COMMUNICATIONS OF EASST

Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence

André Bottaro*, Anne Gérodolle*
*France Telecom R&D

Abstract. *This paper explores Service Oriented Architecture in Ambient Intelligence with focus on distribution and dynamic selection. We aim at building a framework transparently handling these aspects.*

Keywords: Service Oriented Architecture, Ambient Intelligence, OSGi.

1 Introduction

Ambient Intelligence vision assumes that the computers will fade into the background. Building consistent context-aware application inside and outside the home with heterogeneous devices which spontaneously enter and quit the network is the objective of software designers in the field.

This paper proposes a simple declarative model to automate the mutual discovery and binding of components running on distributed network nodes. The model and the associated software architecture above OSGi [14] and *Service Binder* [3] are described in details. Service Binder Model is now integrated in OSGi R4 specification as *Declarative Services* [13]. A middleware hiding the multiplicity of service discovery and distant communication protocols can be built above this model. Many projects aim at defining such a pervasive middleware [5], [4], [15]. Current European IST projects like Amigo, Daidalos and SeCSE tackle with service composition in distributed networks. This paper introduces a pluggable architecture addressing a generic way to interact with multiple service discovery and distant communication protocols.

The implementation specification lies on several technological choices:

- The core framework is Java-based and uses the OSGi standard R3 specification [14] and Service Binder Model [3].
- Discovery technologies are well-known standards: UPNP/SSDP [18], Jini [10], SLP [7], Web Services [6], and CORBA [12].

Service Oriented approach supporting the needs of targeted pervasive applications. is described in Part 2. The implementation of the concepts is detailed in Part 3. Part 4 describes a scenario of use. A conclusion is given in Part 5.

2 A Service Oriented Framework

In smart environments, devices may randomly enter and quit the network., users come and quit the environment with PDAs and mobile devices, install new devices, old devices are updated or replaced, etc. Moreover, devices may deliver services with varying QoS and properties. Some devices may become useless while others may become relevant in distinct situations. Simply identifying every device in our application and being able to deal with the dynamic availability and relevance of these devices are important needs in pervasive computing.

Service oriented programming is an adequate paradigm dealing with these needs. In Service Oriented Architecture, software components register services with associated properties to be requested by other components acting like service clients. Registration and requests are based on the use of a service repository which can possibly be centralized, replicated or distributed through the use of multicast mechanisms.

Distributed Service Oriented Architecture. Numerous Service Oriented Architecture specifications address the issue of Service Discovery and the issue of Distant Communication in distributed environments. If most of the specifications address the same generic issues, they have distinct objectives and are made efficient for distinct environments [15]. UPnP specifies the use of several web-oriented protocols in order to build an attractive solution for small and dynamic networks. Jini aims at specifying a Java middleware for distributed applications on larger networks. SLP is also meant for SOHO (Small Office, Home office) networks. The specification proposes an interesting way to move from multicast discovery mechanisms to centralized one. Web Services may target SOHO network with specifications like WS-Discovery (Web Services Dynamic Discovery). Several specifications make service discovery relevant on CORBA: Naming Service, Trading Service, Event Service. CORBA specification is really large and is not meant for a particular environment. Other comparison details are found in [2].

However, those middlewares do not hide dynamic service availability to application developers. The latter have to repeatedly write error-prone code to request available services and listen to service arrival and departure. Distribution is a non-functional need which has to be dealt by the underlying framework.

Dynamic service availability is shown as a non-functional need in [3][8] which describe a model named Service Binder automating service discovery and service binding in a dynamic service environment. Service Binder model [16] is a simple service oriented model above OSGi Release 3 [14] enabling developers to encapsulate consistent code into components providing and requiring services to each other. The provided and required services are simply declared by every component. A centralized instance manager factory builds an instance manager for every instance of declared components. This instance manager manages component life cycle according to the availability of required services. The implementation of this model greatly simplifies application programming. The developer should only think about the granularity of his application in terms of components and specify what generic services could be provided by components for internal or external application purposes.

However, Service Binder model shows some limitations in order to be the underlying framework of our architecture. First, service properties declared by every service provider are statically written in an XML file and could not be dynamically changed at runtime. This limitation is explained below. Second, the automation of service discovery and service binding is only performed on a single platform. These limitations are overcome by the extensions proposed in part 3.

3 Hiding distribution and managing dynamic service properties

3.1 Hiding distribution mechanisms complexity.

The extension of service discovery and distant communication to distributed mechanisms has to be transparently managed by the underlying framework. This section describes how mechanisms are extended over Service Binder Model in the proposed architecture. This extension is called Extended Service Binder.

Extended Service Binder relies on “export and binding factories” to build and use remote object references, and on “lookup services” to register references with properties and find remote references on the network. Bundles and services using Service Binder extended mechanisms interact with all available OSGi services on the local platform and exported ones on the network. Extended Service Binder is a container managing non-functional aspects.

Remote binding relies on the export-binding pattern defined in ODP [9] and defines two Java interfaces defining the concepts of export and binding factories. An “export factory” is a service that makes a Java object remotely available. The result of “Exporting a service” is a “binding description” that can be serialized and published using a discovery protocol.

Symmetrically, a “binding factory” is used on the client side to bind to a given service, given a “binding description”. A binding factory provides a “bind” method that takes a binding description as parameter and returns an object called “proxy” or “stub”. This stub can then be used by the client to communicate with the remote object.

Extended Service Binder relies on Lookup Services representing remote or local non-OSGi service registry on the platform. They implement the same service interface enabling service providers to register and deregister services, service clients to look up services and be notified by events concerning filtered services.

Extended Service Binder transparently manages distribution. A simple boolean value in a metadata file can be set by the developer to indicate that its service providers are wished to be announced on the network with one or every available discovery protocol ("registry='local, slp,upnp' or "“registry='*'” to be seen in examples, part 4.3) or that its service clients may bind remote providers (“local-only=false” in service client XML description, part 4.3). Transparency is reached with static or dynamic stub generation. A proxy for every targeted technology has to be generated for any used remote service provider. Extended Service Binder structure is illustrated in Fig. 1.

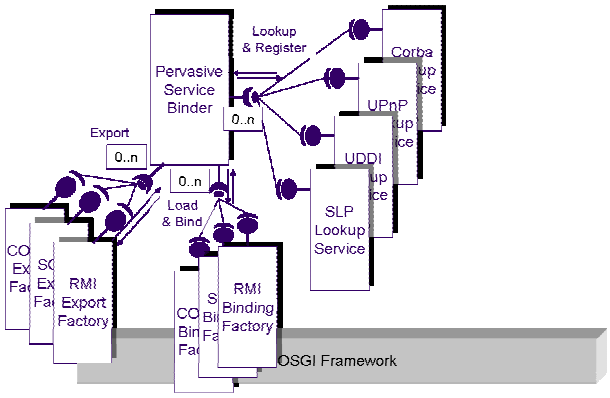

Fig. 1. Extended Service Binder

3.2 Extend the component XML metadata model to manage dynamic properties.

The service dependency management is a non-functional aspect which has to be held separately. Using an external XML file is a natural way to make the separation clearer.

On the one hand, a service requirement in Service Binder Model is described in an XML reference to a service interface with binding and unbinding methods and some dependency properties: policy, cardinality, filters (see example, part 4.3). In order to dynamically add selection mechanisms after service filter at runtime, the declaration of a dynamic selection method could be added. It will be called at service dependency instantiation time. A method called "sort-method" is to be seen in the example, section 4.3.

On the other hand, a provided service in Service Binder Model is described in an XML reference to an object class implementing the provided service interface with statically defined properties (see example, part 4.3). In order to dynamically modify the properties of a provided service, similar extensions are to be held. The declaration of a method setting dynamic properties in provided service declaration and a simple method calling Service Binder to modify service registration through Component Context object could be added. A method called "property-method" is to be seen in the example, section 4.3.

4 An example of pervasive application: audio streaming follow-me

In “follow-me” applications [11] , a service follows the user as he/she moves by borrowing interfaces from devices in the user’s vicinity. Following this concept, Extended Service Binder has been applied to build an “audio streaming follow-me” service. For the sake of simplicity, only a subset of the involved components is detailed here.

4.1 The service offered to the user

When the user moves inside the home, the audio stream "follows" him/her by using the best available rendering resource in his/her vicinity, as illustrated in **Fig. 2**.

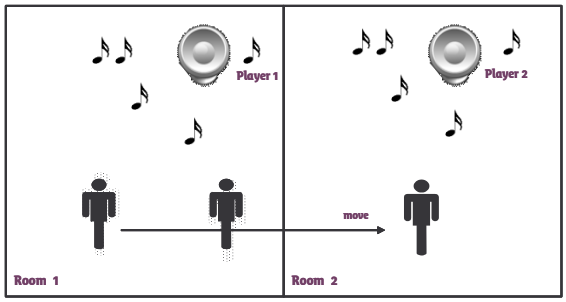

Fig. 2. Follow–Me: When the user moves from a room to another, the player of the second room starts playing and the first stops playing after a while.

4.2 Architecture

To offer this service, the following components were identified. The links between these components are illustrated in **Fig. 3**. The first four components provide services whose Java interfaces are written below.

- A user localisation service based on several localization techniques.
- Several audio renderers able to connect and render an available audio stream.
- One or several yellow pages servers able to give a list of available stations
- A "control point" and "control point factory" whose role is detailed later on.
- A remote control enabling the user to select an audio content and control rendering.

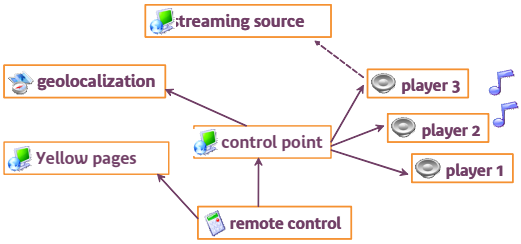

Fig. 3. Architecture of the follow-me application.

4.3 Bundles, components and algorithm details

Each of the main components is packaged in an OSGi bundle and is described in the Extended Service Binder model.

The component providing the AudioRenderer interface is hosted by a bundle which must be deployed on every device with loudspeakers that may be used for audio rendering.

```
<bundle>
  <component class="renderer.AudiorendererImpl">
<onregister property-method="getDeviceProperty"/>
<provides service="api.AudioRenderer" registry="*"/>
  </component>
</bundle>
```

Before registering the AudioRenderer component, the Extended Service Binder calls the "getDeviceProperty" method. This method gives the location of the speakers and the "quality" (built-in loudspeakers, external loudspeakers, hi-fi, etc.) of the rendering. The service is then registered with these properties.

The control point bundle provides a factory of "control points". A specific control point is dedicated to any active user and ensures that the stream chosen by the user is rendered on the adequate platform. To that purpose, it needs to access the localisation service and the available audio renderers. The control point bundle is typically deployed on an "always on, always connected" OSGi platform.

```
<bundle>
<component class="control.CPFactoryImpl">
  <provides service="api.CPFactory" registry="*"/>
  <instantiates class="pack3.ControlPointImpl">
   <provides service="api.ControlPoint" registry="*"/>
   <requires
        Service="api.LocalisationService"
        cardinality="1..1" policy="dynamic" local-only="false"
        bind-method="bindLocService" unbind-method="unbindLocService"/>
   <requires
        Service="api.AudioRenderer" local-only="false"
        cardinality="0..1" sort-method="sortRenderers"
        bind-method="bindRenderer" unbind-method="unbindRenderer"/>
  </instantiates>
</component>
</bundle>
```

The control point currently keeps: a current renderer, a former renderer, and a current context (i.e. the multimedia resource being currently played, the volume information and other rendered properties).

The "sortRenderers" method chooses the available renderer which is in the same room as the users, and (if several) the one with the best rendering quality. This method is called by the Service Binder when the current renderer quits the network, when a new audio renderer enters the network, when an audio renderer modifies its registration, or when discovery is asked to be refreshed by the control point. This refreshment can be asked when the location of the user changes. According to the result of sortRenderers, the method "bindRenderer" may be called by the Service Binder if another audio renderer service must be bound. Whenever the user moves from a room to another, the Extended Service Binder binds the control point to the best available audio renderer.

The "bindRenderer" method sets the local "current renderer" and "former renderer" information and initializes the current renderer with the current playing context. This method call is followed, after

some delay, by the call to "unbindRenderer" method calling the "stop" method of the former renderer. Thanks to the delay between the calls to bindRenderer and unbindRenderer, the user can still listen to the audio stream while the new renderer starts buffering the audio stream.

5 Conclusion and future work

We have presented a middleware architecture called Extended Service Binder, which facilitates the building of pervasive applications. The following features are delivered: transparent access to remote services available through any service discovery protocol, selection between services according to the current context.

Thanks to this architecture, a real peer-to-peer dynamic composition occurs at runtime. Service composition is spontaneously held by the different components on the distributed platforms. Service binding is optimized: Two components running on a single platform are directly bound in the local virtual machine with direct method calls whereas a protocol for distant communication is automatically used if they run on distinct platforms.

The drawback of this approach is that Java interfaces must be agreed in advance between service providers and requesters. Future work will tackle the interoperability between similar interfaces defined through distinct syntactic languages. The definition of useful ontology is currently ongoing with the use of OWL-S. The semantic Web services approach [1] leverages the use of formal and machine-understandable descriptions to enable a flexible matching between heterogeneous services. We are currently investigating such an approach in a pervasive computing context.

6 Acknowledgments

We are particularly grateful to the RNRT (French Research Network in Telecommunications) through the PISE project, which offers us the opportunity of trying out some of our ideas presented in this article in an industrial environment.

We would like to also thank IST, the European research program. This work was partially founded by IST through the Amigo project in Ambient Intelligence.

7 References

1 Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, Katia Sycara, "DAML-S: Web Service Description for the Semantic Web", First International Semantic Web Conference, Sardinia, Italy, June 2002

2 Christian Bettstetter, Christoph Renner, “A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol”, In Proc. EUNICE Open European Summer School, Twente, Netherlands, Sept 13-15, 2000

3 Humberto Cervantes, Richard S. Hall, “Automating Service Dependency Management in a Service-Oriented Component Model”, Proc. 6th Wksp, Component Based Engineering, May 2003

4 Caroline Funk, Christoph Kuhmünch, Christoph Niedermeier, "A Model of Pervasive Services for Service Composition", CAMS05: OTM 2005 Workshop on Context-Aware Mobile Systems, Agia Napa, Cyprus, October 2005

5 Paul Grace, Gordon S. Blair, Sam Samuel."ReMMoC, "A Reflective Middleware to Support Mobile Client Interoperability", Proceedings of International Symposium on Distributed Objects and Applications (DOA), Catania, Sicily, Italy, November 2003

6 Steve Graham, Doug Davis, Simeon Simeonov, Glen Daniels, Peter Brittenham, Yuichi Nakamura, Paul Fremantle, Dieter Koenig, Claudia Zentner, “Building Web Services with Java”, Sams Publishing, Second Edition, 2004

7 Erik Guttman, Charles Perkins, John Veizades, Michael Day, “Service Location Protocol, Version 2”, RFC 2608, June 1999
8 Richard S. Hall, Humberto Cervantes, “Challenges in Building Service-Oriented Applications for OSGi”, IEEE Communications Magazine, May 2004
9 ITU-T & ISO/IEC, “ODP Reference Model: Overview, Foundations, Architecture”, Recommendations X.901, X902, X903 & International Standards 10746-1, 10746-2, 10746-3, 1995
10 Jini.org, “The Community Resource for Jini Technology”, http://www.jini.org
11 Robin Kirk and Jan Newmarch, "A Location-aware, Service-based Audio System", IEEE Consumer Communications and Networking Conference, 2005
12 The Object Management Group, “Common Object Services Specification”, http://www.omg.org/technology/documents/formal/corbaservices.htm
13 OSGi Alliance, OSGi R4 Core Specification, October 2005
14 OSGi Alliance, “OSGi Service Platform – Release 3”, March 2003
15 Pierre-Guillaume Raverdy, Valérie Issarny, “Context-aware Service Discovery in Heterogeneous Networks”, Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2005), June 2005
16 Humberto Cervantes, Richard S. Hall, “Simplifying application development on OSGi”, documentated open source code, http://gravity.sourceforge.net/servicebinder/
17 Sun Microsystems, “Java RMI”, http://java.sun.com/products/jdk/rmi/
18 UPnP.org, “The UPnP Forum”, http://www.upnp.org