Design of Service Discovery & Provisioning Architecture based on the Open Services Gateway Initiative

Vincenzo Suraci[1], Massimiliano Taglieri[2]
University of Rome “La Sapienza”, D.I.S.
[1]vincenzo.suraci@dis.uniroma1.it, [2]massimiliano.taglieri@tele2.it

Abstract - **Mobility has become a central aspect of the lives of the most industrialized countries citizens – in business, education, and leisure. Due to rapid technological and social changes there has been a widespread of cheap wireless and wired technologies, which can communicate each other creating an huge set of potential users and services in a pervasive world. This potential business could become concrete only if users can find the available services and get access to them easily and automatically. In this futuristic scenario a very important role is played by the Service Discovery, Service Composition and Service Provisioning Platforms. In this article we will describe an innovative Service Discovery and Service Provisioning architecture based on an open standardized platform: the OSGi framework. This work presents first results of the service provisioning and service discovery design activity which is being carried out within 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 Provisioning, OSGi Framework, Pervasive System, Service Discovery.**

I. SERVICE DISCOVERY

A service discovery process occur whenever something or someone try to find a suitable and available service. A general Service Discovery architecture is depicted in figure 1. It is based on a simply framework represented by a **Directory Agent** that stores the information on services, receives queries, performs filtering functions and retrieves the services matching the queries. A **Service Agent** communicates with the Directory Agent, registers and deletes the available services. The **User Agent** triggers the Service Discovery process manually or automatically, communicates with the Directory Agent, performs queries on services and receives the available and suitable ones.

To enable an automatic Service Discovery some protocols and software architecture have been defined: SLP, SDP, UPnP, Salutation, Parlay, etc. and quite all the these solutions are based on the above described architecture. But they are not compatible each other, due to the different architecture, rules or ontology.

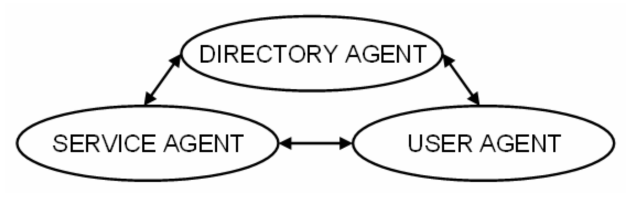

Figure 1 Service Discovery General Framework

We will extend this simple service discovery architecture, improving its capabilities and providing the end user with the required services in an innovative, automatic and pervasive way.

II. SERVICE PROVISIONING

Whereas a user want to use the functionalities of a particular service, a service provisioning process occurs. In literature some protocols and middleware architectures have been defined to provide the services’ functionalities to a client machine, such as CORBA, DCOM, Java RMI, JINI, WS Provisioning, etc.

Each of these technologies uses a different service provisioning approach. For example CORBA, DCOM and Java RMI are middleware that extend the concept of object-oriented programming system by allowing objects to be distributed and invoked across a heterogeneous network, so that each of these distributed object components interoperates as a unified whole [5][6][7].

Instead the JINI approach is based on three main players: there is a *service*, a *client* which would like to make use of this service and there is a *lookup service* (service locator) which acts as a locator between services and clients.

A service provider sends a request to register the *service object* that implements the service to the lookup services. When the lookup service gets a request, it sends back to the server an object known as a *registrar*. It acts as a proxy to the lookup service and any requests that the service provider needs to make of the lookup service are made through this proxy registrar.

The client tries to get a copy of the service through the same mechanism (it must get a *registrar*). But this time it does something different with this, which is to request the service object to be copied across to it (service provisioning operation) [8].

Finally it's interesting to analyze the service provisioning issues by a Web Services point of view. In this context many protocols or initiatives have been defined and often they interact to provide Web services. The most important are: SOAP (provides the definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralized distributed environment [9]), WSDL (provides a model and an XML format for describing Web services), UDDI (defines a SOAP-based Web service for locating WSDL-formatted protocol descriptions of Web services) and WS Provisioning Specification (describes the APIs and schemas necessary to facilitate interoperability between provisioning systems).

III. THE OSGi FRAMEWORK

The Open Services Gateway Initiative (OSGi) was founded in March 1999. Its mission is to create open specifications for the network delivery of managed services to local networks and devices. Services are Java objects implementing a concisely defined interface [2][3].

The Framework forms the core of the OSGi Service Platform Specification. It supports the deployment of extensible and downloadable service applications known as *bundles*.

In the OSGi environment, bundles are the only entities for deploying Java based applications. A bundle comprises Java classes and other resources which together can provide functions to end-users and provide components to other bundles, called *services*. A bundle is deployed as a Java ARchive (JAR) file.

OSGi-compliant devices can download and install OSGi bundles, and remove them when they are no longer required. Installed bundles can register a number of services that can be shared with other bundles under strict control of the Framework.

The Framework manages the installation and update of bundles in an OSGi environment in a dynamic and scalable fashion, and manages the dependencies between bundles and services.

The Framework allows bundles to select an available implementation at runtime through the Framework service registry. Bundles register new services and receive notifications about the state of services. This aspect of the Framework makes an installed bundle extensible after deployment: new bundles can be installed for added features, or existing bundles can be modified and updated without requiring the system to be restarted.

The OSGi framework for its characteristics is the more suitable for the design and deployment of an innovative service provisioning architecture.

IV. PERVASIVE SYSTEMS

The evolution of modern technologies allow to achieve an important objective: to involve the user into a system that can answer to his requirements providing him with useful information and services. This kind of system is called Pervasive System.

For this reason an intelligent service provisioning architecture must be able to select and personalize the services that the user wishes. This property is called Content Adaptation.

Moreover when an user handles a terminal, it's possible that context's changes occur (e.g. low battery, connection's interruption etc.). A good architecture should react and compensate the consequences of these changes. This characteristic is called Context Awareness.

In order to implement these functionalities, it is necessary to manage the following information:

User Profile that maintains the user identity, preferences and properties so that it can be used to personalize the services, to perform Authentication, Authorization;

Terminal Profile that maintains data on the terminal static capabilities (i.e. Storage and Memory size, Screen size, etc.);

Context that maintains data on the user and terminal variable capabilities (e.g. Location, Battery charge, Internet connection, etc).

V. SCENARIO

Who has a handled device, entering in a new pervasive environment (e.g. station, office, vehicle etc), could be interested to identify the presence of useful and available services. This can be done triggering the Service Discovery service from the terminal both manually or automatically. If the search has a positive outcome, the user must be able to access to the chosen services automatically (i.e. without executing complicated system configurations). These services can be static information (e.g. in a railway station, it could be the train timetable) or services that can be managed through software applications (printers, wall display, fax, etc.). In short the user should be able to execute a Service Provisioning process.

We will describe an architecture which can perform all these functionalities, using any of the existing Service Discovery Protocols and the OSGi platform.

VI. PROBLEM DESCRIPTION

Currently service provisioning frameworks don't care about the new functionalities required for a service provisioning process in a pervasive system.

In particular they don't support the automatic download and installation of service applications and they can only execute remote method invocations or use web services. Moreover they don't care about Context Awareness, Content Adaptation and Service Composition. These characteristics are basic for the next generation pervasive services.

It is also necessary to consider that quite all service discovery protocols are not platform independent: they are bound to a particular network (e.g. Bluetooth for the Simple Discovery Protocol) or a particular service (e.g. Web services for the Universal Description, Discovery and Integration, Network services for the UPnP Simple Service Discovery Protocol).

VII. AN INNOVATIVE SERVICE DISCOVERY & PROVISIONING ARCHITECTURE

A. Requirements

In the project development of an innovative service discovery and provisioning framework, first step is to focus on the project's requirements.

The framework should be **Hardware & Software Independent**: it couldn't depend on a particular network, a specific hardware, software or operating system. The framework should be characterized by a high level of **Automation** and it would have to provide mechanisms allowing to the end user to install and to run automatically the services in transparent way. When the architecture interacts with the end user it should be **User Friendly**, hopefully using well known graphic user interfaces (windows like or web based).

The framework should use **context** information (such as localization) to provide services which are compatible with the pervasive environment in which the user and the terminal are involved in. Moreover it should provide **content adaptation** services, for example using the user identity to find its service's preferences and to select the services which are compliant with the user characteristics only.

Compatibility with all existent and future communication service discovery protocols it's necessary for a successful architecture. Finally it should be **scalable**, i.e. it should work on different networks and on different terminals.

B. Service Discovery

An important and innovative framework feature is the independence from the adopted service discovery protocol.

In order to pursue this objective we have planned a *Service Discovery API (bundle)* that makes it available some service discovery functions (such as the service registration, discovery, de-registration etc.) independently from their real implementation (SLP, SDP, Salutation, etc.).

As depicted in the figures 2, 3 and 6 we have chosen to use the Service Location Protocol (SLP) developed by the IETF. It offers a more flexible and a more scalable architecture than all the other discovery protocols.

SLP includes a leasing concept with a lifetime that defines how long a directory agent will store a service registration (peculiarity useful for highly dynamic wireless network scenarios). It supports Service Browsing functionality (in order to find the services in network) and mechanisms for the attributes selection to realize complete queries. Finally it is independent by programming language.

C. Framework

Three agents interact in a service discovery and provisioning scenario: a Directory Agent, a Service Agent and an User Agent.

The innovative characteristics of the OSGi platform suggests to use it in all the three agents, so that the problem's solution is designing an efficient bundle architecture.

The **Directory Agent** provides an OSGi platform and a database application in which all the services could be registered.

The *Directory Agent Bundle* provides all the needed functionalities to register a service into a database, to retrieve it starting from a query and to delete it. The different implementation of the Directory Agent Bundle uses different Service Discovery Protocol.

The *Service Location Protocol Directory Agent Bundle* represents a SLP Directory Agent working in the OSGi platform. When it starts, it listens to the port 427 for incoming SLP messages.

The *http server bundle* is useful in order to allow the User Agent the download of a Service Provisioning Manager Bundle, which represent a simple .jar file registered in the http server root.

We'll describe the service provisioning manager bundle functionalities later in this document.

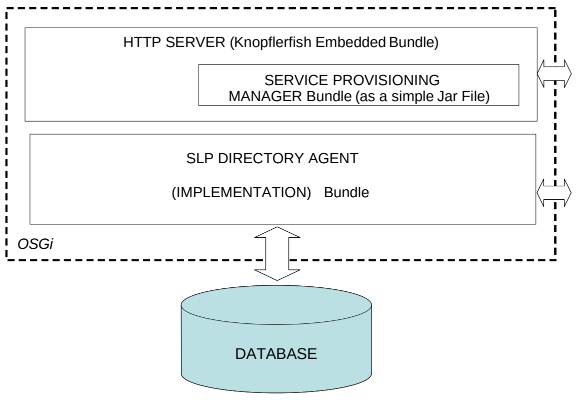
HTTP SERVER (Knopflerfish Embedded Bundle)
SERVICE PROVISIONING MANAGER Bundle (as a simple Jar File)
SLP DIRECTORY AGENT
(IMPLEMENTATION) Bundle
OSGi
DATABASE

Figure 2 Directory Agent

When a Directory Agent Node is activated, the Directory Agent Implementations installed in the OSGi platform starts to listen for possible incoming service requests.

The change of the Directory Agent Implementation make possible to support different service discovery protocols (SLP, UPnP, UDDI, Salutation, etc.) using the same API for the service discovery.

The **Service Agent** main bundle is the *Service Agent Bundle*. It coordinates the other bundles. When the service agent get connected to the net, the *LAN listener Bundle* detects this event and activates the service agent bundle. It retrieves the list of the services installed in this machine and try to register the services into the LAN using the *Service Discovery API*.

The *Service Discovery SLP bundle* depicted in the figure 3 and 6 implements the *Service Discovery APIs*, using the SLP protocol standard. The adopted solution is very useful because allows the same service agent to use different service discovery protocols without modifying the other bundles.

The *Service Bundle Installer bundle* allow the server administrator to manage the available services by means of a graphic user interface, to install new services, to delete and to update the old ones and so on.

The registered services could be simple web pages (information services) or complete software programs (OSGi bundles), and they are stored into the http server running as a service in the OSGi platform (*http server bundle*).

A generic *service bundle* stored into the http server bundle is a stand alone software that can communicate both with the real service or a controlling software installed into the Service Agent, using any communication protocol. It's possible because the service bundle has been built by the service provider itself. For example: a wall display connected to the service agent machine provides a simple service bundle that could be installed in the client OSGi platform and allows the client machine to send a video streaming representing the client's display directly to the wall display.

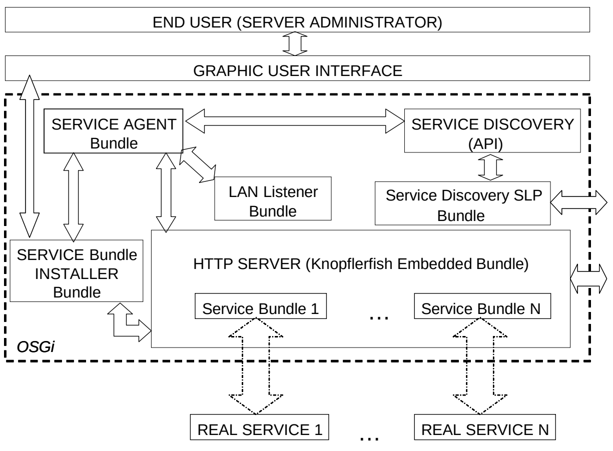

Figure 3 Service Agent Node

This is an innovative approach to provide next generation pervasive services to the end user, who can interacts directly with the software and the hardware around him. Moreover using the OSGi platform all these processes are completely automated.

The **User Agent** main bundle is the *User Agent Bundle*. When a mobile user get connected to the net, the *LAN Listener Service bundle* activate the user agent bundle that searches a *Service Provisioning Manager*, i.e. something that could accomplish all the service discovery and provisioning processes automatically, requiring the minimum interaction with the end user.

The user agent bundle uses the *Service Discovery APIs* to find a available service provisioning manager. If a Directory Agent is working in the LAN, it could listen the user agent request and return to it the URL from which a service provisioning manager could be downloaded.

When the user agent receive the directory agent response, it uses the *Bundle Loader bundle* to download and to install the *Service Provisioning Manager bundle* into the OSGi platform.

This bundle enables the end user to perform a service discovery and provisioning process interacting with him by means of a graphic user interface. For example the end user can select the service type and/or some key words to accomplish a general service research. Moreover he can trigger the downloading and the installation of the services. The figure 4 shows the printer services discovered using an iPAQ 5500 connected in a small office WLAN empowered with the service provisioning architecture. The figure 5 shows the Knopflerfish OSGi console that makes it possible to manage the bundles installed in the *User Agent Node*.

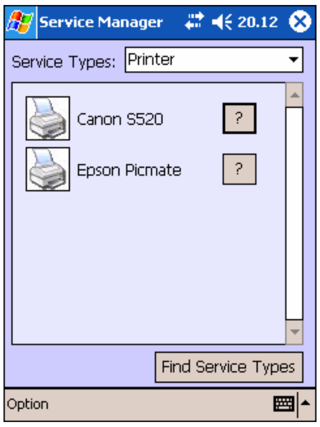

Figure 4 Printer Service Discovery on PDA

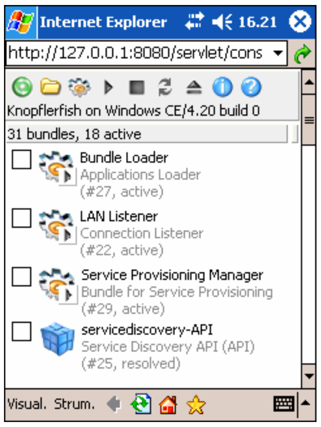

Figure 5 Knopflerfish OSGi Console on PDA

The service provisioning manager can also use the information about the user profile (name, address, telephone number, etc.), the terminal profile (display size, cpu speed, installed software, etc.) and the context status (position, connection speed, battery charge, etc.) provided by the *Resource Finder Bundle* to perform a better service request. All this information could be collected by a specific software (the *Resource Finder*, generally written in a low level programming language such as C++) that communicates directly with the under laying hardware and operating system.

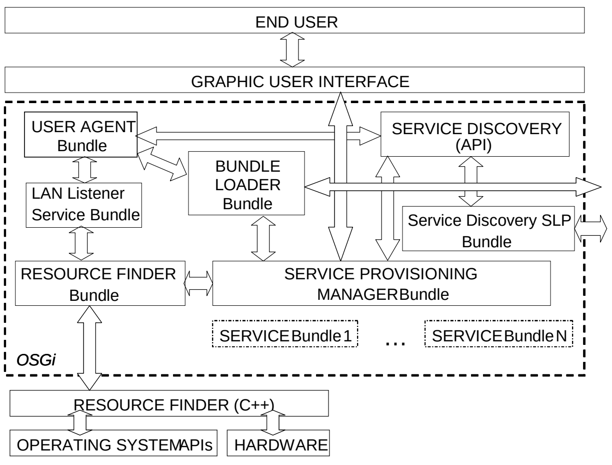

Figure 6 User Agent Node

The service provisioning manager uses the service discovery API to find the suitable and available services in that network. The received list of services could contain URLs of web pages or downloadable bundles. As first case, the local favourite web browser could be used to provide the service to the end user. In the other case, the bundle loader bundle must download and install the service bundle into the local OSGi platform.

VIII. CONCLUSION

In this work we presented an innovative service provisioning and discovery platform based on the OSGi framework investigated in the context of the DAIDALOS project. For its nature, this architecture is easy to install and easy to use. It can express its potentiality in many scenarios in which both user agent and service agent could be mobile agent: whenever a LAN connection has detected the service discovery and provisioning process starts automatically involving the user in a concrete (and not only potential) pervasive environment.

FUTURE WORKS

We are developing the entire architecture using a open source OSGi implementation (Knopflerfish). We want to test it in a wireless LAN in which the user agent is a mobile node. In particular the end user uses a last generation handled device (iPAQ 5550) provided with a 1.3.0 compatible java virtual machine and the Knopflerfish OSGi framework.

ACKNOWLEDGMENT

The authors would like to thank members of DAIDALOS team who provide a fundamental support in the design of this service discovery and provisioning platform.

REFERENCES

[1].Daidalos homepage , http://www.ist-daidalos.org

[2].OSGi, “about the OSGi Service Platform” Technical Whitepaper, Revision 3.0, 12 July 2004, OSGi Alliance, http://www.osgi.org/osgi_technology/download_specs.asp?section=2

[3].Knopflerfish OSGi, “OSGi R3 framework” On line, last update October 2004 http://www.knopflerfish.org/index.html

[4].OMG, “Object Management Group Information”, On line, last update January 2005, http://www.omg.org

[5].CORBA, “CORBA Information”, On line, last update January 2005, http://www.corba.org

[6].DCOM, “Calling COM components from .NET Client”, On Line, Microsoft, last update November 2001 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/callcomcomp.asp

[7].JavaRMI, “Java Remote Method Invocation Specification”, Technical Whitepaper, , Revision 1.8 java 2 SDK, Standard Edition, v1.4, Sun Microsystem, ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf

[8].Jini, “Jini Architecture Specification”, On line, Copyright 1997-2003 Sun Microsystems http://www.jini.org/nonav/standards/davis/doc/specs/html/jini-spec.html

[9]. SOAP, “Web Services Activity”, On line, Copyright 2002-2004 W3C http://www.w3.org/2002/ws/

[10]. IETF, RFC 2608, “Service Location Protocol, version 2”, http://www.faqs.org/ftp/rfc/pdf/rfc2608.txt