could become concrete only if users can easily locate and access the available services, thorough support for Service Discovery. There are a number of service discovery protocols and architectures incompatible with each other and sometimes applicable to only specific networks. Often the use of these protocols can be carried out only by experienced users.**
In this article we will describe an architecture implementing Service Discovery for a Pervasive System, which is simple, effective and compliant with standardization. Whenever a generic user, equipped with his wireless or wired terminal, get access into an IP Network, the Service Discovery service start up finding all the available and suitable services which can be accessed locally or remotely from that access point. This work has resulted in the context of DAIDALOS (Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimised personal Services [1]), a project granted in the European 6th Framework Research Programme, within the IST (Information Society and Technology) thematic area.

Index Terms – **Service Discovery, Service Location Protocol, Pervasive System, Personalization, Context Awareness.**

I. SERVICE DISCOVERY

Whenever services are to be located in a LAN or in the Internet, Service Discovery has to be performed. In order to enable automatic Service Discovery mechanisms, some protocols and software architecture have been proposed (SLP [2,3,4], JINI [5], UPnP [6], Salutation [7], UDDI [8], SDP [9] ecc.). Compatibility, autonomy, and simplicity of service discovery are important requirements for applicability in pervasive systems.
Each Service Discovery solution is characterised by an architecture, a service description model to define service characteristics and rules for filtering and allowing service selection. Existing solutions are often not compatible with each other, in terms of architecture, description model and rules.

In this work we propose a general architecture independent from the particular Service Discovery Protocol, which can perform Service Discovery both in automatic or manual mode, into a pervasive system. We then show a practical implementation based on SLP, which is an IETF standard protocol.

II. SCENARIO

Often service discovery is triggered by a change of context. For instance, a user with a wireless terminal enters in a museum, an office, a university, and she is interested in finding local available services (i.e. a printer, a wall screen, a newscast service). She can either explicitly query the network to be informed about them or receive spontaneous advertisements of services which can be offered in the local environment. Also, instead of looking for a new service, she might need to continue to use a particular one before changing the context.

Service Discovery needs then to be provided both manually and automatically. In addition, it should allow a sufficiently accurate research of service so that, the service discovery process can be followed by an actual utilization of the located services. We will describe an architecture with such features, that enables Service Discovery in a Pervasive System.

III. THE EXISTING SERVICE DISCOVERY ARCHITECTURE

Nowadays almost all the Service Discovery Architectures are based on a simple framework including three main entities:

- **User Agent:** entity which triggers the Service Discovery, namely, it queries a Directory Agent for , available services and receives the list of discovered services.
- **Service Agent** entity which communicates with the Directory Agent to register (publish) or delete the services it provides. Published services can be discovered by a querying user agent.
- **Directory Agent** entity which stores the information on services interacting with service agents, receives queries by user agents, performs filtering on queries and replies to queries providing a list of available services matching the queries.

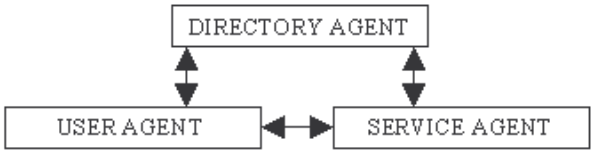

Figure 1 Service Discovery General Framework

Almost all the Service Discovery protocols are based on this kind of architecture. We will extend it, improving its capabilities without adding more complexity.

We need to introduce some critical aspects to describe the whole architecture.

IV. PROFILE and CONTEXT

A generic service discovery could be performed manually using a web search engine, inserting some key words and selecting the wanted web service. By the way, often the service we are looking for is not suitable for our terminal and network capabilities, it does not match our individual preferences, in other words it is not personalized to our needs.

In order to perform a smart service discovery, it can be advantageous to exploit the following information:

- **User Profile:** information about the user preferences, identity and authorisations. It can be used to personalize the services, to identify the user, to perform Authentication, Authorization and Auditing.
- **Terminal Profile:** information about the terminal which are static and do not change over time (i.e. Screen size, Storage and Memory size, etc.)
- **Context:** information about the user and terminal dynamic parameters representing user and terminal status, such as Location, Battery charge, Internet connection type, etc.

Nearly all service discovery protocols do not support the advanced features required for a service discovery process in a pervasive system. Namely, they hardly include Context Awareness and Personalization. **Context Awareness** is essential to provide real time information about the user and the terminal status. **Personalization** perform service adaptation on the basis of user preferences and identity. Quite all service discovery protocols are not platform independent, they are based on a particular network (Bluetooth for the Simple Discovery Protocol), a particular service (Web services for Universal Description, Discovery and Integration, Network services for the UPnP's Simple Service Discovery Protocol) or a particular programming language (Java for the Jini framework).

V. AN PROPOSAL FOR A GENERAL SERVICE DISCOVERY ARCHITECTURE

In this section we describe a general architecture for service discovery. We will concentrate on the interaction between the client and the service discovery server. Main requirements for a definition of an innovative service discovery framework are:

- **Hardware & Software Independency:** the framework should not depend on a particular network, a specific hardware, software, operating system or programming language.
- **Context Awareness:** the framework should use context information (such as location) to provide services which are compatible with the pervasive environment in which the user and terminal are involved in.
- **Personalization:** the framework should be based on user identification and preferences so as to select only the services which satisfy the actual user needs.
- **Compatibility:** the framework should be compatible with all the communication protocols and service discovery protocols.
- **Flexibility:** the framework should be compatible with all future communication protocols and service discovery protocols, and should support the next generation pervasive services.
- **Scalability:** the framework should work on different networks (LAN and internet) and on different clients (Mobile phones, Handled PCs, Notebooks, Desktops, etc.)
- **User Friendliness:** the framework should be easy to use, install and maintain. Use of a friendly graphical user interface is recommended. Low level mechanisms that govern the framework should be transparent to the end user.

Figure 2 depicts the architectures of the User Agent and the Directory Agent as well as their interfaces:

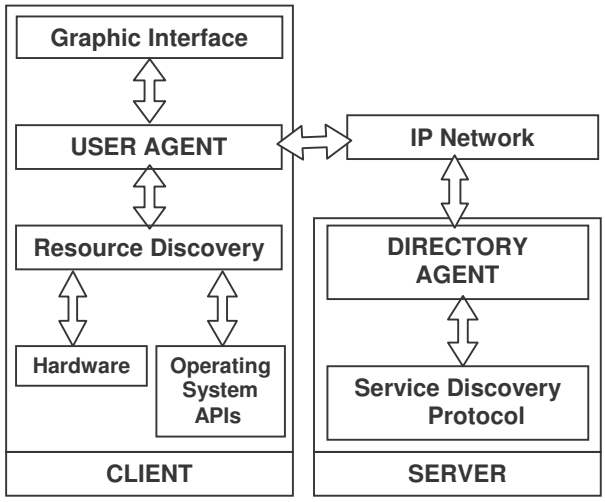

Figure 2 Architecture overview

- **Resource Discovery:** It is the only module which interoperates with the underlaying Operating Systems and Hardware. It provides an adaptation to the specific platform which is used and accesses local system information, in fact it can access information on battery charge, location, connection availability and type, and so on. It then makes the whole architecture hardware and software independent and render the architecture with context aware.
- **User and Directory Agent:** they communicate with each other using a common communication protocol (such as HTTP, SOAP, ecc.). This can be used to tunnel

any specific discovery protocol enabling interaction of user and directory agents across any network. The directory agent receives queries and process them to translate them into specific service discovery protocol messages.
This approach assures compatibility of the architecture to many environments: if a protocol changes, the architecture's structure remain the same. Moreover, the user agent is independent from the specific service discovery protocol used in the service discovery server.
Finally, the directory agent could support Personalization functionalities. In fact it can base the filtering process on user identity information and select queried services which effectively match user individual needs.

- The **client/server** approach provides scalability to this framework: balancing appropriately the load, as the service discovery process could be accomplished even using a thin client (such as mobile phones).
- The use of **IP networks**, provides scalability to this framework in terms of network size. It could operate in a small Local Area Nework, as well as in a larger network, such as the whole internet.
- A **Graphical Interface** provides a user friendly unique interface to the service discovery process. It masks all the underlaying network, hardware and software aspects to the user and it interacts directly with him.

A. Service Discovery Manager

This service discovery framework requires that Client and Server could communicate in a common language. To choose a common communication protocol, the user agent could discover a Service Discovery Manager (SDM) Service. A service discovery manager provides information about the communication interfaces supported by the Directory Agent located in the server.

A user agent could discover a SDM through a standard service discovery process, or by using a proprietary protocol. When a user agent finds a SDM it can use its information to locate the service discovery protocol and to establish a communication channel.

VI. IMPLEMENTATION

This section describes an implementation of the proposed architecture by the University of Rome "La Sapienza".

A. Used Hardware

A PDA (iPAQ 5550) has been used as a client and a Linux based PC for the Service Discovery Server.

A GPS module has been used to detect the PDA geographical position. It is able to connect to the PDA using a bluetooth interface.

A GPRS/GSM PCMCIA card has been used to connect the PDA in the Internet .

B. Protocols

The communication protocol used between User Agent and Directory Agent is HTTP.

The service discovery protocol used in the Service Discovery Server is a Service Location Protocol implementation, in particular the OpenSLP implementation, working both under a linux/unix or windows based environment, was used.

C. Software Modules

As far as the software modules, their implementation is shown in the following figure.

The Directory Agent has been separated into the Web Server and FILTER modules. The first one enables the client/server communication using the HTTP protocol. All the information are encapsulated in HTML documents. The second one translate the incoming information sent by the client, into SLP messages and viceversa.

Due to the fact that Resource Discovery has to communicate directly with the hardware and with the Operating System's APIs, it has been implemented in C++. Instead, the User Agent and Filter have been programmed using Java.

The graphical interface is a web based one, it uses the Java Server Pages (JSP) to create an intuitive graphic user interface that can be accessed from a generic web browser. JSP can be managed by a Tomcat web server, that can be installed as a stand alone web server or as a plug-in of an existing web server (such as IIS, Apache, etc.). JSP can communicate with the filter directly using Java.

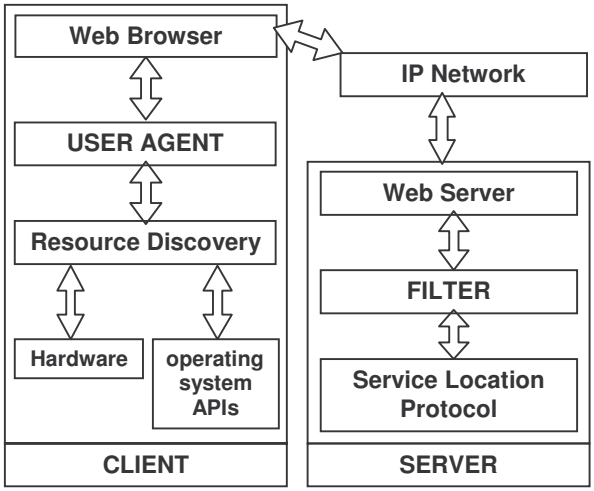

Figure 3 Software modules overview

D. Testbed Description

The network testbed has been represented in figure 4. When Resource Finder and User agent are running in the client, the User Agent tries to find a SDM on the Local Area Network in which the PDA is connected. If an SDM has found, the User Agent controls if it supports the particular protocols described in the SDM.

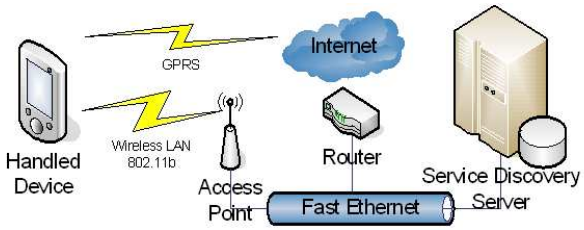

Figure 4 Testbed overview

If no SDM is found, the PDA can connect to a standard (pre-configured) server on the net.

If PDA has no internet connection, and no SDM can be found, the service discovery process can be accomplished only locally using a standard service discovery protocol (if it is supported in that LAN).

We have assumed that the PDA is connected in a LAN that can access the internet. In that LAN a service discovery server supports a SDM based on HTTP communication protocol.

When the PDA receive the SDM information (through a SLP search, casted by the User Agent), the user agent requests the Resource Finder the following information:

- **Terminal Profile** to identify terminal capabilities
- **User Profile** to identify the end user
- **Context** in terms of battery charge, localization information, Internet connectivity, etc.

The Resource finder provides this information in a XML document, through a TCP connection on local port 50000. The user agent connects to the web server (using the SDM information), downloads the main page, populates this HTML document with the XML document, and sends this new HTML document to the default local web browser.

To discover if there is a local web browser, the user agent sends a specific command to the Resource Finder (see figure 5) which is the only module that can communicate with the under laying hardware and software tier.

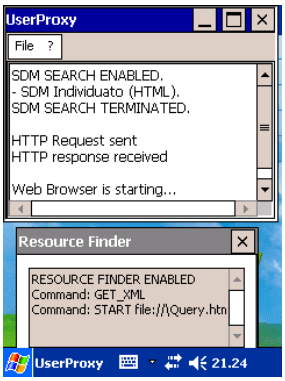

Figure 5 Service Discovery is starting

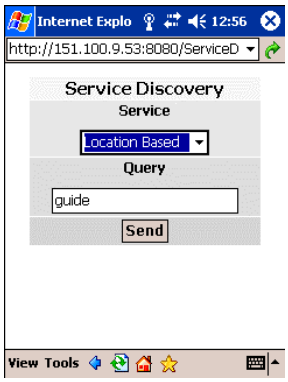

Figure 6 The User performs a Query

Figure 6 shows the client screen before a query for service discovery is made by the user. The end user can submit a service discovery request filling both the Service and the Query fields and pressing the Send button.

The sent web page is provided with the following information:

- Type of service
- Query
- Terminal Profile
- User Profile
- Context

The filter layer translate the end user's query information in specific service discovery protocol's queries. In particular we choose to use the Service Location Protocol (SLP). The filtered services are sent back to the end user web browser.

We tested our architecture using the iPAQ 5550 connected to the GPS receiver trough the bluetooth and connected to internet thanks to the GPRS card.

In the scenario for validation of the implementation we assumed to be a tourist walking in the streets of Rome.

In the following examples we moved near the Colosseum and asked for a guide (figure 7) and for an on-line ticket service (figure 8).

Figure 7 Colosseum Guide Service

Figure 8 Colosseum Online Ticket Service

E. Filtering Algorithm

The introduction of Context awareness and Personalization features for support of the next generation pervasive services (such as location based services), need some extensions to the service registration capabilities. A description model allowing filtering of old as well as more advanced services has to be envisioned.

In general all the service discovery protocols permit to associate an attribute list to a particular service. Each attribute has a name and a list of possible values. The implemented architecture uses some particular attributes "*REQUIRED*" and "*KEYSEARCH*" to record important information for the filtering phase.

The *required* field contains all the parameters regarding the handled device and its owner that are needed by the service to correctly provide its functionalities. For example, if the service is a hi-res video streaming, it would require a display with a minimum width and height, a medium CPU speed and a good battery charge. The filter uses the XML file sent by the end-user with the service request, and the *required* attributes values to evaluate if that service is suitable or not.

The *keysearch* field contains all the key words that should match with the ones inserted by the end-user in the query field (see figure 6).

VII. SLP EXTENSION

The implementation just presented is based on SLP, which is characterised by a simple service description model, which sometimes might be not sufficient to realize complex service queries. We are exploring the potential of the Ontology Web Language (OWL [10]), as a possible add-on to enhance the overall service discovery implementation as far as the service description part is concerned. SLP characterizes the services using a list of attributes, OWL can characterize the services using ontologies, description models which can be represented with an interconnected graph structure (and this structure can be either very simple, using OWL-S, or with a medium complexity, using OWL-DL, or very complex, using OWL-full: either if we are searching a simple or a very detailed way to represent a service entity, OWL seems to be equally the best solution).

The overall idea is both simple and powerful: by adding a "special" SLP attribute (named "OntologyURI") in the SLP service descirption model, we can register each service either with usual SLP attributes and with an OWL ontology. Close to the Directory Agent (DA) we have to use an OWL Proxy Server, which performs OWL parsing.

The entire service discovery process, combining SLP querying and OWL querying and parsing consists in a certain number of steps. The client sends to the proxy either an SLP or an OWL query. Then the OWL proxy server forwards the SLP query to the DA, obtaining a raw list of services (this step is the same as with SLP only). For each returned service, the OWL proxy extracts the OntologyURI attribute and downloads the related OWL file. Now the proxy can perform a query on the ontology: if the current service matches to OWL query requirements, the service is saved, otherwise it is discarded. In a few words, the entire discovery process consists in a two-step discovery, with SLP discovery followed by OWL discovery.

The mechanism described above is being implemented in Java in combination with OWL. We designed and implemented a Java class that deserializes the OWL file into a Java object. So the client side, receives either a list of services and their attributes (this is the SLP part) and a list of instanced Java objects representing the services themselves. This approach requires the client to know the interface of the service. We are now thinking to generalize this approach using SOAP furthering conjunction with SLP attribute to indicate the client where to find the above mentioned interface.

The whole architecture, in an environment where the client knows the class of returned instances, is already adopted within the DAIDALOS project, and is being tested for the CANs' (multimedia Content Adaptation Nodes) discovery, which are a category of highly structured services. With the introduction of OWL the entire process of CAN discovery has been substantially enhanced.

VIII. CONCLUSIONS

In this work we presented a general and innovative architecture for service discovery in Pervasive Systems. It has been designed and developed within the Daidalos European project.

We particularly discussed the main design principles and presented the overall structure of the architecture. We also showed a possible implementation and a demo.

The architecture presented here provides a general and innovative way to perform pervasive service discovery including Context Awareness and Personalization.

This architecture is applicable to many different scenarios in which the service discovery process is necessary and needs to be easy to install and easy to use. Thanks to its capabilities this platform could transform a simple set of services in a real pervasive environment in which the end user feels to be involved in.

ACKNOWLEDGMENT

The authors would like to thank members of DAIDALOS team who partially supported this work by participating to service discovery issues in a context of pervasive computing environment.

REFERENCES

[1]. Daidalos homepage: http://www.ist-daidalos.org
[2].IETF, RFC 2609, "Service Templates and Service Shemes", http://www.faqs.org/ftp/rfc/pdf/rfc2609.txt.pdf
[3].IETF, RFC 2614, "An API for Service Location", http://www.faqs.org/ftp/rfc/pdf/rfc2614.txt.pdf
[4].IETF, RFC 2608, "Service Location Protocol, version 2", http://www.faqs.org/ftp/rfc/pdf/rfc2608.txt.pdf
[5].UpnP Forum, "UPnP Device Architecture", [Online] http://www.upnp.org/download/UPnPDA10_20000613.htm
[6].Sun Microsystem, Technical White Paper,"Jini Architectural Overview", [Online] http://wwws.sun.com/software/jini/whitepapers/architecture.pdf
[7].Salutation Consortium, Internet Draft, "Salutation Architecture Specification", [Online], http://www.salutation.org/spec/s21a1a21.pdf
[8].OASIS, UDDI 3.0 Specification, [Online], http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf
[9].Eugene A. Gryazin, "Service Discovery in Bluetooth", http://www.cs.hut.fi/~gryazin/SD_in_Bluetooth.pdf
[10]. OWL Services Coalition, "OWL-S Semantic Markup for Web Services", http://www.daml.org/services/owl-s/1.0/owl-s.pdf