OSGi et le projet IST Amigo

Anne Gérodolle, André Bottaro

OSGi et le projet IST Amigo

Anne Gérodolle
France Telecom RD
28 chemin du Vieux Chêne - BP 98
38243 Meylan CEDEX
FRANCE
anne.gerodolle@orange-ft.com

André Bottaro
France Telecom RD
28 chemin du Vieux Chêne - BP 98
38243 Meylan CEDEX
FRANCE
andre.bottaro@orange-ft.com

RESUME
Le projet IST Amigo (Ambient Intelligence for the networked home environment) a pour ambition la promotion des réseaux IP domestiques, d'une part par la construction de scénarios exploitant ces réseaux, d'autre part par le travail sur une architecture logicielle. Cette architecture vise à faciliter la mise en relation spontanée d'objets communicants pour offrir des services de haut niveau aux utilisateurs en prenant en compte le contexte d'utilisation.

Le travail présenté ici se situe dans ce deuxième axe, et décrit l'infrastructure basée sur OSGi utilisée dans ce projet pour le développement et le déploiement de composants Java, et les moyens mis en œuvre pour permettre la découverte et l'interaction entre des équipements ou logiciels connectés à un réseau domestique.

Categories
D.2.12 [**SOFTWARE ENGINEERING**]: Interoperability – # *Data mapping, Distributed objects, Interface definition languages.*

General Terms
Design

Keywords
OSGi, Web Services, WS-discovery, WS-eventing

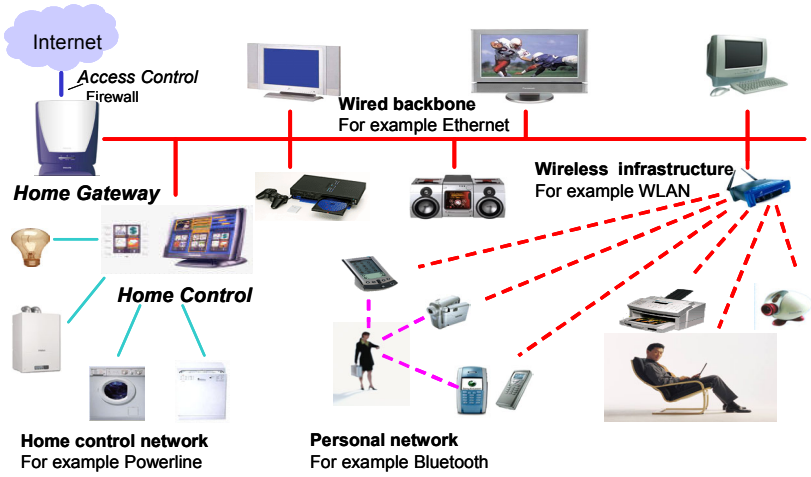
1. INTRODUCTION
Le projet IST Amigo [1] a pour ambition la promotion des réseaux IP domestiques, d'une part par la construction de scénarios exploitant ces réseaux, d'autre part le travail sur une architecture logicielle. Cette architecture vise à faciliter la mise en relation spontanée d'objets communicants pour offrir des services de haut niveau aux utilisateurs en prenant en compte le contexte d'utilisation.

Un « réseau Amigo » intègre à la fois des éléments matériels et logiciels sur étagère (un boîtier « UPnP AV Media Renderer », une machine à laver utilisant un bus courant porteur, etc.) et des composants logiciels « amigo-aware », qui vont des services d'infrastructure aux services offerts à l'utilisateur. Parmi les services d'infrastructure citons : le service d'interopérabilité, qui agit en médiateur permettant l'interaction entre des matériels ou logiciels sur étagère utilisant des protocoles et des modèles hétérogènes, service de stockage, de sécurité, de gestion du contexte, de notification. Notons que tout composant logiciel peut à la fois offrir des services et utiliser des services proposés par d'autres composants ; de même, tout composant peut être à la fois source d'événement et récepteur d'événements fournis par d'autres sources.

Les composants sur étagère imposent des protocoles réseaux et des modèles d'interaction hétérogènes. Une des premières tâches dans le projet a été de définir l'architecture d'un middleware permettant l'interopérabilité entre différents protocoles pour la découverte et l'interaction [2]. Toutefois, pour la communication entre les composants logiciels développés en interne au projet Amigo, il a été décidé d'utiliser des protocoles homogènes. En l'occurrence le choix s'est porté sur les technologies Web Services : le protocole WS-discovery [3] permet à un programme client de découvrir les services présents sur le réseau, le protocole HTTP/SOAP est utilisé pour les interactions synchrones et le protocole WS-eventing [4] pour la souscription à des sources d'événements.

Amigo n'impose a priori aucun langage de programmation ni aucun modèle de déploiement, et le partenaire développant un composant Amigo peut utiliser le langage de programmation de son choix, à condition d'utiliser les protocoles retenus par le projet pour la découverte et l'interaction. Cependant, l'utilisation d'une méthode de développement et d'outils communs limite les risques de divergence d'interprétation des protocoles et rend moins problématique l'intégration de composants développés indépendamment dans une même application. De plus, l'utilisation d'une plate-forme de déploiement comme OSGi ou .Net permet un partage des ressources appréciable (non seulement partage des bibliothèques, mais aussi partage par exemple d'un même serveur Web). Ces considérations pragmatiques nous ont menés à proposer deux environnements visant à faciliter le développement et le déploiement de composants Amigo : Microsoft, partenaire du projet, propose un environnement de développement permettant de développer en C# des composants destinés à être déployés sur une plate-forme .Net. Pour les composants logiciels Java, la cible de déploiement est la plate-forme OSGi. Nous présentons ici l'infrastructure basée sur OSGi.

Ce document est organisé de la façon suivante : après une brève présentation des protocoles utilisés dans Amigo, nous exposons les contraintes et objectifs de notre travail, puis les principes d'architecture retenus. Nous décrivons les composants logiciels disponibles (bundles) et leur utilisation. Enfin, nous terminons par une présentation des travaux en cours.



4

Figure 1 Un réseau Amigo intègre des composants hétérogènes qui communiquent avec des protocoles différents

2. Les protocoles utilisés

2.1 WS-discovery

La mise en relation « spontanée » de logiciels ou de services nécessite un système de découverte permettant aux « clients » de découvrir les « serveurs ». Ce problème est adressé par plusieurs protocoles, qui ont en commun l'utilisation d'une adresse et d'un port multicast réservés – par exemple 239.255.255.253, 427 pour le protocole SLP normalisé à l'IETF, 239.255.255.250, 1900 pour le protocole SSDP utilisé par les équipements UPnP, 239.255.255.250, 3702 pour WS-discovery, etc.. On distingue en général la découverte active (les clients adressent des requêtes sur le groupe multicast, les serveurs répondent en décrivant les services correspondant à la requête) et la découverte passive (les serveurs annoncent sur le groupe multicast leur arrivée et leur départ). Les protocoles prévoient souvent l'utilisation combinée de ces deux modes de découverte, et peuvent pour des raisons de passage à l'échelle prévoir la présence d'agents spécialisés limitant le nombre de requêtes multicast. Il n'est pas dans l'objectif de ce papier de présenter et comparer ces différents systèmes, on pourra pour cela se référer à [5] [6].

Le protocole WS-discovery choisi dans Amigo est une proposition d'un consortium réunissant Microsoft, BEA, Canon, Intel et Webmethods. Ce protocole s'appuie sur les standards du W3C (xml, soap, WS-addressing, etc…) pour définir un protocole de découverte sur un réseau local. La spécification prévoit deux modes de fonctionnement, selon la présence ou non d'un « Discovery Proxy » sur le réseau. Les messages envoyés sont formatés dans des enveloppes SOAP. Les premières spécifications datent d'octobre 2004. Une mise à jour a été proposée en avril 2005. Il n'existait à notre connaissance aucune implémentation en Java de ce protocole.

2.2 WS-eventing

WS-eventing est une proposition soumise en mars 2006 au consortium W3C, qui s'intéresse essentiellement aux mécanismes de souscription à des sources d'événements. WS-eventing adresse la même problématique que WS-Base_notification proposé par OASIS.

.WS-discovery et WS-eventing sont utilisés dans la norme en cours d'élaboration DPWS (Device Profiles for Web Services)

3. L'infrastructure OSGi Amigo

3.1 Objectifs

Le but de notre infrastructure est de faciliter le développement et le déploiement de composants qui exposent ou utilisent des services sur un réseau local. Cette infrastructure devait respecter plusieurs contraintes :

- permettre le déploiement sur des environnements « contraints » (les contraintes ne sont pas ici quantifiées, le fonctionnement vise un environnement J2ME / personal profile).
- permettre la publication et la découverte de services sur un réseau local en utilisant le protocole WS-discovery, de façon interopérable avec l'environnement .Net Amigo.
- permettre l'interaction synchrone (via le protocole HTTP/SOAP) entre clients et services, de façon interopérable avec l'environnement .Net Amigo.
- permettre l'écriture de sources d'événements et de consommateurs d'événements en utilisant le protocole WS-eventing, les sources et consommateurs d'événements appartenant indifféremment aux infrastructures OSGi Amigo ou .Net Amigo.

3.2 Architecture logicielle

Nous avons choisi d'ajouter une contrainte supplémentaire pour éviter de lier le programmeur utilisant notre infrastructure à un jeu de protocoles déterminés : ainsi, nous avons défini des abstractions représentant la notion de système de découverte, d'usine à liaison, de gestionnaire de souscriptions, etc. [8]. La liaison entre ces abstractions et les classes d'implémentation est faite à l'exécution, par le biais du répertoire de services OSGi.

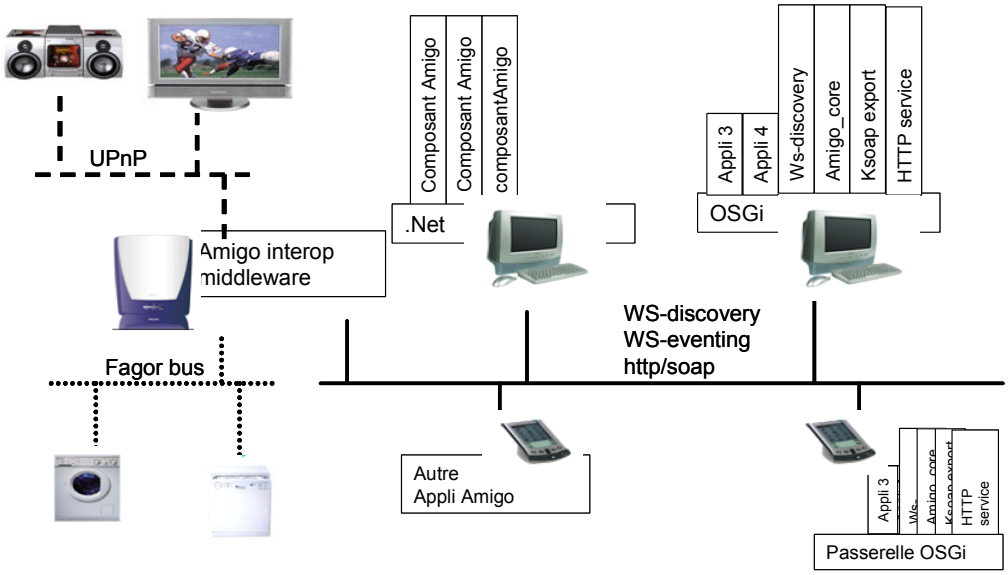


Figure 2. Les composants développés pour Amigo en utilisant l'environnement .Net/C#, OSGi/Java ou de façon indépendante interagissent au travers des protocols WS-*. Une passerelle Amigo permet l'interaction avec des logiciels et matériels sur étagère utilisant d'autres protocoles

Notre infrastructure se traduit par la fourniture de bundles OSGi :

- bundle « amigo_core » qui définit les abstractions mais n'est lié à aucun protocole particulier. C'est le seul bundle qui est utilisé au moment du développement. Il exporte les packages contenant les interfaces de base et doit être déployé sur toutes les plates-formes utilisant ou exposant des services.
- bundle « WS-discovery » qui fournit l'implémentation de l'abstraction « AmigoLookup » (lookup service) défini dans amigo_core mais n'exporte aucun package. Il doit être déployé sur les plates-formes OSGi qui publient ou recherchent des Web services.
- bundle « ksoap-binding » qui fournit l'implémentation des « usines à liaison » (binding factory) permettant à des clients de fabriquer des « stubs » permettant la communication avec des services distants : appel synchrone de méthodes, souscription à des sources d'événements. Il doit être déployé sur les plates-formes OSGi qui hébergent des Web services ou des clients de Web services.
- bundle « ksoap-export » qui permet à un bundle d'exposer des services sur le réseau ou se comporter comme source d'événements. Ce bundle fournit une implémentation de la classe ExportFactory définie dans amigo_core et s'appuie sur le service OSGi standard « http service ». Il doit être déployé sur les plates-formes OSGi qui hébergent des Web services.

Nous avons utilisé la bibliothèque ksoap [9] car elle nous paraissait plus légère à mettre en œuvre sur des plates-formes limitées (profil J2ME/personal profile) que la bibliothèque Axis [10], en particulier pour l'exposition de services Web. Cependant Axis est plus complet lorsqu'il s'agit par exemple de manipuler des types d'arguments ou de retours complexes. Des bundles d'adaptation utilisant la bibliothèque Axis sont donc en cours de développement. Ils pourront être déployés en lieu et place des bundles ksoap, sans donner lieu à modification des applications utilisant l'infrastructure.

3.3 Exemple d'utilisation de l'API Amigo :

Publication d'un service sur le réseau. Dans cet exemple, le programmeur souhaite rendre visible sur le réseau l'objet library, qui implémente la classe test.Library.

```
Library library = new LibraryImpl(…);
ServiceReference ref =
context.getServiceReference(AMIGO_EXPORTER);
AmigoServiceExporter exporter =
(AmigoServiceExporter) context.getService(ref);
AmigoExportedService service =
exporter.createService(Library.class, library);
ServiceReference ref=
context.getServiceReference(AMIGO_LOOKUP);
AmigoLookup lookup= (AmigoLookup)
context.getService(ref);
service.addProperty("ServiceType", "Bookshop");
service.addProperty("Scope", "urn:amigo");
lookup.register(service);
```

Recherche d'un service sur le réseau et utilisation. Dans cet exemple, le client recherche un service déclaré comme « Bookshop » et invoque la méthode getBooks.

```
ServiceReference ref =
context.getServiceReference(AMIGO_LOOKUP);
AmigoLookup lookup = (AmigoLookup)
context.getService(ref);
AmigoService service =
lookup.lookupFirstService("urn:amigo","Bookshop");
Library stub =
amigoService.getSpecificStub(Library.class);
Book[] books = stub.getBooks(…);
```

3.4 Interchangeabilité des protocoles

La première version de l'infrastructure a été livrée en février 2006. L'implémentation de WS-discovery en Java n'était pas encore disponible, et nous avons alors fourni une implémentation de AmigoLookup utilisant le protocole SLP[11]. Cela a permis aux partenaires de commencer leurs développements et leurs tests en utilisant le protocole SLP. Ils ne pouvaient toutefois pas découvrir ou être découverts par les services développés pour la plate-forme .Net, qui utilisait WS-discovery. Lors de la fourniture du bundle WS-discovery, les bundles déjà développés en utilisant l'infrastructure ont pu alors utiliser le nouveau protocole sans modification. Il a suffi de déployer le bundle « WS-discovery lookup » en lieu et place du bundle « SLP lookup ».

Cependant, cette méthode a des limites. Par exemple, concernant les protocoles de découverte, les capacités d'expression de SLP et WS-discovery sont assez différentes : la notion de « scope » se résume à une simple chaîne de caractères dans SLP alors qu'elle est plus riche, et même extensible, dans WS-discovery. En revanche, SLP permet d'ajouter des propriétés arbitraires lors de l'enregistrement d'un service, et d'exprimer des filtres dans la syntaxe LDAP, comme le répertoire de services OSGi, ce que ne permet pas WS-discovery. Il est possible d'interchanger les deux protocoles sans modifier le code client à condition d'employer un dénominateur commun.

4. Déploiement

Ces bundles sont disponibles sur un « Amigo Bundle Repository » et peuvent être déployés en utilisant le bundle OBR[12]. Ce répertoire a pour vocation d'accueillir également les autres composants OSGi développés dans Amigo. Si nous appelons « profil Amigo » un sous-ensemble cohérent de ces bundles, il est envisageable que sur le même réseau cohabitent des passerelles OSGi présentant des profils Amigo différents. Par exemple des plates-formes « serveur » pourront accueillir un grand nombre de bundles (bundles d'infrastructure, serveur de contenu, moteur de raisonnement sur le contexte, etc.) alors que d'autres matériels plus contraints accueilleront un profil beaucoup plus minimal. Aujourd'hui, ce déploiement est fait « à la main » en utilisant les outils OBR. Un des axes de recherche dans le projet Amigo consiste à enrichir la description des bundles, pour permettre un déploiement automatisé obéissant à des critères sémantiques [13].

5. Perspectives

L' « Amigo Bundle Repository » est amené à être enrichi par les bundles de plus haut niveau développés par les différents partenaires. Le travail de recherche sur le déploiement sémantique cité plus haut devrait faciliter le déploiement de ces bundles sur les différentes plates-formes en fonction du contexte d'utilisation.

Nous avons également comme objectif d'intégrer dans Amigo la notion d'« Extended Service Binder » déjà présentée dans PISE[8][14]. La difficulté ici est qu'Amigo ne se situe pas dans un contexte purement Java. Dans un monde « OSGi réparti », les services requis et offerts sont décrits par une interface Java. Le répertoire de services OSGi peut être en quelque sorte étendu par un service de découverte comme SLP ou WS-discovery, il suffit de convenir d'une correspondance entre le monde Java et le protocole, par exemple dans le cas de WS-discovery en déclarant comme « ServiceType » le nom de l'interface Java. Cette contrainte forte n'est pas acceptable dans Amigo. Nous devons donc proposer un moyen de décrire cette correspondance entre interfaces Java et ServiceType.

Nos travaux portent également sur l'adaptation de services. En effet, les services présents sur un réseau Amigo peuvent exposer des interfaces sémantiquement proches et syntaxiquement différentes. Notre approche technique vise la construction dynamique de « smart stubs » implémentant une interface requise et s'appuyant sur un service proposant une interface fournie « peu différente ». Ces « smarts stubs » pourraient être utilisés par notre « extended service binder » de façon totalement transparente pour le programmeur d'application.

6. REMERCIEMENTS

Ce travail a été effectué dans le cadre du projet européen Amigo - IST 004182.

7. REFERENCES

[1] Le projet IST Amigo http://www.hitech-projects.com/euprojects/amigo/index.htm

[2] Sacchetti,D., Bromberg, Y.-D., Georgantas, N. , Issarny, V., Parra,,J. and Poortinga, R. *The Amigo Interoperable Middleware for the Networked Home Environment*. Demonstration paper. In Middleware'2005 CD.

[3] Spécifications du protocole WS-discovery. http://schemas.xmlsoap.org/ws/2005/04/discovery/

[4] Spécifications du protocole WS-eventing , http://schemas.xmlsoap.org/ws/2004/08/eventing/

[5] Bettstetter , C.and Renner, C. *A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol*. Proc. EUNICE Open European Summer School, Twente, Netherlands, Sept 13-15, 2000

[6] Marin-Perianu, R. and Hartel, P.H. and Scholten, J. (2005) *A Classification of Service Discovery Protocols*. Technical Report TR-CTIT-05-25 Centre for Telematics and Information Technology, U.Twente, Enschede. ISSN 1381-3625

[7] Jammes F., Mensch A., Smit H., *Service-Oriented Device Communications Using the Devices Profile for Web Services*, Proceedings of the 3rd Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing (MPAC05), 2005

[8] Bottaro, A., Gérodolle, A. et Lalanda, P., *Pervasive Spontaneous Composition*, First IEEE International Workshop on Service Integration in Pervasive Environments, Lyon, France, 2006.

[9] Le projet ksoap, http://ksoap2.sourceforge.net/

[10] Le projet Axis, http://ws.apache.org/axis/

[11] Service Location Protocol, http://tools.ietf.org/html/2608

[12] OBR http://oscar-osgi.sourceforge.net/

[13] Le Mouël, F., Ibrahim, N., Royon, Y., Frénot, S., *Semantic Deployment of Services in Pervasive Environments*, Proc. 1st Int. Workshop on Requirements and Solutions for Pervasive Software Infrastructures (RSPSI'2006) in conjunction with the Pervasive'2006 Conference, Dublin, Ireland, May 2006.

[14] Le projet RNRT PISE, http://www.rnrt.org/rnrt/projets/PISE.htm