Pervasive spontaneous composition

André Bottaro[1], Anne Gérodolle[1], Philippe Lalanda[2]

[1]France Telecom R&D
28 chemin du vieux Chêne
38243 Meylan cedex, France
{firstname.lastname}@francetelecom.com

[2] LSR-IMAG, 220 rue de la Chimie
Domaine Universitaire, BP 53
38041 Grenoble, Cedex 9, France
Philippe.Lalanda@imag.fr

Abstract

Pervasive Computing is becoming feasible. Service discovery and communication protocols now make it possible to build pieces of software that can be dynamically composed into distributed applications. However, when designing such software, developers must often write code related to distribution and discovery aspects. This non-functional code results in higher development costs and may raise interoperability issues. This paper describes a service-oriented middleware allowing spontaneous distributed service composition to occur at runtime.

Key words: Pervasive Computing, Home, Service Oriented Computing, Distributed Applications, OSGi.

1. Introduction

Mark Weiser [8] brought a great paradox more than a decade ago: Computers have to be numerous and interconnected in order to disappear from the user's awareness. The purpose of pervasing computing is to realize this vision of increasingly ubiquitous network-enable devices. Specifically, it aims at filling our environment with communication devices in order to assist us in our daily activities without our explicit intervention. Indeed, in the near future, human beings will engage interactions with a number of smart devices, faded in into the environment, without being aware of their location or their precise nature and without going through complex, specific interfaces. They will simply express their needs or desires and the environment and the objects in it will configure themselves autonomously.

It seems that we are presently in a transitioning phase. The number and variety of smart communication devices are exploding: PDAs, smartphones, set-top-boxes, cameras, and electronic appliances can be found in many houses today. As envisioned by Moore's law, these devices are getting cheaper, smaller and are pervading every aspect of our life. The problem is that this invasion is chaotic: devices use a number of communication protocols and are rarely interoperable. Standards providing interoperation between access and indoor networks multiply: today, more than 50 candidate protocols, working groups and standard specifications for home networking already exist (see www.caba.org for an updated list). As a consequence, building consistent context-aware applications based on network enabled devices that spontaneously enter and leave the network turns is a real challenge.

Service-oriented computing (SOC) appears as a very promising paradigm to deal with the inherent complexity and dynamism of pervasive computing. Loosely coupled service architectures provide the level of flexibility required to build pervasive applications. However, SOC is today essentially technology-driven: most available platforms focus on the technology allowing developers to publish and compose services and to make them communicate. These platforms are usually not interoperable. In addition, programming dynamic service composition is a critical task that remains painful and error-prone since dynamic service availability is not addressed by current distributed middleware.

The purpose of this paper is to present a declarative model to automate the mutual discovery and binding of services running on distributed and heterogeneous network nodes (or devices). A middleware hiding the multiplicity of service discovery and distant communication protocols is built according to this model. This work has been carried out in the PISE[1] project. The paper is organized as follows. The global architecture and our approach are presented in section 2 and 3. Section 4 shows an implemented application. Section 5 compares our approach to related work. Finally, section 6 concludes and discusses perspectives.

[1] PISE is partially supported by the French Ministry of Industry under the RNRT program. It is conducted by Schneider Electric in collaboration with France Telecom, Trialog, INRIA and the LSR laboratory.

2. Architecture

We are working on a platform-centric, service-oriented architecture. In our view, the home infrastructure is composed of multiple platforms and devices that can be seen as service providers and users. A service platform is an open computing framework running user-related services. Such platforms can be an Internet gateway, a media center, a set-top-box or a home control gateway. Devices are pervasive elements integrated in the environments. Such devices include sensors, remote I/O, screens, loud speakers, lights or HVAC (Heating, Ventilation, Air Condition).

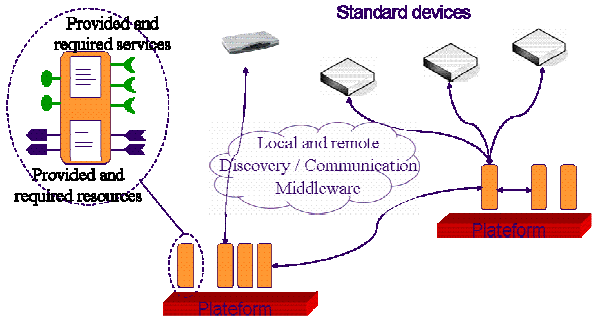

Figure 1 Platform-centric architecture

Many devices are today implemented as service providers (and sometimes requesters). Devices do not necessarily communicate with each other (it generally depends on their communication protocols and ability to understand each other). The smaller devices, in terms of computing resources, communicate through *ad-hoc* wired or wireless protocols. Proxies have to be created on a service platform to make them visible as services.

We consider devices as legacy components of our architecture, and focus in the following on offering a service platform that facilitates the development and deployment of applications targeted to this architecture.

3. Seamless integration of distributed services

3.1 Our approach

Our purpose is to provide a service platform hiding the heterogeneity of the distributed service technologies to the application developers.

To do so, we have been working on top of the OSGi service platform standard (see www.osgi.org). This open service platform defines a minimal deployment model and a set of standard services. The framework defines mechanisms that facilitate the dynamic installation, activation, deactivation, update, and removal of deployment units called bundles. Declarative Services specification has been included in the fourth release in order to automate service binding. This specification defines a component model allowing developers to specify component dependencies (in terms of provided and requested services) in an XML file. The specification is implemented as a bundle (Service Component Runtime - SCR) which manages bindings and service registrations.

In the Declarative Services specification, automatic management of service availability is devoted to OSGi services. We propose an extension to support the seamless integration of different heterogeneous and distributed service technologies.

Although services technologies like Jini (www.jini.org) or UPnP (www.upnp.org) are handled by the OSGi specification, using them it is not as transparent as it could be. For instance, UPnP Device Service Specification provides APIs for interoperability between OSGi services and UPnP devices. Unfortunately, these APIs come in addition to OSGi core APIs to discover and bind services for developers.

Generally speaking, in order to deal with heterogeneous and distributed service technologies, several issues have to be tackled:

- A common service description syntax has to be defined (and the mappings between this syntax and existing protocols have to be implemented).
- Common service discovery mechanisms have to be defined (and the mappings between these mechanisms and existing Service Discovery Protocols have to be implemented).
- Tedious stub code has to be statically or dynamically generated.
- Generic service events have to be defined in order to maintain dynamics.

As previously introduced, our reference architecture is the OSGi service platform. To be consistent with OSGi services, services are described by a set of Java interfaces associated with properties.

3.2 Extended SCR

As specified in the declarative services, service dependencies are described in an XML file. A service dependency contains information like the requested service query with cardinality and optionality. We have added an attribute called « registry » allowing the developer to specify whether provided services are available only on the local platform or can be made available for remote use and, symmetrically, whether a required service must be searched only on the local

platform or on the network. The value « registry » can be set to « local », to one or several service discovery technologies (including slp, upnp, uddi, jini, corba), or to "*", indicating that every available discovery protocol should be used. In the following examples, a requester and a provider are described.

```
<!-- A component requiring a service -->
<component name="service.requester">
<implementation class="pack.ControlPoint"/>
   <reference name="PLAYER"
     interface="api.Player"
     target="(hifi=true)"
     cardinality="1..n"
     policy="dynamic"
     registry="*"
     bind="bindPlayer"
     unbind="unbindPlayer"/>
</component>
```

```
<!-- A component providing a service -->
<component name="service.implementation">
<implementation class="pack.PlayerImpl"/>
 <service>
    <provide interface="api.Player"
      registry="upnp, corba"/>
 </service>
</component>
```

3.3 Lookup and binding services

The extended SCR appears to developers as a specific container transparently managing distributed service availability. The extension relies on "lookup services" and "export and binding factories" concepts not directly visible to the final application developer.

In our model, a lookup service is an OSGi service that allows remote service providers to register and remove service references and remote service requesters to request available services. A lookup service provides the following methods:

- The "register" method registers a service encapsulating all the information attached to a service provider (interfaces, localisation and attributes).
- The "unregister" method unregisters a service.
- The "lookup" method is the search method in the matching repository. The arguments are the interface of the required service and an LDAP filter.
- The "addListener" method enables an object to be aware of remote service events. OSGi eventing is used to deal with distribution. The listener object is notified of every arrival, modification and departure of local and remote services.

Remote binding relies on the export-binding pattern defined in ODP [6]. Two Java interfaces represent 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 an end point reference that can be published as part of a service reference. Symmetrically, a binding factory is used on the client side to transform an end point reference into a proxy allowing interaction with the remote service.

These abstractions (lookup, binding factory, export factory) allow application developers to ignore details of the distribution mechanisms (interaction and service discovery protocols). Moreover, protocol adapters are OSGi services packed in OSGi bundles, so that the choice of the protocols really used by the extended SCR can be decided at deployment time.

At present time, mappings and stubs have been implemented to seamlessly integrate UPnP and SLP [5] associated with RMI. We plan to address other protocols like Jini, Web Services, Bonjour/Zeroconf (www.zeroconf.org), CORBA (www.omg.org).

4 An example of pervasive application

In "follow-me" applications (Figure 2), a service follows the user as he/she moves by borrowing interfaces from available devices in the user's vicinity.

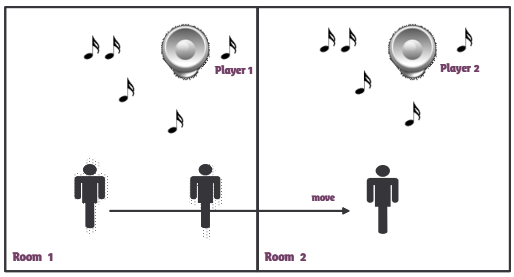

Figure 2 Audio stream follow-me: when the user moves from room 1 to room 2, Player 2 starts playing the audio stream. After a while, Player 1 stops playing

Our framework has been applied to build an "audio streaming follow-me" service. Distributed entities are depicted in Figure 3. Streaming source may be an Internet site or a UPnP Media Server. Players are UPnP Media Renderers and devices with ad hoc SLP interfaces. Yellow pages listing all the known streaming resource may be a UPnP Media Server. A smart service composer using the extended SCR is available as a remote control on a PDA, the control point is on a Home Gateway and the location server is installed on aPC.

The developer of the control point defines two bind methods (bindPlayer and unbindPlayer). As the user moves across rooms, the extended SCR discovers which players are available on the network, and automatically calls the bindPlayer and unbindPlayer when necessary.

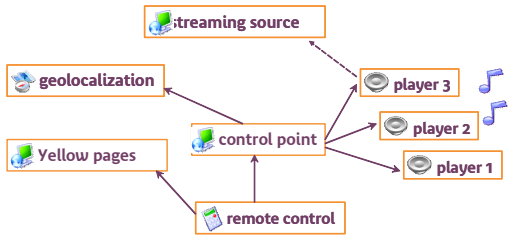

Figure 3 Architecture of the follow-me application. The plain arrows represent established bindings.

5. Related work

Several service–oriented middleware technologies are standardized and available today: UPnP, Jini, Web Services, DPWS, Bonjour, SLP, CORBA, etc. These standards define protocols and mechanisms for service discovery and interaction. However they do not address service binding automation.

Important works are carried out above Web Services. They leverage service discovery with directories like UDDI and service orchestration (BPEL) in order to compose services in a workflow model [1]. The notion of workflow and the target of the multienterprise domain place these works out of the scope of this paper.

Our approach is close to REMMOC's [3], in that it allows protocol adapters to be plugged in at deployment time and hides protocol heterogeneity to SOA developers. However, service binding automation is not addressed by the REMMOC project.

MSD [7] enables interoperability between legacy service clients and providers developed with specific protocols. The MSD architecture acts as a bridge between topologic networks and between discovery technologies. It does not address distribution mechanisms transparency for service development and it does not address automated composition.

6. Conclusion

Our architecture allows developers to focus on functional code. It relieves application developers from the tedious and error-prone programming tasks related to distribution and dynamic service availability.

It also offers an extensible framework where new communication or new service discovery protocols can be added by writing the corresponding export/binding factories or lookup services and packing this software into OSGi bundles. Another pluggable architecture keeping Declarative Services container and Distribution containers separate is probably possible. It can be address in future work to improve separation of concerns, here Dynamic Service availability and Distribution.

Thanks to this architecture, peer-to-peer composition occurs at runtime. Every component declares its own provided and required services. Our framework automatically discovers and maintains the list of available requested services, publishes and maintains the registration of provided services, dynamically links the adequate components through the best possible protocol.

Future work will add contextual selection automation into the architecture. Semantic approaches are relevant for context-aware service composition. Future work will reuse an existing location ontology and define specific application context ontology.

References

1 F. Casati, Ski Ilnicki, Jin Li-Jie, V. Krishnamoorthy, Shan Fabio Casati, Ski Ilnicki, Jin Li-Jie, Vasudev Krishnamoorthy, Shan Ming-Chien, "eFlow: a platform for developing and managing composite e-services", Working Conference on Research Challenges, April 2000

2 Humberto Cervantes, Richard S. Hall, "Automating Service Dependency Management in a Service-Oriented Component Model", Workshop on Component Based Software Engineering, May 2003

3 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, November 2003

4 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, 2004

5 Erik Guttman, Charles Perkins, John Veizades, Michael Day, "Service Location Protocol, Version 2", RFC 2608, June 1999

6 ITU-T & ISO/IEC, "ODP Reference Model: Overview, Foundations, Architecture", Recommendations X.901, X902, X903 & International Standards 10746-1, 10746-2, 10746-3, 1995

7 Pierre-Guillaume Raverdy, Valérie Issarny, "Context-aware Service Discovery in Heterogeneous Networks", IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, June 2005

8 Mark Weiser, "The computer for the 21st century", Scientific American, 265(3):66-75, September 1991