Home SOA : Composition contextuelle de Services dans les Réseaux d'Equipements pervasifs

André Bottaro

UNIVERSITE JOSEPH FOURIER – GRENOBLE I

THESE

Pour obtenir le grade de

DOCTEUR de l'Université JOSEPH FOURIER

Discipline : INFORMATIQUE

ECOLE DOCTORALE Mathématiques, Sciences et Technologies de l'Information, Informatique

Présentée et soutenue publiquement par

ANDRE BOTTARO

Le 12 décembre 2008

HOME SOA : COMPOSITION CONTEXTUELLE DE SERVICES DANS LES RESEAUX D'EQUIPEMENTS PERVASIFS

Directeur de thèse :

Philippe LALANDA

JURY

Président	Andrzej DUDA, Professeur à l'Institut National Polytechnique de Grenoble
Rapporteurs	Valérie ISSARNY, Directeur de Recherche à l'INRIA Lionel SEINTURIER, Professeur à l'Université de Lille
Examinateurs	Didier PELLEGRIN, Chef de Projet, Schneider Electric Vincent OLIVE, Responsable d'unité, Orange Labs
Encadrant	Philippe LALANDA, Professeur à l'Université Joseph Fourier

Thèse préparée au sein des Orange Labs et du Laboratoire d'Informatique de Grenoble

Remerciements

Je remercie les membres du jury, en particulier Lionel Seinturier et Valérie Issarny qui ont évalué mon mémoire avec la profondeur de leur expérience dans leurs rapports respectifs. Je remercie également Andrzej Duda d'avoir accepté de présider ce jury, ainsi que Didier Pellegrin et Vincent Olive en tant qu'examinateurs de cette thèse.

Je n'oublierai pas la pédagogie précieuse que m'a dispensé Philippe Lalanda, mon directeur de thèse à l'Université Joseph Fourier et mon mentor depuis mon premier stage à Schneider Electric jusqu'à l'aboutissement de cette thèse à France Télécom.

Merci à Richard S. Hall pour s'être proposé comme second encadrant en début de thèse et avoir mis mes travaux sur des rails techniques concrets durant l'année 2006.

Je reconnais la présence continue de Vincent Olive, mon responsable d'unité à France Télécom. Vincent m'a proposé les projets et la liberté nécessaires à mon épanouissement.

Ce travail s'est déroulé au sein du laboratoire intergiciel des Orange Labs du Groupe France Telecom et de l'équipe Adèle du Laboratoire d'Informatique de Grenoble. Je tiens à remercier généralement les membres de ces deux équipes pour le cadre de travail stimulant offert et à remercier quelques personnes particulières : Merci à Didier Donsez qui m'a poussé dans une candidature de thèse le premier, à Alexandre Avenel, François-Gaël Ottogalli et Roland Airiau qui m'ont fourni un cadre de projets propice à la communication de nouvelles techniques au sein des Orange Labs, à Johann Bourcier, Clément Escoffier, Mourad Alia qui ont élaboré avec moi des prototypes pour des articles à des conférences de haut niveau, à Aimé Vareille avec qui j'ai partagé mon bureau et son ouverture à des domaines vastes touchant peu ou prou le métier de France Telecom, à Serge Martin, Régine Basset, Marc Lacoste pour la représentation de nos travaux dans des projets collaboratifs.

La compréhension de mes motivations par Kathleen Milsted et Alain Chemarin, responsables successifs du laboratoire Orange, a été essentielle pour la menée à bien de ce travail de longue haleine. Jacky Estublier, directeur de recherche de l'équipe Adèle, a aussi saisi l'adéquation de ma motivation et des objectifs de cette thèse en parallèle.

Une section particulière à Anne Gérodolle et Levent Gürgen avec qui j'ai travaillé en binôme durant de longues périodes. Sans les échanges techniques continus avec Anne et l'interchangeabilité atteinte dans nos rôles d'encadrant, le cadriciel Home SOA n'aurait pu devenir ce recueil d'expériences et cette base de code aisément partageable. Sans la motivation et la contribution complémentaire de Levent au Comité UPnP Device Management, France Télécom n'aurait pu paraître aussi présent auprès de nos partenaires.

La taille et la qualité des prototypes réalisés sont le fruit d'une équipe de développement rassemblée au gré des opportunités administratives, financières et humaines saisies au cours de ces années. Mes vifs remerciements aux concepteurs-développeurs ayant contribué durant une période de prestation, de stage ou d'apprentissage à la réalisation du cadriciel Home SOA : Gregory Bonnardel, Fatoumata Camara, Maxime Vincent, Vincent Moreau, Fabrice Jaumes, Teva Paoletti, Nicolas Peru, Xavier Goblet, Eric Simon, Stéphane Seyvoz, Marius Tankeu, Judex MBoulou, Nicolas Chabanoles et Julien Ravoire.

J'adresse enfin une pensée chaleureuse aux membres de ma famille, mes parents, mes frères et mes beaux-parents, ainsi qu'à mes amis qui m'ont toujours soutenu.

Une pensée émue à ma femme Sophie qui m'a accompagné courageusement dans cette thèse. Merci de partager ma vie et d'être attentive à son équilibre. Ces années de thèse ont été par ailleurs les premières années de Lison et Anatole, sources de bonheur et de fierté.

Résumé

Les équipements électroniques envahissent progressivement l'univers quotidien. D'aucuns souhaiteraient que ces équipements puissent intelligemment réagir à l'activité de l'utilisateur afin de l'assister dans ces activités de tous les jours. Le domaine de l'Informatique Pervasive adresse la vision de ce monde naturellement numérique assistant les individus sans être intrusif.

Face aux défis de l'Informatique Pervasive dans les réseaux locaux, notamment la distribution, l'hétérogénéité et la dynamique des équipements, cette thèse répond par une ligne de conduite et l'approche logicielle Home SOA. Cette ligne de conduite distingue les situations où les solutions protocolaires sont pertinentes et ramène les autres situations à des problèmes de génie logiciel. Parmi les solutions protocolaires, la proposition d'une interface uniforme de gestion de cycle de vie logiciel dans le Comité de Travail UPnP Device Management est une des contributions importantes.

Le Home SOA est l'association de technologies de développement modulaire et d'un ensemble de patrons de conception orientés objets. Au-delà de l'orientation objet, le Home SOA exploite les modèles récents de composants à services et le concept de plateforme de services. Les pilotes orientés service masquent les aspects distribués tout en réifiant la dynamique des entités pervasives sur la plateforme. Les pilotes raffinés adaptent les objets mandataires dans des interfaces à la sémantique du domaine d'application visé. La contextualisation des services de la plateforme alliée à l'automatisation de la sélection de service achève de simplifier le développement d'applications. Le cadriciel est implémenté au-dessus de la plateforme OSGi et est validé par la réalisation d'applications conscientes du contexte et mixant des domaines d'applications distincts dans le réseau domestique.

Abstract

Home SOA: Contextual Service Composition In Pervasive Device Networks.

Pervasive Computing 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.

Addressing Pervasive Computing challenges in local networks such as device distribution, heterogeneity and dynamicity, this thesis exposes a conduct line and a software approach named Home SOA. This conduct line distinguishes the situations where protocol-based solutions are relevant and turn the other situations into Software Engineering problems. Among the protocol-based solutions, the proposition of a uniform protocol interface for software lifecycle management in the UPnP Device Management Working Committee is one of the major contributions.

The Home SOA promotes a set of object-oriented design patterns above modular software platform technologies. Beyond the object orientation, the Home SOA makes benefit of recent service component models and the service platform concept. Distribution is hidden by service-oriented drivers while network dynamicity is locally reified on the platform. Distributed objects are turned by refined drivers into interfaces semantically close to the targeted application domain. Service contextualization and service selection automation reach the objective of simplifying the development or pervasive applications. The proposed framework is implemented on the OSGi platform and is validated by the realization of context-aware applications mixing distinct application domains in the Home network.

Table des Matières

Table des Figures

Table des Tableaux

Chapitre I. INTRODUCTION

1 CONTEXTE

Les équipements électroniques envahissent progressivement l'univers quotidien. D'aucuns souhaiteraient que ces équipements puissent intelligemment réagir à l'activité de l'utilisateur afin de l'assister dans ces activités de tous les jours. Le domaine de l'Informatique Pervasive adresse la vision de ce monde naturellement numérique assistant les individus sans être intrusif. Cette vision porte néanmoins un paradoxe important à la base du travail de chercheurs dans le monde entier : les ordinateurs doivent être nombreux et interconnectés afin de disparaître de la conscience de l'utilisateur. L'envahissement des capacités de calcul et de communication dans l'environnement apporte l'intelligence nécessaire à l'assistance des individus en toute situation et en tout lieu tandis que les techniques de raisonnement sur les activités des individus garantissent la liberté souhaitée universellement. L'environnement quotidien doit devenir l'interface naturelle d'interaction avec l'utilisateur.

L'organisation des équipements électroniques dans l'environnement quotidien demande un effort de recherche et de développement dans de nombreux domaines. La vision de l'Informatique Pervasive ne s'exaucera que par la conjonction de progrès dans divers domaines des sciences dures et humaines. L'augmentation des capacités des équipements électroniques demande un effort dans les domaines de l'électronique, la physique des matériaux, le traitement du signal, le génie logiciel. L'adaptation de ces techniques pour leur usage réel nécessite un travail approfondi dans l'ergonomie, l'étude des usages.

De la répartition des équipements électroniques dans l'environnement doit émerger l'intelligence des applications. L'auto-organisation de différents éléments logiciels répartis est un défi posé au génie logiciel. Coordonner les processus répartis sur plusieurs machines différentes emprunte les techniques de l'informatique distribuée. Aussi, la réaction dynamique au contexte de l'utilisateur implique un lien à des systèmes d'inférence sur la remontée de données de capteurs. Ensuite, l'ouverture de l'environnement à des équipements de natures hétérogènes est une forte demande d'évolution des applications. Encore, la confidentialité et la sécurité des données du système est une autre fonctionnalité à garantir. Enfin, embarquer des applications aussi complètes dans des équipements contraints en ressources achève de rendre complexe la tache des ingénieurs logiciels.

L'Informatique Pervasive fait de l'imbrication logicielle un domaine de recherche scientifique.

2 PROBLÉMATIQUE

L'ouverture de l'environnement à des entités électroniques distinctes et variées dans un contexte évoluant rapidement est la caractéristique des environnements pervasifs. L'organisation de ces entités distinctes en un réseau cohérent pose le premier défi de la répartition des équipements. Un système réparti coordonne les fonctions de plusieurs unités de calculs disposées en réseau afin de fournir les fonctionnalités agrégées d'une même application. Des techniques intergicielles permettent de masquer l'aspect distribué de fonctions distantes dans les langages de programmation. Masquer la communication distante à des entités programmatiques inconnues à l'avance demeure toutefois un défi de l'Informatique Pervasive.

L'hétérogénéité des entités intervenant dans l'environnement est le deuxième défi auquel cette thèse apporte des réponses. Les entités d'un environnement pervasif apparaissent avec leurs protocoles de communication standards ou propriétaires, des langages de programmation distincts et des modèles de données sémantiquement différents. Des techniques de médiation technique permettent d'unifier le contrôle d'entités aux protocoles distincts. Rendre cette couche de médiation technique extensible et capable d'évoluer dynamiquement demeure toutefois un défi. Automatiser la correspondance sémantique des interfaces de contrôle de ces entités est un défi dont les réponses actuelles, issues en particulier du Web Sémantique, semblent encore plus partielles.

La dynamique des environnements pervasifs est le troisième défi auquel sont confrontés les concepteurs d'applications. L'une des caractéristiques de l'Informatique Pervasive est l'exigence d'adaptation des applications à un contexte changeant : adapter les services offert à l'utilisateur en fonction de son activité et de sa localisation, adapter la demande de ressources selon les capacités et la disponibilité des équipements, enfin adapter la réponse du système en fonction des variables de l'espace environnant les équipements et les utilisateurs. L'adéquation entre ressources requises et fournies doit être établie et rétablie dynamiquement selon l'évolution des entités disponibles. La forte dynamique de l'environnement exacerbe la demande de flexibilité des applications et l'aspect lâche du couplage entre les entités impliquées. Par-delà cette flexibilité dont certaines réponses sont aujourd'hui offertes par de nouvelles approches logicielles comme l'orientation service, la sélection de la meilleure configuration parmi les configurations disponibles demeure un défi important pour les applications envisagées. De plus, cette sélection se doit d'être non ambiguë afin que l'utilisateur ne soit pas confronté à deux réponses différentes du système dans des situations identiques.

Le mémoire se concentre sur la réponse à ces trois défis posés par l'Informatique Pervasive. L'ouverture des environnements pervasifs à de nombreuses entités inconnues au préalable pose d'autres défis comme la sécurité. Si l'environnement est ouvert à toute entité inconnue, il se doit paradoxalement d'authentifier et de réglementer l'accès aux différentes parties. La sécurité est aussi liée au respect de l'intimité des individus. La remontée d'information nécessaire aux applications attentives au contexte exige des protections pour les informations prises sur les utilisateurs. Enfin, l'aspect embarqué des équipements accroît aussi la difficulté de la conception des applications innovantes visées. La demande de flexibilité des applications exige des plateformes d'exécution des capacités de reconfiguration applicative. Face au prix du nombre des équipements à répartir dans l'environnement, les concepteurs d'applications tentent un compromis entre des plateformes peu couteuses dont les contraintes interdisent l'évolution de l'application originale et des plateformes plus riches qui s'accordent avec l'extensibilité demandée à moyen terme par l'utilisation. Catégoriser les exigences techniques de flexibilité et mesurer leur impact sur le coût des applications est aussi un devoir pour les chercheurs et architectes du Génie Logiciel.

3 CONTRIBUTION

Cette thèse apporte des réponses aux différents défis posés par l'Informatique Pervasive. Masquer la distribution dans un environnement changeant a été le premier but recherché. Cette première étude a porté d'abord sur une comparaison de différents protocoles applicatifs de découverte et de communication répandus aujourd'hui : UPnP [UPnP06], IGRS [IGRS06], DPWS [DPWS06], Apple Bonjour/iTunes [StC05], Jini [Wal99], SLP [GPVD99], Salutation [SALU99]. Elle a débouché sur un premier prototype appelé Extended Service Binder [2006-4][2006-2] mêlant des techniques de l'Informatique Distribuée à un modèle de programmation à service. Tandis que la modularité du logiciel dessine le contour des différentes fonctions d'ensembles protocolaires (découverte, description, contrôle), la conception mixe cependant la gestion de la dynamique avec la gestion des aspects distribués dans une brique dont la scission est difficile. Ce prototype a fait l'objet d'un brevet [2005-1] et a été intégré aux démonstrations du projet RNRT Pise [2008-3][2006-6] et du projet IST Amigo [2006-5]. L'étude particulière d'IGRS a été demandée opportunément par un représentant de France Télécom au comité ISO/IEC face à la proposition de l'ensemble protocolaire en normalisation ISO/IEC par les IGRS Labs – www.igrs.org – en 2006. Elle a conduit à la demande de modifications [2006-7] acceptée par les IGRS Labs. La modification principale permet aux équipements UPnP et IGRS de coexister sur les réseaux locaux.

Les études suivantes se sont évertuées à scinder la gestion de la distribution et la gestion de la dynamique de l'application. L'approche orientée service demeure la base de la construction en raison de l'abstraction apportée dans la représentation des entités de l'environnement pervasif et du couplage lâche qu'elle permet entre ces différentes représentations. Elle est alors utilisée dans la définition et la programmation de pilotes – "drivers" – orientés services. Ces pilotes réagissent à l'apparition, la disparition, la modification des entités – souvent des équipements – du réseau domestique en enregistrant, supprimant, modifiant la représentation de chacune de ces entités dans le registre de la plateforme de services. Le registre de services est le réceptacle logique de toutes les entités pervasives de l'environnement domestique. Le DPWS Base Driver élaboré dans le projet ITEA ANSO est un exemple de pilote orienté services [2008-2]. Ce travail a fait l'objet d'une présentation et d'un RFP (Request for Proposal, un document d'exigences techniques) acceptés par l'Alliance OSGi [2007-5][2007-6] – www.osgi.org. La spécification et l'implémentation ont été délivrées sur le répertoire open source du projet IST Amigo[1].

Le concept de plateforme de services est à même de permettre la gestion de l'hétérogénéité. La pièce maitresse de l'approche proposée est encore une fois le registre de services. Une médiation technique est effectuée par des pilotes dits "raffinés" qui transforme les représentations d'entités pervasives en objets spécialisés au domaine applicatif visé. Ces pilotes raffinés réagissent dynamiquement à l'apparition, la disparition, la modification des entités utiles dans le registre de services en enregistrant, supprimant, modifiant les objets d'adaptation. Le registre de services est ainsi le réceptacle de ces objets à la sémantique proche de celle des applications qui les requièrent. Cette approche peaufine un concept de la spécification OSGi [OSGi05] – "refined driver" est un nom repris de la spécification OSGi. Tandis que les pilotes orientés services masquent les aspects distribués des applications dans des "proxys" catégorisés par protocole, les pilotes raffinés masquent l'hétérogénéité des représentations protocolaires par des adaptations spécialisées. La médiation technique permet d'adapter les services découverts aux requis des applications. Cette adaptation fait partie des techniques de composition de services dans les réseaux pervasifs – définies dans le "Home

[1] France Telecom DPWS Base Driver specification, Java API, implementation and examples: http://gforge.inria.fr/frs/?group_id=160&release_id=1804

SOA" [2008-5][2008-4][2007-4][2007-2]. Cette approche est en particulier appliquée dans des travaux liant des protocoles de Voix sur IP avec des protocoles de contrôle domestique standards [2008-6]. Ces travaux ont été l'objet d'un deuxième RFP OSGi [2006-3] et de contributions aux projets IST ePerSpace, ITEA ANSO et Systerminal – projet labellisé par le pôle français de compétitivité "Images et Réseaux".

La dynamique de l'environnement a été le sujet des travaux suivants. Ces travaux tentent de raffiner les approches du laboratoire Adèle, Service Binder [CeH03] et aujourd'hui iPojo [EHL07] par addition de techniques de sélection contextuelle de services. Cette sélection est permise d'une part par l'addition de propriétés contextuelles aux services disponibles dans le registre de services et d'autre part par l'addition de propriétés aux entités requérant des services. Après un travail de catégorisation du contexte [2006-1], un premier prototype [2007-3] est construit au-dessus du cadriciel OSGi Declarative Services [OSGi05] qui est la normalisation de l'approche appelée Service Binder. Les aspects contextuels sont mieux séparés par l'usage de techniques à POJO (Plain Old Java Object) dans un deuxième prototype [2007-7].

La dernière contribution vise la gestion de l'hétérogénéité des plateformes d'exécution dans le domaine de l'administration des équipements du réseau domestique. Elle se concrétise dans un effort de normalisation auprès du Forum UPnP [2007-9] – www.upnp.org –, précisément dans la codirection de l'UPnP Device Management (ex-Execution Platform) Working Committee (Co-directeurs : l'auteur pour le Groupe France Telecom et Jooyeol Lee pour Samsung) depuis le 9 octobre 2007. Les travaux devraient durer jusque mi-2009. La proposition de cette thèse d'un modèle générique d'administation du cycle de vie logiciel de plateformes d'exécution[1] a influencé directement la spécification UPnP et a un impact sur d'autres organisations : le Broadband Forum[2], la Home Gateway Initiative et l'Alliance DLNA mentionnent ces travaux. Un RFP accepté par l'Alliance OSGi [2007-8] est le miroir de ce travail pour la plateforme d'exécution OSGi. Une réponse (RFC) à ce RFP est en cours d'élaboration par les membres de l'OSGi Residential Expert Group.

Ces différentes contributions pavent des chemins orientés dans une même direction : faciliter la composition de services offerts par des équipements électroniques d'un environnement pervasif. Distribution, dynamique et hétérogénéité sont les trois aspects complexes qu'adressent ces travaux de thèse parmi les problèmes posés par l'ouverture de ce type d'environnement.

Le réseau domestique est choisi comme cible des travaux car il est un environnement pervasif par excellence. Il est un terrain de prédilection pour l'expérimentation des modèles de conception recherchés. Premièrement, la maison est un système ouvert : de nombreux utilisateurs et équipements divers peuvent aller et venir dans la maison et y séjourner plus ou moins longtemps. Ensuite, le réseau domestique est dynamique et distribué : l'utilisateur est mobile et vaque à de nombreuses activités qui peuvent être assistées par divers équipements qui se connectent, se déconnectent au gré de leurs déplacements dans la maison ou en raison de mesures de sauvegarde d'énergie. Aussi, ces équipements distribués sont hétérogènes : leurs capacités, leurs langages de programmation, leurs protocoles de communication et leurs sémantiques d'utilisation sont variés. Enfin, les aspects embarqués sont exacerbés par la concurrence forte sur les prix de vente entre les fabricants de ce marché en plein essor.

[1] Email publié le 21 juillet 2008 sur la liste du Comité

[2] Proposition n°dsl2008.689.00 "Proposal for Enhanced Application Management via TR-069" présentée par H. Kirksey, Motive, et tirée d'une présentation de Levent Gurgen et de l'auteur.

4 CADRE DE TRAVAIL

Cette thèse s'est déroulée au sein de l'équipe Adele du Laboratoire Informatique de Grenoble (LIG). La mission de l'équipe est la définition d'environnements et d'outils pour le génie logiciel industriel. La thèse participe à la réflexion sur l'un des thèmes principaux de travail de l'équipe : la composition dynamique d'application sur des plateformes à services. L'orientation service est devenue un paradigme fondamental dans le panorama technique d'Adele. Le couplage lâche et l'abstraction de l'approche se sont avérés pertinents pour la conception d'applications à l'évolution rapide. L'effort de recherche et de développement de la plateforme OSGi felix et de certains modèles de programmation au-dessus de celle-ci montre l'expérience de l'équipe dans le domaine des plateformes à services.

La thèse s'est déroulée en parallèle au sein de projets internes des laboratoires de recherche et de développement du Groupe France Telecom, appelés aujourd'hui Orange Labs. Les opérateurs de télécommunications tels que le Groupe France Telecom sont, d'une part, considérés comme les acteurs-clefs pour l'administration des équipements des clients et tentent, d'autre part, de fournir des applications innovantes qui bénéficient des fonctionnalités fournies par les équipements possédés par les clients. Dans les deux perspectives, l'objectif est d'inventorier les équipements utilisés par le client et de composer leurs fonctions sans requérir de participation spécifique de sa part. Ces cas d'utilisation ont appelé des études dans le domaine intergiciel, domaine du centre de R&D MAPS (Middleware Plateformes AvancéeS) et en particulier au sein du laboratoire MAPS/AMS. Ce dernier a développé des compétences intergicielles que l'on découvre dans les publications et l'effort open source visible dans la communauté ObjectWeb (www.objectweb.org).

Certains travaux de cette thèse s'inscrivent dans le cadre de projets nationaux et européens. L'auteur a contribué nommément dans le projet RNRT Pise, projet visant l'administration d'équipements industriels sur une large échelle, puis dans trois projets européens aux objectifs innovant dans le contrôle d'équipements dans le réseau domestique : IST ePerSpace, ITEA ANSO, IST Amigo[1]. Des travaux concernant la communication interpersonnelle à partir du réseau domestique sont en cours de contribution dans le projet Systerminal[2] du Pôle de Compétitivité Images et Réseaux de la région Bretagne.

Enfin, certaines contributions de cette thèse ont pour cadre des groupes de normalisation internationaux. L'intérêt technologique de certains groupes et la position de France Telecom dans ceux-ci ont tiré la dissémination de certains travaux auprès du Forum UPnP [2007-9], de l'Alliance OSGi [2006-3][2007-2][2007-5][2007-6][2007-8], de l'organisation ISO/IEC [2006-7]. La création du Comité de Travail UPnP Device Management (ex-Execution Platform) et la proposition technique d'une interface générale d'administration de cycle de vie logiciel sont une des contributions principales, et ont un impact sur d'autres organisations : le Broadband Forum, la Home Gateway Initiative et l'Alliance DLNA.

5 ORGANISATION DU DOCUMENT

Ce document est divisé en trois parties. La première partie de ce mémoire présente l'Informatique Pervasive et dresse un état de l'art des recherches (cf. Chapitre II) et des technologies (cf. Chapitre III) dans ce domaine. L'état de l'art du Chapitre II est établi selon les trois défis majeurs auxquels cette thèse apporte des réponses :

[1] Anne Gérodolle a présenté les contributions au nom de l'équipe dans le projet IST Amigo.

[2] Serge Martin représente l'équipe dans le projet Systerminal.

- La répartition de l'intelligence dans l'environnement,
- La gestion de l'hétérogénéité des langages et des protocoles dans les applications pervasives,
- La sélection dynamique et contextuelle des fonctionnalités utiles de l'environnement selon le contexte de l'utilisateur, les qualités des équipements disponibles et l'espace environnant.

La deuxième partie expose la problématique adressée par cette thèse au-delà des limites des réponses apportées dans l'état de l'art (cf. Chapitre IV). Cette partie décrit ensuite la réponse nommée Home SOA [2007-3][2007-4][2008-2][2008-5], un ensemble de patrons de conception logicielle cohérents en réponse aux défis exposés (cf. Chapitre V). L'approche logicielle est cependant complétée dans certaines situations par des réponses protocolaires.

La troisième partie décrit (cf. Chapitre VI) et évalue (cf. Chapitre VII) les réalisations d'applications sur le réseau domestique au-dessus de ce cadriciel. Les réalisations les plus importantes sont :

- La transparence de la distribution d'applications orientées service avec l'Extended Service Binder [2005-1][2006-2][2006-4][2006-5][2006-6][2008-3].
- Le DPWS Base Driver, réponse du cadriciel pour le contrôle d'équipements suivant le nouveau protocole Devices Profile for Web Services [2007-5][2007-6][2008-2].
- L'application du modèle de pilotes raffinés et du modèle de contextualisation de services du cadriciel, dans le cadre d'applications multi-protocolaires de contrôle d'équipements [2007-3][2007-4][2007-7][2008-5], de communication interpersonnelle enrichie [2008-6] et d'administration d'équipements [2008-4] sur le réseau domestique .
- La spécification et l'implémentation d'une interface UPnP d'administration du cycle de vie logiciel de plateformes d'exécution hétérogènes au sein du Comité de Travail UPnP Device Management [2007-8][2007-9].

Le Chapitre VIII conclut le mémoire en ouvrant des perspectives à ce travail.

Chapitre II. INFORMATIQUE PERVASIVE ET ORIENTATION SERVICE

Le paradigme de l'orientation service est présenté comme réponse aux problèmes intergiciels posés par l'ouverture des environnements pervasifs. Ce chapitre présente premièrement une définition des environnements pervasifs et les réponses apportées par l'orientation service dans la section 1. La section 2 énonce les principes généraux des intergiciels de communication répartie orientés services.

Les objectifs du travail présenté sont de simplifier voir de masquer trois des préoccupations principales des applications pervasives : les aspects distribués de la découverte et de la communication entre entités pervasives, l'hétérogénéité des protocoles et des interfaces existants dans l'environnement, les aspects contextuels de la composition des fonctions offertes par les entités pervasives présentes. Les objectifs ouvrent une recherche de solutions intergicielles. Les sections 3, 4 et 5 dressent un état de l'art des approches et des réponses apportés dans la littérature technico-scientifiques face aux trois défis posés en connaissance des manques des technologies existantes (cf. Chapitre III pour la description de technologies existantes).

1 LE GENIE LOGICIEL AU SECOURS DE L'INFORMATIQUE PERVASIVE

1.1 Informatique Pervasive

Ce qui caractérise un environnement pervasif est son ouverture à des entités distinctes et variées dans un contexte évoluant rapidement (cf. section Chapitre I.1 et Chapitre I.2). Les critères de nombre, mais surtout de variété et de dynamique des équipements électroniques impliqués dans une application décident du caractère pervasif de celle-ci. L'Internet, les réseaux locaux comme le réseau domestique, les réseaux mobiles peuvent être considérés comme des environnements pervasifs dès lors que les applications envisagées impliquent l'interaction d'équipements électroniques variés et à la présence variable.

Dans l'acception générale de l'Informatique Pervasive, le déploiement et l'organisation des équipements électroniques sont de surcroit voués à l'assistance de l'utilisateur. Le quotidien de l'utilisateur est visé par le dessin général que brosse Mark Weiser, l'apôtre de l'Informatique Pervasive (dite aussi Ubiquitaire) [Wei91]. Mahadev Satyanarayanan [Sat01], l'éditeur en chef du magazine IEEE Pervasive Computing, introduit

le sujet en citant Mark Weiser et en définissant l'essence de l'Informatique Pervasive comme la création d'environnements saturés par des capacités de calcul et de communication intégrées harmonieusement avec les utilisateurs humains : "The essence of that vision [Mark Weiser's vision] was the creation of environments saturated with computing and communication capability, yet gracefully integrated with human users".

Les exemples de défis techniques sont nombreux. Rendre transparente la découverte et l'utilisation d'équipements électroniques adéquats dans des situations quotidiennes est un défi que tentent d'adresser les projets du domaine de la maison intelligente (smart home) [Hel02][VaF02]. [VFL05] donne l'exemple d'un système de sécurité surveillant l'utilisation du four et alertant l'utilisateur suivant différents systèmes d'affichage d'informations. La remontée d'informations issues de capteurs nombreux et hétérogènes par leurs protocoles et leurs modèles de représentation de données est un autre défi. [Sta02] présente une solution visant l'assistance de personnes âgées à leur domicile. Ces défis techniques sont adressés par le Génie Logiciel (cf. section 1.2) qui tente de structurer et de simplifier l'imbrication logicielle demandée par l'Informatique Pervasive.

Face à l'ouverture de l'environnement, de nombreuses technologies protocolaires sont proposés [Hel02] dans chaque domaine d'application. Elles permettent généralement d'inventorier dynamiquement les équipements présents afin de les contrôler. A cette fin, elles comportent des protocoles de découverte et de description qui les font appartenir à la classe des technologies orientées service (cf. sections 2.3 et 2.4). Les apports des techniques intergicielles et de l'orientation service sont énoncés dans les sections suivantes.

1.2 Les techniques intergicielles

Les réponses du Génie Logiciel aux problématiques de l'Informatique Pervasive sont intergicielles. L'objectif principal des intergiciels est de masquer les aspects complexes de l'imbrication logicielle nécessaire aux applications complexes. Le nom "intergiciel", ou middleware, introduit sa propre signification : il représente les couches logicielles du "milieu". Ces couches s'insèrent entre les couches matérielles, dites hardware, et l'application logicielle finale. Elles adaptent généralement l'application à un environnement matériel ou logiciel qui peut varier ou ajoutent la gestion d'une préoccupation (un aspect généralement non-fonctionnel d'une application) au logiciel global. Ce nom est apparu avec le développement rapide des applications distribuées utilisant des protocoles de communication répartie, des applications complexes constituées de composants variés, des applications aux cibles matérielles et réseaux hétérogènes.

Développer et maintenir des applications distribuées mènent les développeurs à faire face à l'hétérogénéité grandissante des technologies. Afin de simplifier l'écriture et la maintenance de ces applications, il est naturel de séparer l'application des couches de communication avec le matériel et le réseau au travers d'interfaces génériques. La conception d'un intergiciel est ici la définition d'interfaces génériques masquant des implémentations correspondant à des environnements variés. Les Web Services sont un exemple d'intergiciel qui masque l'hétérogénéité des langages de programmation et la variabilité des implémentations. Une section de cet état de l'art – section 4 – fait mention des travaux d'aujourd'hui qui tentent de masquer l'hétérogénéité des protocoles de découverte et de communication.

L'autre but des intergiciels est de fournir des fonctions complémentaires à la fonction principale d'un logiciel. Ces fonctions sont généralement appelées préoccupations non-fonctionnelles, les aspects fonctionnels désignant la logique métier de l'application. Par exemple, Java EE (Enterprise Edition) offres des services de gestion des transactions, la sécurité et persistance aux applications d'entreprise. Le développeur de ces applications est simplement tenu d'implémenter certaines interfaces définies par l'intergiciel et de déclarer les

besoins de l'application dans des fichiers de configuration au déploiement. La section 5 montre des intergiciels qui offrent aux applications des moyens de gérer la sélection de services de manière déclarative.

Les intergiciels impliquent généralement la mise en place de mécanismes d'indirection pour donner l'accès aux implémentations spécifiques d'une interface. Ces mécanismes introduisent inévitablement des pertes de performance. Par conséquent, la conception des interfaces intergicielles doit toujours être accompagnée d'une implémentation démontrant que le surcout ajouté est négligeable face aux atouts d'un intergiciel [Kra03]. Certaines contributions de cette thèse – [2007-7][2008-2][2008-5] – sont évaluées sur des critères de performances liés au caractère embarqué des applications visées. Parfois, de nouvelles difficultés apparaissent avec l'utilisation d'un intergiciel : la configuration de celui-ci pour le déploiement de chaque application [Bern96].

1.3 L'Orientation Service

L'orientation service est sous-tendue par la recherche de couplage lâche entre les applications. L'orientation service peut être considérée selon différents points du vue. [Pap03] distingue par exemple les avantages de neutralité technologique, de couplage lâche et de transparence de la localisation. Tandis que la neutralité technologique semble partiale par le choix des Web Services et alors que la transparence de la localisation ne semble pas être le propre des architectures orientées service, le couplage lâche semble être la marque saillante des approches à service. Cette thèse décrit l'orientation service selon trois caractéristiques essentielles qui assurent le couplage lâche [2008-2] :

- Abstraction : l'interface de service abstrait les fonctionnalités et les aspects non-fonctionnels d'un service.
- Dynamique : le courtage et la liaison tardive des services permettent une composition réactive.
- Administration partagée : l'hypothèse de la pluralité des administrateurs renforce le caractère temporaire des liaisons.

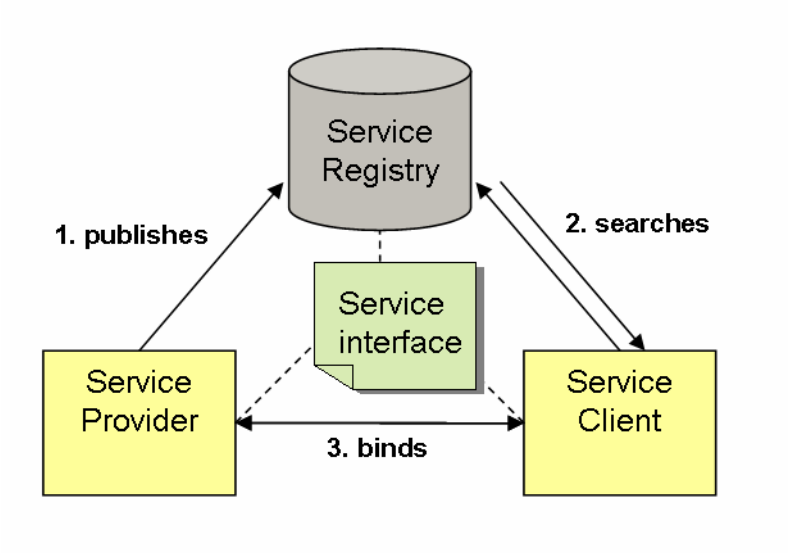


Figure 1 Architecture Orientée Service

Le paradigme d'orientation service définit un schéma d'interaction (cf. Figure 1) entre fournisseur de service, demandeur (ou client) de service et le répertoire de service qui partagent la connaissance d'une interface de service. L'abstraction procurée par l'interface de

service, la dynamique de l'interaction et l'hypothèse d'administration partagée sont illustrés par ce schéma et expliqués dans les sections suivantes.

1.3.1 Couplage lâche par la définition d'une interface de service

Le paradigme d'orientation service répond premièrement à l'ouverture des environnements pervasifs par des mécanismes de découplage des entités fournissant des fonctionnalités, appelées services, et les entités composant ces fonctionnalités. Le découplage repose sur la définition et la publication d'interfaces de services séparées des implémentations. Les interfaces de services décrivent un ensemble d'opérations et les qualités non-fonctionnelles. Dans une architecture orientée service idéale, aucune propriété ne reste implicite entre un client et un serveur. Toutes les liaisons client-serveur sont soumises à une recherche de fournisseur de service selon un contrat explicite.

Par la séparation entre interface et implémentation, par la définition de contrat, l'orientation service diffère peu de l'orientation composant [Szy98]. Toutefois, les composants sont généralement décrits selon un modèle de Boite Grise qui laisse apparaître un certain niveau de connaissance de l'implémentation interne. Cela les rends différents des services qui seraient des Boites (plus) Noires [Cur07]. Les approches à composants seraient pertinentes dans une conception pre facto (pré-développement), les approches à services plus pertinentes dans une approche post facto [CCCS07].

La recherche de couplage lâche est l'explicitation des informations demeurant implicites entre un client et un serveur. Force est de reconnaitre qu'aujourd'hui, la plupart des architectures orientées service garde implicite le partage d'un même protocole de communication et d'une même sémantique métier. Seules les propriétés non-fonctionnelles du service - par exemple, la QoS, la sécurité - sont l'objet de flexibilité apportée par la sélection de services (cf. section 5) pour un type de service recherché [CuM03]. L'hétérogénéité des protocoles et la variabilité de l'expression des interfaces dans un même domaine métier sont des aspects abordés par de nombreux travaux de recherche actuels (cf. section 4).

1.3.2 Couplage lâche renforcé par courtage et liaison tardive

L'orientation service répond deuxièmement à l'ouverture et la dynamique des environnements pervasifs par la définition d'un mécanisme de courtage de service. Les fournisseurs de services enregistrent leurs services auprès d'un répertoire de services. Ils tiennent les enregistrements à jour et les suppriment s'ils ne les fournissent plus. Des mécanismes de découverte passive où les clients sont notifiés des départs et arrivées des fournisseurs se distinguent des mécanismes de découverte active où les clients sont forcés de renouveler leurs requêtes afin de connaître la disponibilité à jour des fournisseurs de services (cf. section 2.3.3). Dans les deux cas, la dynamique du répertoire de services permet une gestion réactive des applications clientes à la disponibilité des fournisseurs.

Les clients ou demandeurs de services requièrent les services selon des critères définis par l'interface de service – nom de l'interface, niveau de qualité de service, etc. Le répertoire de service répond aux requêtes de services des clients en retournant la liste des fournisseurs de service enregistrés qui correspondent aux critères du demandeur (cf. sélection de service dans la section 5). Ce mécanisme de courtage selon toute propriété de service rend la composition flexible.

Enfin, ce schéma d'interaction implique des mécanismes de liaison tardive entre fournisseurs et demandeurs de services. Les liaisons se font et se défont lors de l'exécution. Ces mécanismes de liaisons tardives que l'on retrouve dans les architectures à composants [BCLQ06] alliés au courtage flexible de l'interface de service renforcent le caractère lâche de la liaison entre une application et ses fournisseurs de service. L'application se compose dynamiquement avec les fournisseurs disponibles selon des critères définis au préalable pour chaque liaison.

1.3.3 Couplage lâche pour une administration partagée

L'orientation service répond enfin à l'ouverture des environnements pervasifs en prenant l'hypothèse que l'administration des entités de services est effectuée par des acteurs distincts. Alors que les approches à composants définissent un cycle de vie, les services ne peuvent en définir sans faire référence à un modèle de composant.

L'abstraction offerte par l'interface de service présente les fonctionnalités disponibles sans garantir la disponibilité et la qualité de celles-ci sur le long terme avant négociation. La séparation de l'administration de services fournis et de celle d'une application exacerbe l'aspect temporaire des liaisons. La durée des liaisons de services doit être négociée ou bien, ce qui est le cas de l'Informatique Pervasive, l'application doit se prémunir contre l'éventualité de la disparition du fournisseur de service en cours d'utilisation [EHL07].

1.4 L'Orientation Service pour l'Informatique Pervasive

L'Informatique Pervasive décrit des environnements ouverts où les applications visées devront composer des entités qui ne se connaissent pas à l'avance et réagir dynamiquement à l'arrivée et au départ de nouvelles entités dans l'environnement. L'aspect pervasif d'une application est synonyme d'une grande variabilité des entités constituant l'environnement:

- Variabilité des technologies : les entités peuvent être programmées selon des langages de programmation variés et communiquer selon des protocoles de communications hétérogènes.
- Variabilité des comportements : les entités sont développées de manière indépendantes. Le fonctionnement de chacune peut interférer sans alignement ni sur un algorithme commun ni sur une égalité des qualités de fonctionnement. Les qualités de fonctionnement de part et d'autre, appelées propriétés "non-fonctionnelles", peuvent être des propriétés de sécurité, de qualité de service, etc.
- Variabilité contextuelle : le contexte des applications est changeant. L'utilisateur, les équipements et les éléments environnants agissent sur le contexte applicatif (cf. section 5.1). La pluralité des acteurs et leur relative indépendance implique le caractère partagé du contrôle et de l'administration des programmes distribués.

La programmation des applications pervasives s'oppose par conséquent à la programmation usuelle d'applications distribuées. En effet, avant que Jini [Wal99], SLP [GPVD99], UPnP [UPnP06], les Web Services et les évolutions de CORBA (Corba Trading Service notamment) ne laissent apparaître les prémisses du courtage distribué d'une interface de service, la programmation d'applications distribuées reposait sur la conception conjointe de clients et de serveurs devant être déployés dans un environnement statique et homogène. Dans une application distribuée usuelle, les clients et les serveurs partagent un même langage de programmation, un même protocole et une connaissance mutuelle de leur implémentation. De surcroît, ces clients et serveurs sont administrés de manière commune par une même entité responsable de l'adéquation des actions d'administration sur les différentes parties distribuées.

L'avènement des technologies du tournant du siècle comme Jini, SLP, UPnP, les Web Services et CORBA répondent partiellement au défi de la gestion de la variabilité des applications pervasives. La définition des interfaces de service et leur courtage selon de multiples critères procurent des outils pour la gestion de la variabilité des comportements. La définition de protocoles de découverte présente un premier outil pour la gestion contextuelle

en permettant aux applications de maintenir une connaissance de la disponibilité dynamique entités applicatives.

Les concepts d'architecture orientée service sont appliqués dans le domaine de l'Informatique Pervasive par de nombreux travaux, par exemple [KKS07]. Dans un réseau pervasif, des équipements peuvent être modélisés comme des fournisseurs de services et des demandeurs de service. Certains équipements demandeurs de services composeront les services disponibles sur le réseau afin de fournir d'autres services de plus haut niveau. Tandis que les Web Services sont la technologie phare pour l'application du SOA (Service Oriented Architecture) largement prônée dans le domaine de l'entreprise (B2B) [Pap03], les technologies orientées service sont légions dans le domaine de l'Informatique Pervasive [Hel02] (cf. Chapitre III).

Les travaux de cette thèse s'inscrivent dans cette mouvance. Ils prennent pour base les concepts de composants et de services et appliquent des techniques logicielles pour relever certains des défis de l'Informatique Pervasive. L'état de l'art des intergiciels distribués orientés service permettent de comprendre et d'appréhender les limites des technologies actuelles pour les défis adressés. Leurs principes sont l'objet de la section suivante.

2 INTERGICIELS DISTRIBUES ORIENTES SERVICES

Les intergiciels de communication répartie orientés service présentent un ensemble de couches techniques génériques. L'adressage, le nommage, la découverte, la description, l'invocation, la notification et des couches techniques dites non-fonctionnelles. Dans le cas des intergiciels utilisant le protocole TCP/IP, toutes les couches de cette classification excepté l'adressage appartiennent au niveau application de la pile de protocoles utilisée par Internet. Des exemples d'intergiciels au-dessus de TCP/IP sont donnés dans la section des protocoles Plug-n-Play (cf. section Chapitre III.1.1), des protocoles de communication interpersonnelle (cf. section Chapitre III.2) et d'administration (cf. section Chapitre III.3.2).

2.1 Adressage

La communication entre plusieurs entités d'un réseau nécessite avant tout l'affectation d'une adresse propre à chaque entité. Les intergiciels utilisant les protocoles TCP/IP affectent une IP à chaque nœud du réseau à l'aide de protocoles tels DHCP.

2.2 Nommage

Nommer les entités du réseau ou d'une application permet d'adresser une entité indépendamment de son adresse réseau. Dans certains cas, l'adresse réseau peut changer relativement fréquemment alors que le nom demeure le même. L'application Apple iTunes installée sur une machine donnée est l'exemple d'une application que l'on peut rechercher par un nom alors que la machine sous-jacente pourra connaître plusieurs IP différentes après d'éventuels redémarrages en mode DHCP. SIP est un autre cas de protocole utilisant un nommage distinct de l'adressage (cf. Chapitre III.2.1). Le nommage n'est pas une phase obligatoire dans tous les intergiciels.

2.3 Découverte

La connaissance des équipements à contacter peut être indiquée à une application de deux façons distinctes :

- Une liste d'adresses ou de noms est inscrite par configuration de l'application.

- Seuls certains critères de sélection sont inscrits par configuration dans l'application. L'application recherche les entités réseaux répondant à ces critères durant l'exécution à l'aide d'un protocole de découverte.

La plupart des protocoles de contrôle d'équipements domestiques sont pourvus d'une couche de découverte (cf. section Chapitre III.1). Ces protocoles sont souvent appelés "Plug-n-Play" (traduction possible : "branche et joue"). Il suffit de les brancher afin qu'ils soient détectés, reconnus par une application qui invite l'utilisateur à opérer une liste d'actions sur ceux-ci.

Ces protocoles obéissent de ce fait au paradigme de l'orientation service. En effet, la découverte repose sur la connaissance d'un type de service, ou ensemble de fonctions, partagée par le fournisseur de service – l'équipement – et le demandeur de service – l'application détectant l'équipement. Cette connaissance étant publique – non associée à un seul équipement – le couplage entre demandeur et fournisseur est dit *lâche*.

La liaison entre demandeur et fournisseur de service s'effectue à l'exécution après la phase de découverte. Un fournisseur de service publie ses services auprès d'un registre de service et un demandeur de service peut demander au registre la liste des fournisseurs répondant à des critères de sélection dont le plus commun est le nom de l'*interface de service* (cf. Figure 1).

La section suivante décrit une variante de ce modèle de collaboration utilisant les moyens multicast répandu sur les réseaux locaux. La section d'après énumère les possibilités offertes par les architectures définissant un répertoire de service. Enfin, une section distingue les mécanismes de découverte active des mécanismes de découverte passive.

2.3.1 Moyens multicast

Afin de garder la fonctionnalité de découverte mais d'éviter le déploiement d'un répertoire de services, certaines solutions utilisent des moyens de communication multicast. Sur les réseaux locaux, la plupart des protocoles Plug-n-Play (cf. section Chapitre III.1.1) utilisent le protocole IP Multicast à cette fin. Au contraire du protocole IP Broadcast qui permet d'envoyer un message à toutes les entités connectées à un même réseau, la communication d'un paquet multicast ne vise que les entités ayant joint au préalable un groupe multicast donné. Le protocole IGMP (Internet Group Management Protocol) permet aux entités réseaux de joindre une adresse multicast. Une adresse multicast est définie par une adresse IP et un port, l'adresse IP devant appartenir à une adresse de classe D entre 224.0.0.1 à 239.255.255.254. Les protocoles standards possèdent une adresse publiquement enregistrée auprès de l'organisme IANA (Internet Assigned Numbers Authority). Par exemple, UPnP utilise le protocole SSDP et son adresse multicast 239.255.255.254:1900. L'implémentation du protocole IGMP est généralement répandue sur les routeurs.

Le protocole IP Multicast passe difficilement à l'échelle de grands réseaux. Sur le réseau Internet, certains intergiciels comme JXTA (www.jxta.org), IRC (Internet Relay Chat), mettent en place des artifices utilisant la communication unicast TCP sur Internet et la communication multicast sur les réseaux locaux afin d'envoyer un même message à un groupe d'entités distinctes réparties sur de nombreux réseaux locaux et sur Internet.

Les moyens multicast du réseau prennent la place logique du répertoire de services (cf. Figure 2). Les fournisseurs de services publient leurs services par annonce multicast sur le réseau. L'annonce est écoutée par les clients qui se sont joints au réseau multicast défini par le protocole. Les clients peuvent aussi envoyer une requête multicast afin de connaître tous les fournisseurs de services. La requête peut être raffinée par des moyens et des critères de sélection de service (cf. section 5.3). Chaque fournisseur de service correspondant aux critères de sélection répond alors par un message de réponse unicast à l'adresse du requérant.

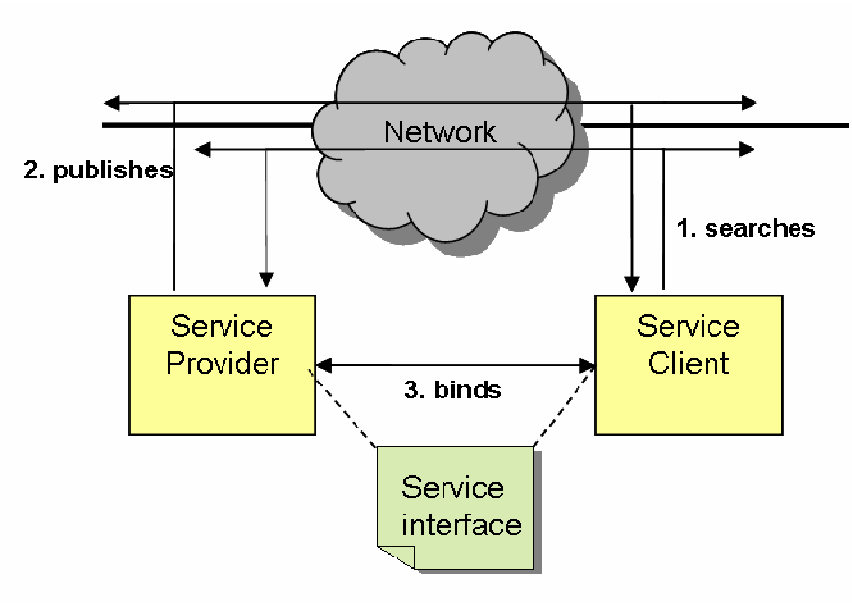


Figure 2 Architecture Orientée Services avec utilisation de moyens multicast

2.3.2 Répertoire de service

Une architecture utilisant un ou plusieurs répertoires de services est une solution qui permet le passage à l'échelle des protocoles de découverte utilisant IP Multicast. En effet, les moyens multicast chargent la bande passante et les moyens de routage. De surcroit, ces moyens sont contre-indiqués dans des réseaux moins fiables que les réseaux Ethernet, par exemples les réseaux Wifi (Wireless Fidelity) [FAYF04]. En établissant des moyens unicast sans communication de groupe, un répertoire de service fiabilise et économise les messages liés à la découverte sur les réseaux locaux. Certains protocoles – par exemple SLP [GPVD99] et DPWS [DPWS06] – définissent le passage d'un mode de découverte multicast à un mode de découverte unicast (et vice-versa) lors de la découverte (et le départ) d'un répertoire de services sur le réseau.

Un répertoire de service ou plusieurs répertoires de services peuvent être déployés selon les exigences techniques visées par un protocole :

- Fiabilité par redondance : SLP permet par exemple le déploiement de plusieurs répertoires identiques. Les fournisseurs de services doivent enregistrer leurs services dans tous les répertoires découverts.
- Passage à l'échelle et séparation de domaine d'administration par spécialisation : SLP prévoit aussi la possibilité de spécialiser chaque répertoire déployé dans un domaine d'administration donné (scope, cf. section 5.3).
- Passage à l'échelle par hiérarchisation : une hiérarchie de répertoires permet le passage à l'échelle Internet de UDDI [UDDI02].

2.3.3 Découverte active et découverte passive

Un protocole de découverte active définit les requêtes qu'envoie le demandeur de service à l'adresse du répertoire de service ou à une adresse multicast. Il définit aussi les réponses que le demandeur reçoit de manière synchrone de la part du répertoire de service ou directement de la part de tous les fournisseurs correspondant à la requête active.

La découverte est dite passive si le demandeur de service peut écouter les publications spontanées de services. Deux situations se distinguent :

- Le protocole de découverte définit un mode de souscription aux événements de publications auprès du registre de services. Les souscriptions sont généralement appelées requêtes persistantes. Le répertoire de services garde en mémoire les requêtes indiquées persistantes et envoie une réponse de manière asynchrone dès qu'un service publié correspond à la requête. La version 3 de la spécification UDDI (http://uddi.org/pubs/uddi_v3.htm) définit une interface de souscription.
- Le protocole de découverte indique que les fournisseurs de services publient spontanément leurs services par les moyens multicast. Les demandeurs de services doivent alors se joindre à l'adresse multicast afin de recevoir passivement les annonces. Bien que DPWS définisse des modes de découverte active avec et sans répertoire de services, cette notification spontanée des fournisseurs demeure multicast dans les deux situations. Les serveurs DPWS envoient un message Hello et un message Bye multicast lors de leur connexion et de leur déconnexion indépendamment de la présence d'un Discovery Proxy.

La plupart des protocoles de découverte définisse les 2 types de découvertes pour des raisons d'efficacité. En effet, si le protocole de découverte n'est qu'actif, les demandeurs de services se doivent d'envoyer des requêtes périodiquement ou au moment opportun d'utiliser un service. Les requêtes périodiques chargent le réseau si la période est courte et le demandeur de service effectue des erreurs si la période est longue. Ces erreurs sont l'inactivité tandis qu'un service est présent durant une fin de période et le blocage lorsque le demandeur utilise un service qui vient de disparaître durant une fin de période. N'envoyer des requêtes qu'aux moments opportuns implique que le demandeur n'ait pas besoin de réagir aux événements de présence des fournisseurs. Cela implique aussi que le demandeur est prêt à patienter le temps d'une requête – souvent quelques secondes – avant d'utiliser un service.

Au contraire, si la découverte n'est que passive, le demandeur de service est notifié en temps réel des arrivées et des départs de services mais peut demeurer inactif pendant un long moment si tous les services présents ne se sont déclarés qu'avant son démarrage. Ce problème peut être pallié par une périodicité des annonces faites par les fournisseurs. Mais ce n'est toujours pas optimal.

Dans un réseau fiable, une connaissance temps réel de la liste des services présents est obtenue si le demandeur peut envoyer une requête active au démarrage et ensuite écouter passivement les annonces spontanées des fournisseurs à leur connexion et leur déconnexion. Les 2 modes de découvertes sont donc utiles. Par ailleurs, du fait de la possibilité d'erreurs de communication, la répétition des requêtes actives et des messages de notification spontanée est souvent recommandée.

2.4 Description

La phase de description est nécessaire dans les intergiciels où les clients et serveurs ne se connaissent pas au préalable. Elle est pertinente en Architecture Orientée Service puisque la recherche du couplage lâche repose sur la définition publique de tous les requis du client par rapport à un serveur (cf. section 5 pour les aspects concernant la sélection de service).

La plupart des intergiciels de communication répartie décrivent les entités à l'aide d'interfaces, qui sont dites "interfaces de service" en orientation service. Une interface de service est un ensemble d'opérations abstraites. Une opération abstraite est une opération sans sont implémentation, c'est-à-dire la définition de son nom, de ses arguments typés dans un espace de nommage. Une interface de service mentionne généralement :

- Des espaces de noms utilisés : une interface peut importer des éléments – types, messages, etc. – depuis d'autres interfaces ou spécifications. C'est le cas des

Web Services [GDSD04] avec ses "namespaces" (traduction anglaise des espaces de noms). C'est aussi le cas de Jini [Wal99] où les interfaces sont des interfaces Java et les espaces de noms sont des packages. Dans cette dernière technologie, les imports sont nommés explicitement dans l'interface et se réfèrent à des classes et des interfaces définies par ailleurs.

- Des types : les arguments des opérations sont typés. Les types peuvent faire référence à des types d'un espace de nommage extérieur. Généralement, chaque technologie définit des types simples comme les entiers, les chaines de caractères, les booléens, etc. UPnP [UPnP06], par exemple, ne définit que des types simples, des types complexes (souvent XML) pouvant se cacher dans les variables de type "chaine de caractère" du format UPnP Template Language. Les Web Services, en revanche, distinguent et définissent les types simples et complexes explicitement dans l'interface de service dans le format WSDL (Web Services Description Language). Le Forum UPnP, conscient de l'imperfection des modèles de description actuels, apporte une correction dans la version 1.1 à venir de la spécification UPnP Device Architecture.

- Des messages : les messages sont définis par une liste ordonnée de variables typées qui peut être échangées entre un client et un serveur. Les Web Services, la technologie à services la plus générique, définit ce niveau de granularité. Dans d'autres technologies, ce niveau n'est pas visible; les listes d'arguments d'entrée et de retour sont alors définies directement dans les opérations.

- Des opérations : une opération est décrite par son nom dans l'espace de nom de l'interface, par un message d'entrée ou un message de retour ou par les deux types de messages. Les messages d'entrée et de retour définissent respectivement une liste ordonnée d'arguments d'entrée et de retour. Les Web Services les appellent opérations. UPnP les appellent "actions". Les opérations sont appelées de différentes manières selon les technologies. Dans les intergiciels liés à un programme orienté objet – par exemple Jini – les opérations sont appelées "méthodes". Quatre types génériques d'opérations se distinguent :

  - request-response : ce mode opératoire est le plus commun. Le client envoie un message de requête, le serveur répond par un message de réponse. Les 2 types de messages sont donc définis dans cette opération.

  - one-way : ce mode opératoire dérive du mode request-response. La différence est qu'aucun message de réponse n'est attendu après réception et traitement de la requête par le serveur.

  - solicit-response : ce mode opératoire est un mode inverse au mode request-response. Le serveur prend ici l'initiative d'envoyer un message au client auquel celui-ci répond par un message. Ce mode nécessite la souscription préalable du client sur les événements adéquats du serveur (cf. section 2.6 pour les mécanismes de souscription). Il est disponible par exemple avec WSDL. L'ordre d'écriture des messages d'entrée et de retour dans la définition d'une opération indique implicitement le mode opératoire choisi. Dans le mode solicit-response, le message de retour – output – est écrit avant le message d'entrée – input (cf. Figure 3).

  - notification : ce mode dérive du mode solicit-response. La différence est qu'aucun message de réponse n'est attendu par le serveur après notification du client. Il nécessite de même la souscription préalable du client sur les événements adéquats du serveur (cf. section 2.6 pour les mécanismes de souscription). Ce mode de notification est répandu dans la plupart des intergiciels de communication réparti au contraire du mode solicit-response.

```
<operation name="weatherUpdateRenew">
  <output message="tns:RenewRequest"/>
  <input message="tns:RenewResponse"/>
</operation>
```

Figure 3 Exemple d'opération solicit-response décrite dans le langage WSDL

- Des événements : les événements correspondent aux envois de message spontanés par le serveur au client, soient aux opérations de type solicit-response et notification décrites ci-dessus. L'émission d'événements d'un serveur vers le client nécessite la souscription du client à ceux-ci. Il existe plusieurs modes de souscription dont les principaux suivent les modèles d'observation classique et publish/subscribe.

D'autres éléments peuvent être disponibles :

- Des variables d'états : la définition de variables d'état permet de modéliser l'état et le comportement attendu de l'implémentation de l'interface. Une variable d'état peut être liée à un argument d'entrée ou de retour d'une opération. Le changement de valeur de la variable peut aussi être la source d'un événement. Les variables d'états sont par exemple définies et utilisées de cette manière dans le modèle UPnP. Cette notion d'état est naturelle pour les équipements d'un réseau local comme le réseau domestique. En effet, les équipements comme la machine à laver et la télévision, ne peuvent servir qu'un client à la fois et l'état de l'équipement est partagé par tous les clients potentiels. Toutefois, la notion de variables d'état n'est pas indispensable. Par exemple, DPWS [DPWS06] n'en définit pas, bien que la notion puisse apparaître implicitement lorsque que des opérations semblent liées à des événements par le partage de mots identiques dans la définition des messages et arguments échangés.

- Une liste prédéfinie de métadonnées : une liste de paramètres à évaluer par chaque implémentation peut être indiqué par l'interface de service. Par exemple, UPnP et DPWS standardisent des noms de propriétés pour la description des équipements – Manufacturer, FriendlyName, etc.

Un fournisseur de services annonce les interfaces qu'il implémente en précisant:

- Son adresse ou son nom sur le réseau : un fournisseur de service doit fournir une adresse ou un nom unique sur le réseau afin d'être utilisé par des clients.

- Les paramètres nécessaires afin d'appeler ses opérations et souscrire à ses événements : des adresses précises différentes de l'adresse principale peuvent être données pour l'accès aux opérations et à la souscription d'événements. C'est le cas par exemple dans la description des services UPnP.

- Les métadonnées distinguant son implémentation : chaque implémentation d'une interface de service est particulière et les particularités peuvent être déclarées selon une liste de propriétés. Plusieurs technologies telles que SLP et UPnP permettent l'ajout d'une liste libre d'attributs valeurs. Les Web Services permettent l'ajout de métadonnées décrits selon des schémas XML génériques. L'expression de métadonnées permet la sélection de services selon des critères applicatifs (cf. section 5 pour des exemples de métadonnées). Elle renforce le sens du couplage lâche [CuM03].

- Les protocoles selon lesquels ses opérations sont accessibles. Généralement, un serveur propose ses services selon seulement un seul protocole de communication et la transposition de l'interface sur ce protocole est indiquée par la technologie. Par exemple, le format des messages SOAP pour l'appel d'opérations décrites dans une interface de service se déduit de la lecture de la

spécification UPnP. En revanche, d'autres spécifications permettent le choix libre d'un ou plusieurs protocoles de communication. Le format des messages est alors indiqué pour chaque protocole dans la description, dite "concrète", du fournisseur de service. Les Web Services permettent théoriquement ces déclarations spécifiques appelées "bindings". En pratique, SOAP est souvent utilisé comme seul protocole de communication par les Web Services. De rares travaux montrent l'intérêt de déclarer des variantes multiples d'un même service selon des protocoles distincts [MKF02].

- Les variantes qu'il peut offrir pour un même service : les variantes peuvent différer non seulement selon les protocoles de communication mais aussi selon le niveau de propriétés non-fonctionnelles comme la sécurité, la qualité de service.

2.5 Invocation – ou contrôle

Les opérations d'un service (cf. section 2.4 ci-dessus) sont appelées par le client par appel distant selon un protocole de communication. SOAP, RMI sont des exemples connus de protocoles d'appels d'opérations distantes au-dessus de TCP/IP. Les intergiciels de communication répartie spécifient le contenu des en-têtes et du message utile – payload – de ces protocoles pour les messages d'entrée, les messages de retour et les messages d'erreur. Par exemple, UPnP, TR-69, etc. utilisent SOAP, Jini utilise RMI, Corba recommande IIOP et RMI, etc.

2.6 Notification – ou événements

La "notification" recouvre les opérations de type notification et solicit-response (cf. section 2.4). Dans les deux cas, l'envoi de message est initié par le serveur et destiné au client. Le contrôle est ici inversé et requiert une opération préalable : la souscription du client à des événements du serveur. Elle peut s'effectuer selon deux patrons de conception différents : l'observation classique et le modèle Publish/Subscribe.

2.6.1 Observation classique

Dans le modèle classique, les intergiciels classiques normalisent 2 interfaces de services :

- L'observé : l'interface "observé" définit une opération de souscription à un ou plusieurs événements. Le serveur implémente cette opération qui sera appelée par le client pour montrer son intérêt à certains événements. La souscription est valide pour une durée déterminée ou jusqu'au moment où l'opération de retrait de souscription est appelée.
- L'observateur : l'interface "observateur" définit une opération de notification. Le client implémente cette opération qui sera appelée par le serveur lors de la détection d'événements intéressant le client.

Par exemple, DPWS spécifie l'usage de WS-Eventing, qui suit ce mécanisme de souscription et de notification classique.

2.6.2 Publish/Subscribe

Le paradigme Publish/Subscribe définit un mécanisme plus élaboré. Ce mécanisme nécessite au moins un acteur supplémentaire qui hébergera des "sujets" ("topics"). Les serveurs publient leurs événements auprès des sujets pertinents. Les clients s'abonnent aux

sujets qui les intéressent. Chaque sujet est responsable de transmettre les événements publiés à la liste des abonnés.

Ce paradigme induit un couplage très lâche entre clients et serveurs. En effet, ces entités ne se connaissent pas si elles n'interagissent que par ce mécanisme. Les clients et serveurs ne connaissent que les sujets. Seuls les sujets ont une connaissance des clients intéressés et des serveurs d'événements correspondant.

Parmi les spécifications du monde des Web Services, WS-Notification est le protocole qui repose sur ce paradigme. Toutefois, seul l'usage de WS-Eventing est spécifié par DPWS, la norme des Web Services pour les réseaux locaux.

2.7 Aspects non-fonctionnels

D'autres couches techniques viennent compléter les intergiciels de communication répartie avec la gestion d'aspects non-fonctionnels comme la sécurité, la qualité de service (QoS), la journalisation (logging), les transactions, la persistance des données, l'administration. UPnP est un exemple d'intergiciel du réseau local qui présentent des spécifications complémentaires pour la sécurité, la QoS. L'administration des équipements est en cours de spécification dans le comité de travail UPnP Device Management.

3 GESTION DES ASPECTS DISTRIBUES DANS UNE APPLICATION ORIENTEE SERVICE

RM-ODP, le modèle de référence de traitement distribué ouvert normalisé communément par l'ITU-T et l'ISO/IEC, est une référence dans la conception d'architecture distribué. Corba, Java RMI et d'autres solutions intergicielles sont conformes aux modèles de RM-ODP [DHDS98]. La partie 3 du modèle [ODP95] décrit les concepts de transparence de la distribution. La transparence peut être apportée à différents aspects de la communication répartie :

- Localisation – masquer l'utilisation d'adresses physiques, en particulier la distinction entre local et distant.
- Accès – masquer les différences de représentation de données et de mécanismes d'appel.
- Substitution – masquer la substitution d'un objet par un autre objet
- Migration – masquer le changement de localisation d'un objet à une adresse différente.
- Persistance – masquer la désactivation et la réactivation d'un objet utilisé.
- Erreur – masquer les erreurs et la recouvrance d'un objet utilisé.
- Réplication – maintenir la cohérence d'un groupe d'objets répliqués à l'interface commune.
- Transaction – masquer les actions requises par les propriétés transactionnelles de l'appel d'opérations.

Les travaux décrits dans ce mémoire adressent les trois premiers aspects. La transparence de la localisation est l'objectif de la gestion de la distribution dans ce mémoire. La transparence de l'accès est l'objectif de la gestion de l'hétérogénéité (cf. section 4). La transparence de la substitution est adressée par l'étude des mécanismes de sélection, de substitution et de continuité de services (cf. section 5).

Un dialogue entre la recherche sur les langages et la recherche sur les mécanismes distribués s'est instauré dans l'histoire de l'Informatique Distribuée [WWWK94]. La recherche sur les langages est accompagnée d'un effort parallèle d'adaptation des modèles de langages aux applications distribuées. Les procédures dans les années 80 ont été associées à des techniques d'implémentations des Remote Procedure Call [BiN84]. Le paradigme objet a eu des déclinaisons sur les protocoles – SOAP, IIOP notamment – et dans le patron de conception Remote Proxy. Il a donné lieu à la normalisation du modèle RM-ODP dans les années 90 et à des patrons de conception tels "export-bind" [KrS05].

Les différents intergiciels qui implémentent ces concepts de distribution transparente – Corba [CORB95], Jini [Wal99], DCOM [DCOM98] – montrent effectivement une programmation où, d'une certaine manière, les appels distants ne sont pas distincts des appels de méthodes locaux. Toutefois, les objets distants doivent implémenter des interfaces particulières et la séparation de l'application en unités de déploiement épouse la séparation des nœuds distribués de l'application.

Certaines solutions prouvent la faisabilité de modifier une application afin de la rendre distribuée dans une phase post-développement. Coign [HuS99], J-Orchestra sont des exemples de solutions de partitionnement automatique d'applications. Ces solutions permettent de ré-architecturer le code binaire d'une application locale afin de la rendre distribuée. Cette fois, la transparence est complète mais comme les applications n'ont pas été créées pour être distribuées, le résultat de la répartition peut tomber dans les écueils de l'Informatique Distribuée comme l'annoncent Waldo et al. [WWWK94].

Des systèmes de déploiement de modules applicatifs existent (cf. section Chapitre III.3.3). La plateforme OSGi est un exemple [OSGi05], la plateforme .NET en est une autre. Ces systèmes permettent le chargement et le déchargement d'unités de déploiement et la résolution automatique des dépendances de ressources entre elles. Utiliser la frontière de ces modules applicatifs comme frontière possible pour la répartition et utiliser les exceptions de chargement et de déchargement de module afin de masquer les erreurs dues à la distribution est une option choisie par des travaux récents [RAR07] qui utilisent la technologie OSGi.

La recherche d'une solution de gestion transparente de la distribution dans une application distribuée est par conséquent premièrement la recherche d'un compromis entre transparence de l'implémentation et assurance de la prise en compte des écueils de l'Informatique Répartie dans les liaisons effectivement distribuées.

3.1 Scénarios

3.1.1 Contrôle distribué d'équipements domestiques

Dans les environnements pervasifs tels que le réseau domestique, les interfaces des équipements sont souvent standards ou définies à l'avance. Par conséquent, faciliter le développement distribué d'applications revient souvent à générer les parties de code réparties de manière statique avant le développement. Le scénario d'expérimentation choisi par article [RAR07] illustre ce point de vue paradoxalement à l'exposition par l'article de capacités de génération de code post-développement.

Un scénario de contrôle d'équipements domestiques peut être le développement d'un point de contrôle multimédia. Ce dernier est une application permettant à l'utilisateur de naviguer dans le contenu d'équipements de stockage multimédia (e.g., NAS (Network Attached Storage), PC, PDA, téléphone haut de gamme) et de faire jouer les contenus découverts sur des équipements de restitution audio-video (e.g., télévision, chaine hi-fi). Le concepteur aura remarqué qu'UPnP est le protocole de référence implémenté par de nombreux équipements de stockage et par des équipements de restitution. Les premiers suivent une interface normalisée UPnP Media Server tandis que les seconds suivent

l'interface normalisée UPnP Media Renderer. Le concepteur souhaite que le développeur n'ait à composer son application qu'avec des interfaces programmatiques dans le langage de programmation visé et à la sémantique spécifique de domaine multimédia tout en étant exempt de tout détail protocolaire.

3.1.2 Répartition post-développement d'une application logicielle

Les scénarios demandant la répartition transparente d'un programme informatique a posteriori sur différents nœuds d'un réseau ne sont pas communs dans les environnements pervasifs d'aujourd'hui tels que le réseau domestique. Les systèmes de distribution transparente les plus évolués visent pourtant ce mode de répartition. La raison principale de la répartition post-développement d'un programme sur des équipements distribués est probablement la nécessité de répartition de charge. Le choix par les auteurs de [RAR07] de l'exemple contradictoire du contrôle d'une machine à café domestique donné montre la difficulté de trouver un exemple pertinent pour la génération de code post-développement dans l'environnement domestique.

Un scénario plus pertinent techniquement pourrait être le suivant : un point de contrôle multimédia muni de transcodeurs est préalablement développé par une équipe. Les transcodeurs permettent au point de contrôle de pallier les incompatibilités éventuelles entre le format des contenus des équipements de stockage et les formats supportés par la lecture des équipements de restitution. Une équipe étrangère à la première souhaite reprendre l'application et la déployer sur un équipement aux ressources trop contraintes pour le fonctionnement correct des fonctions de transcodage. Il s'avère qu'il est préférable de répartir l'application sur 2 équipements distincts : la partie contrôle est pertinente sur l'équipement contraint qui, par sa faible consommation électrique, est destiné à être toujours allumé alors que les transcodeurs, dont les fonctions sont optionnelles pour l'application, ont leur place sur le PC, équipement aux capacités abondantes de calcul et de mémoire. Comme l'application est un programme complet qui n'est conçu que pour fonctionner sur une seule et même machine, le besoin suivant apparaît : répartir l'application sur deux nœuds réseaux distincts dans une phase post-développement.

3.2 Techniques intergicielles

3.2.1 Mémoire partagée

Les systèmes à mémoire partagée distribuée peuvent être considérés comme des implémentations sophistiquées de systèmes distribués. Le but de ces systèmes est très différent des intergiciels distribués usuels. En effet, ils ne sont utilisés qu'à des buts d'Informatique Parallèle où les demandes de ressources de calcul peuvent être importantes. Ainsi, la recherche dans ces systèmes tente de fournir des sémantiques correctes et efficaces pour l'exécution de programmes multi-threadés [ACDK96]. Habituellement, ces systèmes proposent un langage de programmation spécifique où l'utilisateur indique les propriétés distribuées des données partagées. La recherche de performance repose sur l'élaboration de modèles complexes de cohérence mémoire. La cohérence des structures partagées sur plusieurs stations de calculs a un coût et peut devenir un goulot d'étranglement pour les applications distribuées. Les systèmes DSM sont couteux par la mémoire et les capacités des processeurs demandées par la réplication synchronisée des données entre machines.

La combinaison d'une machine virtuelle et d'un tel système [YuC97] montre un système où la distribution est rendue transparente sur des plateformes hétérogènes – grâce à l'homogénéité procurée par la technologie commune de machine virtuelle. La transparence de programmation de Java/DSM est exceptionnelle tout en restant un système coûteux pour les applications réparties de l'Informatique Pervasive.

3.2.2 Passage par copie, passage par copie et restauration

Les intergiciels distribués des systèmes de communication d'entreprises (B2B) et de contrôle d'équipements reposent sur des modèles aux conséquences moins lourdes que celles de la mémoire partagée. La mémoire partagée impose un protocole sophistiqué de maintenance de cohérence entre espace d'adressages distincts. En dehors des cas d'utilisation de l'Informatique Parallèle, souvent pertinents pour des opérations de calculs importantes, cette maintenance de cohérence permanente entre objets partagés est peu pertinente.

Les appels distants dans les intergiciels basés sur un langage ne tentent pas de garder la même sémantique que les appels locaux sur des variables accédées au travers de pointeurs mémoire (références). La puissance des langages de programmation est généralement utilisée pour la construction de structures complexes qui sont basées sur le partage de références dans un tas (heap). Dans les situations distribuées, l'espace d'adresse étant distinct, les appels distants ne pourront pas bénéficier des avantages importants des langages de programmation.

La sémantique usuelle de l'appel distant est le "passage par copie" (call-by-copy) – aussi appelé "passage par valeur". Un paramètre passé dans un appel distant est l'objet d'une copie en profondeur. Aucune modification à distance sur l'objet passé n'est répétée sur l'objet local.

Une alternative au passage par copie est le "passage par copie et restauration". Cela indique qu'après l'appel de méthodes, toutes les modifications effectuées par l'appelé sur les données passées sont transférées à l'appelant et répercutées sur ses données. La "restauration" implique des complications posées par la délicate gestion de structures de données dans le tas.

Le "passage par copie et restauration" diffère du "passage par référence" si les données sont utilisées par l'appelé après l'appel (serveur à état) et si l'appelant exécute un programme multi-threadé où les données sont sujettes à une modification parallèle durant l'appel.

Le passage par référence n'est possible que dans certains langages de programmation comme Java, il est toutefois restreint à certains cas d'utilisation. Avec Java RMI, les références distantes passées en arguments d'appel de méthodes distantes doivent implémenter l'interface UnicastRemoteObject. Le mécanisme est couteux puisque tout appel sur cette référence distante doit alors être retransmis au système appelant.

NRMI (Natural Remote Method Invocation) [TiS03] est une version modifiée de Java RMI. NRMI met en œuvre un mode appelé "passage par copie et restauration" (call-by-copy-restore) à la place du simple mode "passage par copie" (call-by-copy) de RMI pour les arguments n'implémentant pas l'interface UnicastRemoteObject. Les auteurs affirment que NRMI garantit que les appels de méthodes distantes seront identiques aux appels de méthodes locaux dans le cas où les clients utilisent un processus unique et les serveurs n'ont pas d'états, c'est-à-dire qu'aucun état accessible par un argument des méthodes n'est maintenu par le serveur.

3.2.3 Patron de conception "Remote Proxy"

Le patron de conception "Remote Proxy" ou "mandataire distant" est l'évolution des techniques RPC (Remote Procedure Call) dans la programmation orientée objet. RPC est un paradigme répandu qui permet d'implémenter le modèle client-serveur en Informatique Distribuée. Le client initie la connexion RPC en envoyant une requête à un serveur distant connu afin d'exécuter une procédure avec les paramètres donnés. Une réponse est envoyée au client où l'application reprend son cours. Tant que le serveur traite la requête, le client est bloqué.

Au-delà de l'implémentation du modèle architectural client-serveur, les techniques RPC permettent, dans une certaine mesure, de rendre transparent l'aspect distribué de l'appel de procédure. Idéalement, le développeur utilise une interface de procédure identique pour les appels locaux comme pour les appels distants. C'est l'implémentation de la procédure qui appelle directement une entité locale ou relaie l'appel à un serveur distant. Le retour de la procédure retourne dans un cas le résultat de l'appel de l'entité locale, dans l'autre cas le résultat de l'appel distant reformé dans le format indiqué par l'interface. L'implémentation locale de l'interface de procédure qui relaie la requête dans un protocole distribué est appelé "stub" (ou "talon") tandis que l'implémentation distante qui interprète les requêtes dans le protocole distribué pour reformer l'appel sur le langage de programmation coté serveur est appelé "skeleton" (ou "squelette").

Le patron de conception "Remote Proxy" répond aux situations d'appels de méthodes sur des objets distants. Un patron de conception est une réponse à un problème récurrent :

- Le **problème** est que l'appel de méthodes se fait entre objets définis dans des espaces d'adresses différents et que la communication répartie nécessite habituellement la programmation de transformation d'appels dans un protocole particulier.
- La **solution** au niveau langage rend identique l'appel local et l'appel distant de ces objets dans la phase de développement. Ce patron de conception reprend le patron nommé "Proxy" [GHJV95] dans le cadre d'applications distribuées. De même qu'avec RPC, stubs et skeletons sont utilisées de part et d'autre de la connexion distribuée entre un client et un serveur distant. La différence est que le stub est encapsulé dans un objet implémentant la même interface objet de l'objet distant.

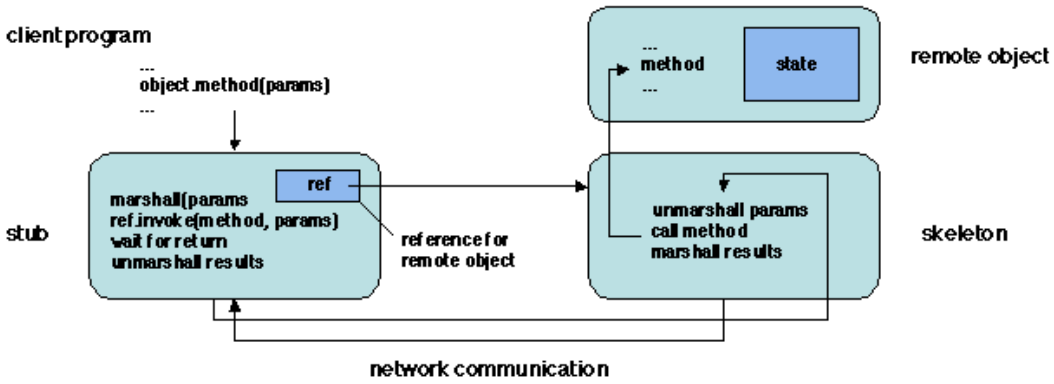


Figure 4 Appel de méthode distante [KrS05]

Les détails d'un appel distant à partir du client local au travers du stub sont illustrés par la Figure 4. Lorsque le client invoque la méthode sur l'objet proxy, le stub "sérialise" les paramètres (marshalling), construit la requête dans le protocole distribué et envoie la requête à l'adresse indiquée par la référence de l'objet distant, référence construite ou obtenue au préalable. Coté serveur, le squelette effectue les mêmes opérations dans le sens inverse : dé-sérialise les paramètres (unmarshalling), relaie l'appel à la méthode de l'objet indiqué dans la requête, sérialise les paramètres de retour et les envoie dans le protocole distribué au stub qui dé-sérialise les paramètres pour les retourner dans la réponse au client. Le code du client séparé du stub est semblable à celui d'un client qui s'adresserait à un objet local.

3.2.4 Patron de conception "export-bind"

Le patron de conception nommé "export-bind" [KrS05] dérive des concepts de RM-ODP [ODP95] où sont définies les liaisons – "bindings" – et les références des objets à lier –

"references". "export-bind" répond apporte une solution à un problème récurrent en Informatique Distribuée :

- **Problème** : le client et le serveur d'une interface programmatique ne sont pas conçus pour fonctionner de manière distante avant l'exécution. Le client connaît toutefois le nom de l'objet à appeler.
- **Solution** : la solution repose sur les notions de serveur de nom – Name Server – d'usine à exporter – Export Factory – et d'usine à liaisons – Binding Factory (cf. Figure 5). Le serveur de noms est un annuaire de références nommées. L'espace de nommage et le serveur de nom sont partagés par le client et le serveur. Le serveur, dès qu'il est prêt à servir son interface de manière distribuée, appelle l'usine à exporter afin que celle-ci construise la référence ((a) step 1) nommée et l'enregistre auprès du serveur de noms ((a) step 2). Le client recherche le serveur par son nom auprès du serveur de nom ((b) step 1). Le serveur de nom lui retourne la référence associée. Cette référence est conçue pour être interprétable par l'usine à liaison. L'usine à liaison est un composant accessible dans l'environnement d'exécution du client. Le client passe la référence à l'usine à liaison qui crée ou obtient un objet proxy encapsulant le stub de l'objet distant ((b) step 2).

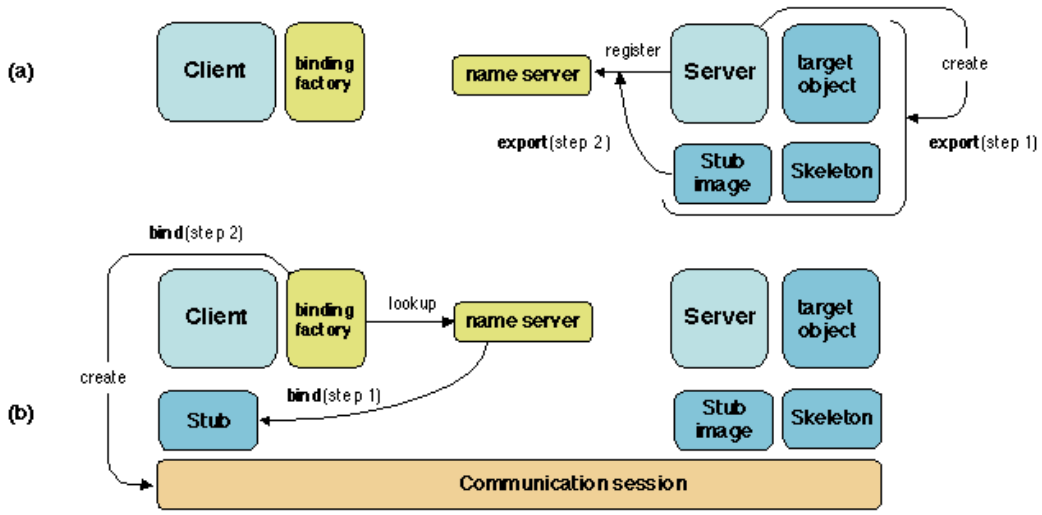


Figure 5 Le patron de conception export-bind [Kra08]

La variante la plus simple de ce modèle présente un squelette et un stub préalablement créé au développement sur la plateforme du serveur. Cette variante requiert le chargement de code à l'exécution puisque le stub est téléchargé et installé dynamiquement sur la plateforme du client.

Une variante plus élaborée impose que les squelettes et stubs ne soient générés qu'à l'exécution. Cela requiert des techniques de génération de code binaire à la volée (cf. section 3.2.7).

3.2.5 Mobilité logicielle

Certains cas d'utilisation d'Informatique Distribuée font appel à des techniques de migration de code. Ces cas d'utilisation se répondent le plus souvent à des problématiques de répartition de charge même si certains auteurs montrent que ce n'est pas la seule problématique. L'article de Fugetta et al. [FPV98], qui demeure une classification de référence en matière de mobilité logicielle, énumère d'autres cas d'utilisation :

personnalisation de services, extension dynamique des fonctionnalités d'une application, autonomie, tolérance aux pannes, et opérations en mode déconnectés.

[FPV98] classifie en particulier la mobilité de code selon deux catégories :

- Mobilité forte : la mobilité forte se définit par le déplacement d'une unité d'exécution de code avec son état d'exécution, c'est-à-dire la pile et le pointeur d'instructions.
- Mobilité faible : seul le code et des données d'initialisation sont transférés. Dans les données transférées, il est possible d'inclure les données décrivant l'état de l'exécution lors de son arrêt sur la plateforme d'origine, ce qui permet à l'unité d'exécution de reprendre dans un état applicatif similaire sur la plateforme cible. Cet état "faible" décrit l'état de l'application sans lien direct avec la pile d'exécution. De ce fait, la mobilité faible peut se rapprocher de la mobilité forte dans ses conséquences.

Les techniques de mobilité faible sont communes aux techniques des intergiciels distribués traditionnels tels que Java RMI, CORBA, DCOM, etc. Elles permettent les patrons de conception "Remote Proxy" et "export-bind". La différence majeure entre les systèmes de code mobiles et les intergiciels distribués traditionnels est la notion de localisation du code pour le développeur d'application (cf. Figure 6). Les systèmes traditionnels tendent à rendre la localisation des objets distribués transparente alors que les systèmes de code mobile la rendront apparente. Dans ces derniers, le développeur nommera explicitement les plateformes sur lesquels il souhaite déplacer des unités mobiles. En effet, les applications visées sont l'optimisation du déploiement de ces unités pour les besoins énumérés plus haut.

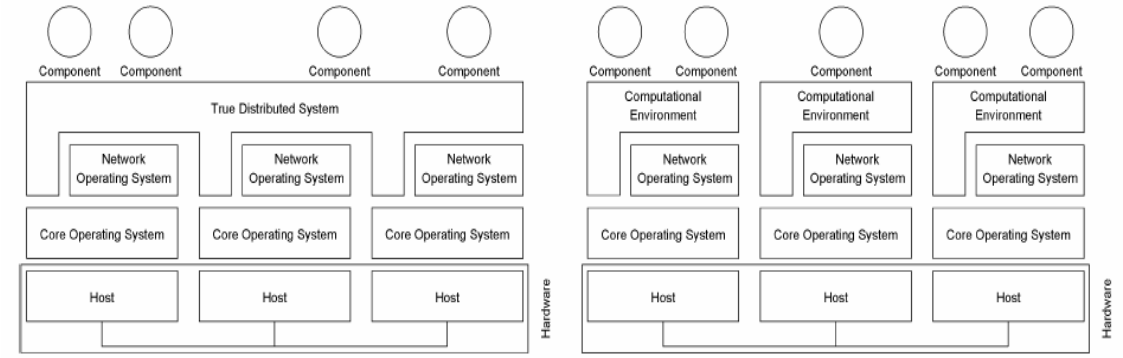


Figure 6 Systèmes distribué traditionnel et systèmes de code mobile [FPV98]

L'Informatique Pervasive, par ses besoins d'adaptation des applications embarquées à leur environnement fait souvent appel aux techniques de Code à la Demande [Don07] qui s'apparente à des techniques de mobilité faible, précisément de la mobilité de code sans état. Ces techniques étendent les fonctionnalités d'une application.

3.2.6 Protocoles binaires et protocoles humainement lisibles

Des protocoles binaires sont spécifiés par certaines technologies pour la communication répartie (par exemple, DCOM, Java RMI). Ces protocoles binaires définissent un format d'écriture d'opération et de paramètres attachés à un langage de programmation. Les identifiants d'objets sont des numéros, les types d'objets sont des classes identifiées par un code binaire (codebase) qui indique pour chaque partie le même chargeur de classe (classloader), les arguments de la méthode sont de même décrits de manière binaire selon un code allégé par une connaissance préalablement partagée entre les deux parties. L'aspect binaire rend le protocole compact. Par leur optimisation et leur compacité, ces protocoles binaires sont particulièrement performants pour échanges entre clients et serveurs

partageant un même langage de programmation. En revanche, ils sont moins adaptés pour des échanges entre parties aux langages hétérogènes.

Au contraire de technologies attachées à des langages de programmation définissant un format binaire pour la communication répartie, SOAP définit un format d'échange humainement lisible à base de XML agnostique au choix des technologies de plateformes. Les technologies utilisant cette approche sont en vogue depuis le début des années 2000 : Web Services, UPnP, etc.

Utiliser un protocole humainement lisible tel XML signifie que les paramètres d'appels de méthodes ou de procédures distants doivent être convertis en chaines de caractères, ce qui requiert une charge de traitement supplémentaire sur le client comme sur le serveur. Des pertes de performances sont montrées par des travaux de comparaison ([DaZ02] mentionne un facteur 2 à 30 selon les tests effectués). Le gain d'interopérabilité entre plateformes est apporté par les protocoles humainement lisibles au détriment de la performance.

[MSSG07] énumère de nombreux travaux existant qui comparent les protocoles d'administration basés sur la technologie XML et les autres technologies telles que SNMP. Tous les travaux montrent que les performances des protocoles basés XML sont pauvres en regard de celles de protocoles tels que SNMP [SNMP02] ou Corba [CORB95] dont les protocoles de communication spécifient des messages plus compacts – SNMP BER, Corba RMI/IIOP.

3.2.7 Réingénierie de code binaire

Réécrire le code binaire d'une application est une technique qui permet de rendre certaines transformations transparentes pour le développeur. La réingénierie de code binaire peut être utile pour le test et l'analyse de code, pour les technologies masquant l'implémentation d'un aspect complexe dans les applications (ici l'aspect distribué). La réingénierie automatique du code binaire est plus pertinente que celle du code source en particulier lorsque l'implémentation de l'aspect complexe dépend du contexte de déploiement ou d'exécution de l'application. Cette réingénierie consiste en la détection d'un schéma de code visé et en la transformation des occurrences détectées suivant un autre schéma.

Le code binaire de la technologie Microsoft COM est un standard portable pour partie. En particulier, les interfaces des composants COM, décrites dans le langage MIDL (Microsoft Interface Description Language), se détectent dans le code binaire [HuS99-2]. Il est alors possible de réécrire le code binaire d'appel des interfaces afin d'ajouter des stubs et des skeletons pour la distribution des appels selon la technologie DCOM.

La réingénierie de code binaire Java est encore plus aisée puisque le code Java compilé, appelé bytecode est destiné à l'interprétation par une Machine Virtuelle Java standardisée (JVM). Des technologies de réingénierie de bytecode Java sont même disponibles : ASM [BLC02] a récemment supplanté l'usage d'Apache BCEL (Byte Code Engineering Library) répandu auparavant dans les communautés Java.

Transformer chaque référence d'objet en référence indirecte implique plusieurs changements dans le code binaire de l'application. Par exemple, toutes les créations des objets potentiellement distribués doivent être réécrites en créations de proxys. Les passages par référence de l'objet "this" doivent être transformées en passage de proxy, si les champs d'objets sont utilisés par d'autres objets, l'accès à tous ces champs doivent être remplacés par des "getters" (méthodes d'obtention directe de la valeur des champs) et des "setters" (méthodes de configuration de champs). De telles méthodes peuvent être générées automatiquement pour toute classe d'application Java. Par exemple, si l'application souhaite changer la valeur d'un champ objet.helloString = "hello", le code doit être changé en objet.setHelloString("hello").

3.2.8 Les écueils de l'Informatique Distribuée

Selon Jim Waldo et al. [WWWK94], la vision habituelle de l'Informatique Distribuée comporte généralement des idées fausses. L'idée d'une description uniforme d'objet dans un unique espace d'adresse est fausse. Les applications distribuées doivent distinguer l'espace d'adresse local et l'espace d'adresse distant lorsque la communication entre objets intervient entre des processus distincts sur la même machine ou sur des machines différentes.

Une situation intermédiaire entre l'Informatique locale et l'Informatique Distribuée peut être définie lorsque plusieurs processus distincts fonctionnent sur la même machine. Dans ce cas, les erreurs de communication ne sont plus un problème car les erreurs affectent simultanément tous les processus au même moment.

Les auteurs constatent que beaucoup d'efforts sont portés sur les langages de programmation afin de rendre l'écriture d'applications distribuées et l'écriture d'applications locales identiques pour le développeur. L'erreur principale vient de la pensée que les difficultés de l'Informatique Distribuée sont portées par le langage de programmation qui tentent de cacher l'accès à des mémoires différentes et la gestion de la concurrence. Pour les auteurs, les difficultés proviennent en réalité des propriétés intrinsèques de la communication distante, difficultés qui sont représentées par les 8 hypothèses fallacieuses en Informatique Distribuée selon Peter Deutsch [Rot06] :

- Le réseau est fiable.
- La latence est nulle.
- La bande passante est infinie.
- Le réseau est sécurisé.
- La topologie est fixe.
- Un seul administrateur existe.
- Le coût de transport est nul.
- Le réseau est homogène.

La latence du réseau est probablement le problème essentiel qui distingue l'Informatique usuelle de l'Informatique distribuée. Un appel distant est beaucoup plus lent qu'un accès mémoire. Afin de limiter l'impact du problème, il est important de limiter le nombre d'appels distants. Si les appels distants sont limités en nombre, il faut aussi les limiter en taille puisque la bande passante n'est jamais infinie. La bande passante dépend des réseaux traversés et des taux d'erreur associés.

Les erreurs dues à la communication en réseau (fiabilité, topologie) sont plus ou moins fréquentes selon les réseaux. Les équipements se joignent et quittent le réseau sur les réseaux dynamiques. Les machines peuvent aussi changer de localisation sur Internet. Les équipements de connectivité connaissent des erreurs même si elles sont peu fréquentes. Ces cas impliquent la nécessité de fiabiliser les échanges de messages et de préparer les appels de méthodes distantes aux cas d'erreurs. Certains procédés d'abstraction permettent de rendre les changements invisibles aux applications. Par exemple, le protocole DNS permet de rendre transparent les changements d'adresses IP.

Les autres problèmes à adresser lors de la répartition d'une application sont les besoins de sécurité de communication, d'adaptabilité en fonction des reconfigurations de l'environnement par différents administrateurs, le coût des solutions techniques mises en place pour permettre les traitements distribués et pallier les problèmes cités.

Une erreur est de penser que le réseau est homogène. Le succès des standards à base de technologies web (Web Services par exemple) montre que ce point est sérieusement pris

en compte par les applications distribuées d'aujourd'hui puisque l'interopérabilité de standard protocolaires humainement lisibles est privilégiée par rapport à la performance apportée par d'autres solutions. Toutefois, l'utilisation de technologies communément admises et indépendantes des langages de programmation ne suffit pas. De nombreux standards existent toujours, de nombreuses variantes existent pour chacun et, de surcroît, deux implémentations d'une même technologie peuvent encore fonctionner selon des comportements différents. Et si l'hétérogénéité est un problème important en Informatique Distribuée, il est encore plus saillant dans l'Informatique Pervasive (cf. section 4).

La transparence de l'aspect distribué dans le langage de programmation est possible. De nombreux langages et plusieurs approches logicielles rendent les interfaces d'objets distants presqu'identiques aux interfaces d'objets locaux. Cependant, les développeurs se doivent de garder en tête la distinction entre les concepts d'appel locaux et d'appel distants. Les interfaces potentiellement distribuées doivent être clairement identifiées et leur conception se doit d'adresser les problèmes qui se posent en Informatique Distribuées. L'implémentation de stubs doit gérer ces problèmes pertinents et l'implémentation des clients doit aussi s'effectuer avec la connaissance de la répartition.

3.3 Exigences techniques

La réalisation des scénarios de contrôle d'équipements pervasifs (cf. section 3.1.1) impliquent une réponse aux exigences techniques suivantes :

- **Transparence de la localisation pour la recherche de fonctions** : durant la phase de développement, la recherche de ressources dans l'environnement distribué doit être semblable – idéalement identique – à la recherche de ressources sur l'environnement local à l'application développée.
- **Transparence de la localisation pour l'utilisation des fonctions découvertes** : durant la phase de développement, l'appel d'objets distants doit être semblable – idéalement identique – à l'appel d'objets locaux.
- **Possibilité d'ajout des propriétés de reconfiguration, de sécurité, de fiabilité** : cette possibilité doit être prévue afin de pallier certains problèmes dus à la distribution.

La réalisation des scénarios de répartition d'applications post-développement (cf. section 3.1.2) impliquent une réponse aux exigences techniques suivantes :

- **Délimitation automatique des parties applicatives pertinentes pour la répartition post-développement** : dans une phase post-développement, la délimitation des parties à répartir demande une analyse automatique des liens entre entités de l'application. Cette analyse est importante à des fins de minimisation de la latence des réponses applicatives dues aux appels distants et l'adaptation de la bande passante utilisée.
- **Réécriture de code binaire pour le traitement post-développement** : dans la phase post-développement, seule la partie binaire de l'application est disponible.
- **Sécurité, reconfigurabilité, fiabilité des communications mises en œuvre** : le code réécrit doit être adapté aux défis de la répartition envisagée. Sécurité, reconfigurabilité, fiabilité sont des préoccupations apparaissant avec la répartition. Le degré d'exigence se rapport à l'environnement visé.

3.4 Etude de diverses architectures existantes

3.4.1 Techniques de Proxy

Dans les années 80, Birrel et Nelson [BiN84] asseyaient les bases de la transparence d'appels répartis distants associée au paradigme procédural. La structure de leur système de répartition annonce la structure des futurs Objets Distribués et du patron "export-bind". [BiN84] décrit des modules de programmes qui importent et exportent des interfaces au travers d'un espace de nommage réparti. Une interface contient alors un ensemble de procédures avec leurs types d'arguments et de retours. Un générateur de stubs crée le code binaire des stubs associés à une interface lors de la compilation.

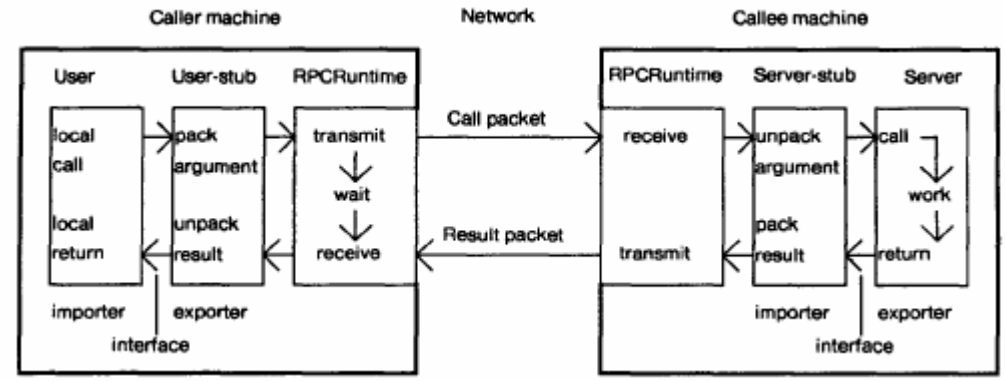


Figure 7 Système d'appel de procédure distant [BiN84]

Corba [CORB95] et DCOM [DCOM98] reprennent les concepts d'appels de procédures répartis et les transposent dans le paradigme objet. Les interfaces sont des ensembles de méthodes dont la signature contient les types d'arguments d'entrée et de sortie. Ces intergiciels définissent un Langage de Description d'Interface (IDL) indépendant du langage de programmation afin d'adresser le défi de l'hétérogénéité des machines. Les stubs de Corba et DCOM traduisent (marshalling) des appels locaux d'un langage de programmation du client en des messages dans un protocole de communication neutre. Les skeletons traduisent (unmarshalling) alors ces messages dans le langage de programmation du serveur. Le retour de l'appel de méthode s'effectue de manière inverse du serveur vers le client. [DCOM98] spécifie un système de gestion d'appel réparti au-dessus de la technologie COM.

Java RMI (Remote Method Invocation) diffère des technologies RPCs comme Corba, DCOM car elle n'adresse pas l'hétérogénéité des plateformes d'exécution [Wal98]. En raison de cette hypothèse d'hétérogénéité, Corba et DCOM ne permettent pas le polymorphisme des objets passés entre client et serveur. Le lien défini statiquement par l'IDL ne peut pas être redéfini par le serveur à l'exécution sans la définition de conventions complexes pour la transposition entre IDL et les différents langages de programmation.

Java RMI impose le langage Java sur toutes les plateformes distribuées participant à une application. Fort de cette hypothèse et des capacités de chargement dynamique de classe permis par Java, le client (ou le serveur) d'une communication distribuée peuvent définir des sous-types d'interfaces. La machine recevant l'objet inconnu sera capable de reconnaître le sous-type et d'installer le stub adéquat. Le stub aura été compilé durant la compilation de l'objet. Le téléchargement de stubs et le chargement de classes de Java permet ensuite le chargement dynamique des classes du sous-type sur le membre recevant le code inconnu auparavant. La responsabilité de la génération du stub repose sur l'objet alors que dans les systèmes RPCs comme Corba et DCOM, la responsabilité repose sur chacune des parties prenantes de la communication.

3.4.2 Découverte de service et Proxy

L'appel distant d'une méthode ou d'une procédure implique que la localisation de l'entité distante soit connue au préalable. Dans un système simple, l'adresse des entités distribuées peuvent être connues par elles avant exécution. Dans la plupart des systèmes (le système de Birrell et Nelson [BiN84], Corba [CORB95], Jini [GTA02]), une entité commune sert d'annuaire de références réparties. Les serveurs s'enregistrent auprès de cet annuaire tandis que les clients s'y connectent pour rechercher leurs interlocuteurs durant l'exécution. Les premiers annuaires étaient des serveurs de nommage (e.g., Corba Naming Service) qui répertoriaient les références d'implémentations d'interfaces sous un nom de serveur. Ensuite sont apparus les serveurs de courtage ou répertoires de services (e.g., Corba Trading Service, Jini Lookup Service). Ces serveurs permettent un courtage des objets enregistrés selon une liste extensible de critères (cf. section 2.3).

3.4.3 Réingénierie de code binaire et Proxy

Coign [HuS99] démontre la faisabilité de la réingénierie de code binaire pour la répartition d'une application. Le système Coign agit sur les composants de la technologie COM (Component Object Model de Microsoft). Dans une première phase de profilage, Il permet d'intercepter tous les appels aux interfaces COM définies dans le programme. Un échantillon de scénarios pertinents est alors nécessaire afin de déterminer quelles interfaces sont possiblement "distribuables". Si des arguments pointent sur un système de mémoire partagée, les interfaces correspondantes sont déclarées non distribuables. Les interfaces appelées peu fréquemment sont les plus éligibles. Dans une deuxième phase, lorsque les interfaces à distribuer sont choisies parmi les interfaces distribuables, Coign génère des objets proxys (stubs et skeletons) de type DCOM [DCOM98] afin de "distribuer" les interfaces choisies.

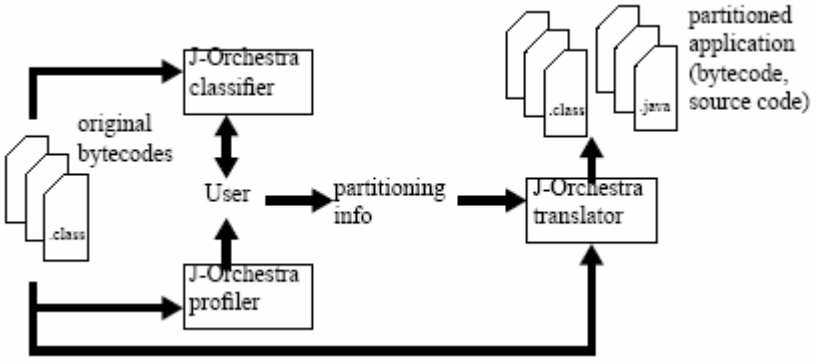


Figure 8 Partitionnement automatique de programmes Java avec J-Orchestra

J-Orchestra [TiS02] est un système de partitionnement automatique de programmes Java. Il transforme le bytecode d'applications Java (code binaire Java) en applications distribuées s'exécutant sur plusieurs machines virtuelles. Selon un plan de déploiement inscrit par l'utilisateur en entrée, J-Orchestra réécrit le bytecode des appels locaux de méthodes en appels RMI et transforme la représentation des objets appelés en proxys distants, etc. Une phase distincte d'analyse statique et de profilage permet de minimiser le trafic réseau associé à la partition. Le profilage est effectué selon des heuristiques simples procurées par J-Orchestra. L'analyse statique permet d'indiquer à l'utilisateur les parties qui peuvent être distribuées (cf. Figure 8). Les parties qui doivent demeurer entières sont généralement des parties de code natives même si J-Orchestra parvient à distribuer aussi le code natif grâce à des proxys Java.

3.4.4 Découverte de service, réingénierie de code binaire et Proxy

La technologie OSGi [OSGi05] spécifie un système de déploiement de modules applicatifs, appelés bundles, sur une plateforme Java locale (cf. Chapitre III.3.3.1). Utiliser la frontière de ces modules applicatifs comme frontière naturelle possible pour la répartition est l'option choisie par plusieurs travaux récents. La modularité du système permet une délimitation naturelle par le développeur des entités à distribuer sans utilisation indispensable d'un outil de profilage (cf. section 3.4.3)

Au-dessus de ce système modulaire, la technologie définit un système de recherche et de liaison de services locaux entre bundles. Ce système est basé sur le patron de conception Service Locator [Fow04] qui répond à la demande de découplage d'une application par rapport à des composants substituable, communément appelés plugins, et qui donne le contrôle de la composition applicative à une entité extérieure à l'application et ses plugins. Ce patron de conception est l'application objet du paradigme d'orientation service :

- Une interface de service est une interface dans un langage objet, sous le nom de laquelle un objet est enregistré.
- Un fournisseur de service enregistre un objet dans le registre de service. Les références enregistrées sont disponibles à tous les objets de la plateforme sous condition de partage de domaine d'application et de droits d'accès éventuels.
- Un demandeur de service, ou client, est un objet qui requiert la référence et invoque des méthodes sur l'interface recherchée.

Ce patron combiné au patron "Remote Proxy" permet le choix transparent d'une liaison locale (appel programmatique) si les clients et serveurs sont co-localisés et d'une liaison distante (appel protocolaire) dans le cas réparti.

Parmi les travaux basés sur ces deux possibilités, certains visent les scénarios de contrôle d'équipements domestiques standards, ce qui implique une certaine influence du développement par les interfaces standardisées des équipements :

- Implémentations de l'**OSGi Device Service Specification** [OSGi05] (par exemple, Apache UPnP Base Driver[1]) **allié à des outils de génération de code source Java pour UPnP sur OSGi [2006-3]** comme les outils de D. Donsez[2]. Ces implémentations et ces outils permettent aux développeurs de concevoir des applications de contrôle d'équipements en utilisant des APIs Java qui représentent les services UPnP par des interfaces Java aux méthodes projetant noms et types naturellement dans le système Java. Les objets appelés sont soit le code d'implémentation du service dans un bundle OSGi, soit un objet mandataire du service UPnP distant qui est réifié sur la plateforme par le Base Driver. La publication [2006-3] expose la finalité de ces outils (cf. Chapitre VI.4.3).

D'autres travaux visent la répartition d'applications de manière complètement transparente pour le développement :

- **Knopflerfish OSGi-Axis bundle**[3] : un bundle qui, grâce au générateur dynamique de mandataires Web Services Apache Axis et au serveur HTTP de la spécification OSGi, permet de représenter toute implémentation de service OSGi

[1] http://felix.apache.org/site/apache-felix-upnp.html

[2] http://www-adele.imag.fr/users/Didier.Donsez/dev/osgi/upnpgendevice/readme.html

[3] http://www.knopflerfish.org/releases/current/repository.xml

comme un Web Service servi par la plateforme. La génération est une instanciation d'objets adaptée à l'offre de service OSGi qui est analysée par réflexivité Java. La génération de stubs clients semble naturellement possible. Ce système manque toutefois d'un système complémentaire de découverte de services répartis sur les plateformes OSGi impliquées.

- **Newton**[1] : ce projet est basé sur les technologies OSGi [OSGi05] et Jini [GTA02] et suit les concepts d'architecture SCA (Service Component Architecture), un ensemble de spécifications issu de la collaboration industrielle nommée Open SOA (www.osoa.org). Newton propose un modèle à composants consistant en la composition de services locaux OSGi et distants Jini en ensembles appelés composites décrits par un fichier de métadonnées. Celui-ci indique les classes à instancier, classes qui représentent les composants de l'application. Il décrit aussi les services que chaque composant requiert ou offre avec le type de lien. Le type peut être spécifié local ou distant. Si le lien est dit local, il sera fait d'appels programmatiques direct Java comme dans la programmation OSGi classique et utilisera le registre de service OSGi. S'il est dit distant, il sera fait d'appels RMI et utilisera le répertoire de service Jini.

- **Extended Service Binder** : ce travail, qui fait partie des contributions de cette these est principalement basé sur les technologies OSGi, SLP, RMI [2008-3][2006-6][2006-4][2006-2][2005-1] et a fait l'objet de quelques variantes [2006-5]. Il est l'objet de la section Chapitre VI.2.

- **R-OSGi** [RAR07] : ce travail tire son originalité de la complétude de son approche. Celle-ci combine un système de réécriture binaire à l'exécution allié à un algorithme de "closure" de ressources dépendantes de code, un répertoire de services OSGi réparti masquant la recherche distante de service, l'utilisation du modèle du patron de conception "Tableau Blanc" pour la communication asynchrone et l'utilisation des exceptions de chargement et de déchargement de module afin de masquer les erreurs dues à la distribution. Si certains traits de l'implémentation et les choix des scénarios sont critiquables, force est de reconnaître que l'article [RAR07] est une référence de la communauté OSGi qui pose clairement le problème et la solution de scénarios de répartition d'applications post-développement en regard des possibilités procurés par les nouvelles technologies de déploiement modulaires telles qu'OSGi [OSGi05].

- **D-OSGi** [MSSG08] : l'objectif de ces travaux est de distribuer le modèle d'application orienté service d'OSGi à une fédération de plateformes. L'originalité de cette nouvelle plateforme appelée D-OSGi réside en l'utilisation du modèle publish-subscribe pour les échanges entre plateformes et sur le choix de l'intergiciel Lime pour l'implémentation de cette architecture distribuée. Cet intergiciel connu dans l'InformTraatique Mobile [MPR06] permet le partage d'un espace constitué de données appelées tuples entre différentes plateformes à la connexion intermittente. L'intergiciel Lime offre tolérance aux pannes, mobilité de code et d'autres propriétés intéressantes dans l'Informatique Pervasive. Les performances associées sont toutefois inférieures à celles de R-OSGi. Les auteurs présentent ce défaut comme bien inférieur aux avantages apportés par le support de Lime. D-OSGi est comparé à R-OSGi dans [MSSG08] et utilise officiellement une technique similaire de closure de ressources dépendantes que l'article appelle la technique du porte-documents ("briefcase").

[1] http://newton.codecauldron.org

R-OSGi est probablement la tentative la plus achevée de distribuer des services sur plusieurs plateformes homogènes de manière transparente. La phase de réécriture s'effectue à l'exécution, phase la plus tardive après développement. Le travail débuta en 2005 dans un laboratoire Suisse et a été le sujet de plusieurs articles publiés par Jan S. Rellermeyer jusqu'à la fin de l'année 2007. Le résultat du projet est livré en open source[1]. Le projet Newton de la Société Paremus et l'Extended Service Binder (cf. Chapitre VI.2) ont été concurrents dans cette recherche de transparence sur un système d'applications modulaires. L'Extended Service Binder utilise RMI pour la génération de proxies au moment des liaisons de services et SLP pour la découverte de services sur des plateformes OSGi réparties. Le choix de SLP, commun à R-OSGi, n'est certainement pas un choix pertinent puisque c'est un des rares protocoles à ne permettre que la découverte active alors que le mode événementiel est demandé par les applications pervasives. Si R-OSGi est plus avancé, c'est qu'il optimise la génération de proxies avec le kit de développement ASM. Cette génération de proxies et la communication sont démontrées plus efficaces sur R-OSGi par rapport à un système RMI (cf. [RAR07]). L'utilisation des erreurs de déploiement de modules afin de masquer les erreurs dues à la distribution est aussi une astuce intéressante. Surtout il met en œuvre une analyse des dépendances de code afin de déployer le code manquant sur la plateforme cliente distincte.

4 GESTION DE L'HETEROGENEITE DES EQUIPEMENTS

Alors que les applications pervasives deviennent possibles et que le réseau domestique devient une réalité [Aie06], seules quelques rares ilots d'équipements montrent une collaboration réelle au service de l'utilisateur. Connecter les ilots de découverte ("Connecting Islands of Discoverability") est un des quatre thèmes de recherche en Informatique Ubiquitaire discerné par Keith Edwards [Kei06].

Le meilleur exemple est certainement représenté par les équipements domestiques multimédia dont le succès est accompagné aujourd'hui par un effort de normalisation important par les deux consortiums UPnP – www.upnp.org – et DLNA – www.dlna.org : des équipements jouent le rôle de serveurs de contenus (Media Server), par exemple les équipements de stockage autonomes appelés NAS (Network-Attached Storage), le mobile, le PC. Un deuxième type d'équipements possède le rôle de serveur de restitution (Media Renderer), par exemple la télévision, la chaine Hi-Fi, le centre multimédia, la Set-Top-Box. Ces deux entités sont mises en relation par une troisième entité appelée point de contrôle (Control Point). Ce dernier va permettre à l'utilisateur au travers d'interfaces graphiques de naviguer dans le contenu disponible, de choisir un contenu à visionner, et de jouer ce contenu sur l'équipement de son choix, par exemple la télévision du salon.

D'autres exemples existent par exemple dans la domotique. Un système centralisé se connecte à de nombreux capteurs et appareils électronique de confort. Il permet à l'utilisateur de contrôler les lumières, les volets et les portes afin d'optimiser le confort et la sécurité dans les situations de la vie quotidienne. Le système contrôle habituellement tous les équipements au travers d'un protocole unique.

Ces exemples de systèmes domestiques montrent non seulement que les domaines d'applications demeurent séparés mais que de nombreux protocoles concourent pour l'implémentation des mêmes systèmes (cf. Figure 9). Le domaine multimédia est par exemple occupé principalement par UPnP/DLNA, les technologies Apple (Bonjour, iTunes) et le standard IGRS – www.igrs.org. La domotique est, elle, visées par les organisations ou marques importantes suivantes : KNX (la convergence des standards européens EHS,

[1] http://flowsgi.inf.ethz.ch/r-osgi.html

BatiBUS, EIB), l'initiative asiatique Echonet, l'Alliance ZigBee, la compagnie LonWorks, la compagnie X10.

Les scénarios ci-dessous montrent le besoin de permettre aux applications d'utiliser aisément plusieurs protocoles issus de domaines d'applications parfois distincts. Le problème ne se limite pas seulement au besoin passerelles de traduction ou dans le besoin d'interfaces génériques pour accéder à tout protocole. En effet, la variété de la sémantique des interfaces définies au-dessus de ces protocoles hétérogènes rend inutile les passerelles simples entre protocoles. Il s'agit aussi de surmonter la "fragmentation des interfaces" [Vay01]. Même dans un seul protocole, deux types d'interfaces définissant sémantiquement une imprimante peuvent être décrites dans un ensemble d'actions sémantiquement proches mais syntaxiquement distincts. Des techniques de médiation évoluées sont nécessaires afin de rapprocher les protocoles et interfaces à la sémantique similaire mais à la syntaxe distincte.

4.1 Scénarios

L'innovation dans les services de télécommunications est tirée par les possibilités de convergence du réseau IP – ADSL, IPTV, VoIP, Partage multimédia, domotique. De nombreux scénarios innovants reposent sur le rapprochement de plusieurs protocoles et de différents domaines d'applications. Dans cette section sont proposés trois types de scénarios : les scénarios de contrôle d'équipements, de communication personnelle et d'administration d'équipements. Dans chaque scénario, l'hétérogénéité des protocoles et le rapprochement de domaines d'applications distincts sont mis en exergue.

4.1.1 Contrôle d'équipements

Figure 9 Domaines d'applications et protocoles hétérogènes

Maxandre est un jeune homme qui a installé les dernières technologies réseaux dans son appartement.

4.1.1.1 Système home cinéma intelligent

Maxandre a installé une télévision UPnP qui lui permet de rechercher des fichiers vidéo sur son PC à partir de la télévision dans son salon afin de les visionner sur celle-ci. Il possède aussi un boitier intelligent qui écoute les événements UPnP afin de réagir à ceux-ci en adaptant les services de confort environnant comme les lumières X10 et les volets ZigBee. Lorsqu'un film démarre sur la télévision, les lumières se tamisent et les volets se ferment dans le salon.

Ce premier scénario mêle la domotique au multimédia et utilise plusieurs protocoles dans le domaine de la domotique. Plusieurs protocoles, UPnP et iTunes, auraient pu aussi être

utilisé pour la recherche de contenus multimédia sur des applications diverses, UPnP et Apple iTunes.

4.1.1.2 Partage de services entre plateformes intelligentes

Le boitier intelligent de Maxandre est un équipement qui est branché au réseau domestique par un cable Ethernet et un cable éléctrique avec une technologie de courants porteurs. Cependant, le boitier n'est pas capable d'écouter les protocoles sans fils comme le protocole ZigBee. Par conséquent, la compagnie vendant le boitier vend aussi une passerelle complémentaire afin de lier les équipements radio aux applications du boitier. La passerelle fait le pont entre les équipements radios ZigBee et un protocole connu par le boitier.

Une mise à jour du boitier intelligent est nécessaire afin de communiquer avec les nouveaux services procurés par le boitier complémentaire. Par conséquent, lorsque Maxandre branche la passerelle, il lui est demandé s'il accepte que la mise à jour puisse être effectuée.

Ce deuxième scenario illustre une application domotique usuelle où les équipements sont accédés au travers d'une passerelle spécifique. La passerelle est connectée utilise une couche physique spécifique du réseau capillaire formé par les équipements radio ZibBee. Cette passerelle permet à d'autres plateformes d'utiliser les services domotiques.

4.1.1.3 Diagnostique du fonctionnement d'équipements

Maxandre s'aperçoit que l'une des lumières du salon ne se tamise pas lorsqu'un film commence sur la télévision. Il ouvre les pages web du boitier intelligent afin de tester et de configurer son réseau domestique. Il teste les équipements de lumières du salon un par un et configure la localisation des lumières ZigBee. La localisation de la lumière correspondante n'était pas configurée avant cette opération. Le système Home Cinema permet maintenant de tamiser toutes les lampes du salon.

Ce scenario d'auto-diagnostique par l'utilisateur (self-care) montre le besoin d'outils agnostiques aux technologies afin d'aider les utilisateurs dans leurs taches d'analyse des dysfonctionnements de leur réseau. Des scénarios utilisant les protocoles du réseau domestique dans des applications d'administration sont décrits dans plusieurs travaux, par exemple [NPD07].

4.1.2 Communication interpersonnelle

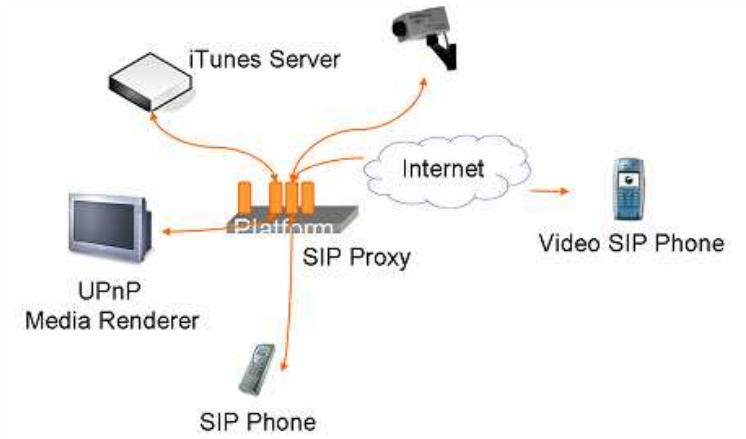


Figure 10 Terminal de télécommunication éclaté sur plusieurs équipements

Alice communique avec Bob confortablement assise dans le salon. Grâce aux capteurs disséminés dans le salon et à une caméra adaptée à la visiophonie, le système de communication permet à Bob de voir Alice parler dans son canapé. Le téléphone perfectionné de Bob envoie aussi des images à Alice qui les voit directement sur son téléviseur. Durant la conversation, Alice souhaite faire écouter à Bob le dernier album de

musique qu'elle a acheté. Elle choisit à partir de son téléphone un contenu qui se trouve sur un serveur de stockage disponible sur le réseau domestique. Le flux musical est joué aux oreilles de Bob et d'Alice.

Le système de communication d'Alice est ici un terminal éclaté sur plusieurs équipements du réseau domestique (cf. Figure 10). Le système cache l'hétérogénéité des protocoles multimédia de la maison – la caméra IGRS, la télévision UPnP et le serveur de contenu iTunes – et connecte les flux de ses équipements au téléphone SIP de son interlocuteur dans une application de communication interpersonnelle.

4.1.3 Administration d'équipements

L'application de diagnostique de Maxandre (cf. 4.1.1.3) est une application de contrôle à la frontière du domaine de l'administration d'équipements. L'outil de self-care de Maxandre pourrait être connecté à un serveur d'administration de sont fournisseur d'accès Internet au travers d'un protocole d'administration comme le protocole TR-069 du Broadband Forum. Cela permettrait au fournisseur d'accès Internet d'exécuter des batteries de test lors de problèmes relevés par le client.

Les pannes du système de Maxandre pourraient être de nature logicielle. Un des équipements du réseau domestique est mal configuré et l'outil de self-care permet la configuration correcte des équipements comme dans le scénario précédent. Si le problème provient d'un module logiciel qui a des dépendances de ressources insatisfaites, le système peut permettre la détection du problème et l'installation des modules logiciels manquant sur l'équipement pertinent.

L'administration d'équipements pure vise la gestion logicielle des plateformes d'exécution. L'auto-configuration est une fonction permise aujourd'hui par les systèmes comme celui du Broadband Forum. Les scénarios les plus innovants aujourd'hui visent la gestion de modules logiciels sur toute plateforme d'exécution d'applications modulaires du réseau domestique. Cette fois, le système masque l'hétérogénéité des technologies de plateformes existantes, par exemple les nombreuses distributions Linux, la plateforme OSGi, la technologie .NET, le chargeur d'application Java MIDP (cf. Chapitre III.3.3).

4.2 Techniques de médiation

Afin d'adresser l'hétérogénéité de technologies protocolaires ou programmatiques, il est naturel d'emprunter des techniques de médiation. La médiation est l'action de jouer l'intermédiaire entre deux interlocuteurs aux langages ou à la culture distincte. En informatique, la médiation est la transformation soit d'un langage dans un autre, soit d'un protocole dans un autre ou encore d'une interface programmatique dans une autre afin de permettre la communication entre plusieurs programmes qui ne sont pas interopérables.

4.2.1 Passerelles et pivot commun

Lorsque seuls deux protocoles (ou langages ou interfaces programmatiques) distincts existent, il est naturel d'utiliser une passerelle afin de transformer les appels effectués dans un protocole en appels dans l'autre protocole. Dans les applications de contrôle d'équipements, il est possible d'élaborer des passerelles entre un protocole de découverte et un autre. Indiss [BrI05] démontre la faisabilité d'un tel système. Tous les messages de découverte émanant d'équipements SLP sont réémis par la passerelle en UPnP et vice-versa. Les passerelles Internet sont un autre exemple, elles retranscrivent les adresses locales des équipements domestiques en l'adresse publique de la passerelle dans les messages de requêtes sortantes et permettent la retranscription inverse pour les messages de retour. Cela permet de générer un nombre d'adresses IP locales important sans occuper les bandes d'adresses Internet qui sont comptées tout en permettant aux équipements locaux d'accéder à des informations sur

Internet de manière transparente. JNI est un autre exemple de passerelle : JNI est une librairie Java permettant à un programme Java d'inter-opérer avec des librairies écrites dans un langage natif. L'article [YSHT07] montre un exemple de pont entre interfaces programmatiques. Le système décrit enregistre les objets Xlets d'une plateforme (Java) MHP dans le register de service OSGi tandis que les services OSGi sont accessibles dans le repertoires iXC de la plateforme MHP.

Lorsque les protocoles (ou langages ou interfaces programmatiques) deviennent nombreux, il devient plus efficace d'élire ou d'élaborer un protocole (ou langage ou interface programmatique) commun. En effet, comme le montre de nombreux travaux (MSDA [RCCI05], une architecture de pilotes Jini [Vay01], etc.), le nombre de passerelles à élaborer lorsque un nombre n de protocoles existe augmente en $O(n^2)$ – le nombre total de passerelles entre n protocoles est $n(n+1)$ – alors que l'élaboration d'un protocole commun entre n protocoles ne nécessite qu'une transformation de chaque protocole en le protocole commun, soit n transformations. C'est ainsi que MSDA [RCCI05] propose un protocole commun afin de composer les réseaux d'équipements aux protocoles divers : SLP, UPnP, Jini et quelques autres. Le consortium Salutation a suivi une approche similaire durant quelques années au tournant du siècle (cf. Chapitre III.1.1.7). C'est ainsi que la spécification JBI (Java Business Integration) propose une API commune afin que des applications disparates puissent communiquer entre elles. Les ESB (Enterprise Service Bus) proposent un bus à messages commun afin de permettre la communication entre des applications qui ne sont pas écrites pour être interopérables au préalable. La médiation vers un pivot commun est aussi l'approche prise par les travaux finaux de cette thèse nommés Home SOA (cf. Chapitre V.4) pour le contrôle d'équipements domestiques.

4.2.2 Approche protocolaire et approche programmatique

Le pivot commun entre moyens de communication distincts (langage, protocole, interface programmatique) peut être de nature protocolaire ou programmatique, c'est-à-dire décrite dans un langage de programmation. Lorsque plusieurs applications sont distribuées et qu'aucun protocole n'existe, il est naturel d'élaborer ou d'élire un pivot de nature protocolaire. Ce pivot protocolaire commun nécessite une réécriture partiels des clients et serveurs afin qu'ils communiquent dans ce protocole. C'est une approche prise par les protocoles indépendants des langages programmatiques en général. Elle est suivie par

- par la plupart des protocoles de contrôle d'équipements qui lient des applications embarquées sur des équipements hétérogènes (cf. Chapitre III.1 pour la définition de protocoles de contrôle d'équipements),
- par les protocoles d'administration d'équipements qui permettent à des serveurs d'administrer des équipements hétérogènes (cf. Chapitre III.3 pour la définition des protocoles d'administrations, cf. Chapitre V.6 et Chapitre VI.6 pour l'élaboration du protocole UPnP DM)
- les ESBs et les Web Services qui lient des applications disparates,
- etc.

Lorsque plusieurs protocoles existent, il est aussi possible d'élire ou d'élaborer un protocole commun Toutefois, ce moyen de médiation protocolaire s'avère souvent peu efficace : premièrement, il induit des retransmissions d'information dans un protocole distinct, ce qui augmente la bande passante nécessaire et qui allonge le temps de communication entre deux applications aux protocoles distincts (2 protocoles et le protocole pivot) par rapport à une communication directe entre deux applications communicant avec le même protocole (seulement un protocole) ou même au travers d'une passerelle (seulement 2 protocoles). Deuxièmement, sa pertinence dépend de la sémantique des applications utilisant les protocoles différents. Si la sémantique des applications est très proche, par exemple celle

d'applications de messagerie instantanée textuelle qui utiliseraient des protocoles différents, l'une Windows Messenger et l'autre le protocole Jabber, les passerelles protocolaires seront simples et ne nécessiteront aucune modification des applications afin d'inter-opérer. Lorsque la sémantique est variable, par exemple dans les applications de contrôle d'équipements où les types d'applications sont divers, la simple réémission des messages d'un protocole dans un autre génère des interférences sans aucune interopération. C'est le cas des architectures proposant un protocole pivot entre protocoles aussi divers que SLP, UPnP, Jini sans cibler d'applications. Ces protocoles étant très génériques, un client Jini ne pourra inter-opérer avec une télévision UPnP que si le client est une application multimédia et que le pont est spécifique à l'application de jeu de flux sur une télévision. Si le pont protocolaire n'est pas spécifique, une des deux parties (client ou serveur) devra être réécrite afin de communiquer avec l'autre partie. C'est probablement la raison de l'échec du standard Salutation par exemple (cf. Chapitre III.1.1.7).

Dans de nombreux cas, il est plus pertinent d'élaborer des interfaces programmatiques communes. Le pivot commun est alors effectué au niveau programmatique. REMMOC [GBS03] est un modèle pionnier dans la médiation de protocoles de découverte au travers d'interfaces programmatiques communes. OSGi [OSGi05] est un des standards qui s'est fondé sur une approche programmatique, sur cette vision que le contrôle d'entités hétérogènes nécessite des techniques au niveau langage. Home SOA (cf. Chapitre V.4) affine l'approche dans le cadre du contrôle d'équipements dans les réseaux pervasifs. Certaines équipes comme celle de Keith Edwards [NTKR06][KNS01] proposent aussi une approche programmatique mais avec un grain plus fin de composition qui la fait appartenir au domaine des approches sémantiques. Dans tous les cas, le patron Adapteur [GHJV95] est utilisé afin de représenter des objets existants selon les interfaces d'un pivot commun.

4.2.3 Approche syntaxique et approche sémantique

La différence majeure entre les deux approches tient en la définition du grain de transformation. Les approches usuelles considèrent le type du service, parfois même le type associé à l'ensemble des services d'un équipement comme grain de comparaison et de transformation. La médiation établit des transformations entre types reconnus de manière syntaxique.

Un courant théorique plus marginal mais en expansion aujourd'hui considère que le grain de comparaison et de traduction doit être plus sémantique. Les approches dites sémantiques reposent en finale évidemment sur des comparaisons syntaxiques mais la sémantique ressort de comparaisons syntaxiques de grain plus fin : le nom de chaque opération et type des arguments d'entrée et de retour sont considérés. Si les signatures d'opérations sont proches, cela donne des indices sur la proximité sémantique des opérations. Grace à ce grain plus fin, les chances de substitutions d'opération par une autre sont plus grandes que la substitution d'un service par un autre avec l'approche syntaxique usuelle. L'équipe de Keith Edwards [NTKR06][KNS01] œuvre sur des systèmes de recombinaison d'opérations identifiées par les types de données d'entrée et de sortie. Les comparaisons sémantiques peuvent aussi reposer sur des équivalences sémantiques indiquées par des sortes de dictionnaires appelés "ontologies". La souplesse des équivalences sémantiques montre toutefois rapidement ses défauts dans les applications concrètes où la rigueur des approches syntaxiques est préférée. Cette thèse a été l'occasion d'une preuve de concept d'une approche sémantique (cf. Chapitre V.4.4).

4.2.4 Génération semi-automatique et automatique d'adapteurs

Grâce aux techniques de passerelles et d'approches de médiation protocolaire, le nombre d'entités contrôlables – les services d'une architecture orientée service – dans un même protocole augmente. Ces situations augmentent l'ampleur du problème d'hétérogénéité sémantique visible dans tout environnement ouvert à des applications développées par

différentes parties. Dans ces environnements, il s'avère que de nombreuses interfaces sont sémantiquement proches mais syntaxiquement incompatible. Le type "imprimante" peut par exemple être trouvé en plusieurs exemplaires sous des formes syntaxiques différentes. Ce phénomène est parfois nommé "fragmentation d'interfaces" [Vay01][Law00]. D'autres travaux [PoF03] montrent que les adapteurs peuvent être qualifies "à perte", c'est-à-dire générer des pertes en nombre et en qualité des opérations et ne pas fournir totalement les fonctionnalités attendues. Ces travaux proposent une approche d'évaluation de ces pertes.

Différents travaux tentent de générer automatiquement ces adapteurs. Des méthodes de génération semi-automatiques sont proposés dans quelques projets. Elles permettent à l'utilisateur – ou l'administrateur – d'indiquer au système qu'une application est compatible avec une autre malgré une incompatibilité syntaxique apparente. Grâce à l'analyse de l'utilisateur, le système peut alors tenter de générer un adapteur après introspection des interfaces requises par l'application cliente et les interfaces fournies de l'application serveuse. Si l'adaptation n'est pas simple et que l'utilisateur possède quelques connaissances techniques, le système peut encore proposer à l'utilisateur d'indiquer quelles méthodes semblent sémantiquement proches et, plus loin, lui proposer un panneau graphique pour la programmation de conversion de types d'arguments [KeB03]. La génération d'adapteurs semi-automatique repose ainsi sur différents degrés de compétences de l'utilisateur pour l'adaptation du système.

La génération automatique d'adapteurs repose sur des techniques algorithmiques et sémantiques avancées. Certaines constructions algorithmiques peuvent rapprocher et adapter des interfaces dont les signatures de méthodes sont proches par leurs types d'entrée et de sortie [Tha94]. L'héritage de l'orientation objet et le polymorphisme des méthodes sont des critères d'analyse. Ces techniques logiques ne tiennent toutefois pas compte des noms de méthodes et d'arguments. Elles peuvent être associées à des techniques sémantiques. Les ontologies, une sorte de dictionnaires de correspondances de noms et de types à tout niveau, permettent dans quelques cas simples d'analyser et d'adapter automatiquement des interfaces syntaxiquement incompatibles. Ces techniques sont gourmandes en ressources et leurs résultats sont souvent limités, l'objectif étant ambitieux.

4.3 Exigences techniques

4.3.1 Exigences générales de l'Informatique Pervasive

La flexibilité est l'exigence technique principale face à l'ouverture des environnements pervasifs. La flexibilité visée doit permettre aux architectures déployées d'accepter rapidement et sans heurt des nouveaux protocoles à la demande tout en gardant des applications en cours de fonctionnement. La flexibilité concerne aussi l'intégration d'équipements aux fonctions sémantiquement pertinentes découverts avec les pilotes présents. Des exigences techniques précises découlent de cette demande générale de flexibilité :

- **Couplage lâche** : Les applications bâties sur l'utilisation d'équipements disponibles dans un environnement ouvert doivent être faiblement couplées avec ces équipements. De la lâcheté du couplage dépend l'évolutivité de l'application à l'exécution et au développement d'améliorations futures.
- **Interfaces uniformes**. L'hétérogénéité des protocoles doit être masquée au développeur. Celui-ci doit pouvoir aisément utiliser les équipements les plus adaptés sans connaître profondément les interfaces réseaux de chaque technologie. Cela rejoint l'exigence technique de **séparation des préoccupations** générale : la logique de l'application doit être séparée de la gestion générique totale ou partielle de certains aspects par le système

environnant – dans le cas des travaux de cette thèse, les trois aspects sont répartition, hétérogénéité, dynamique.

- **Chargement logiciel à la demande**. La gestion d'un nouveau protocole ou d'un nouvel équipement ne doit idéalement arrêter le fonctionnement d'aucune application. Cela implique des propriétés de chargement dynamique de code ou de fonctionnalités par le système. Dans un système moins performant, l'ajout de fonctionnalités au redémarrage est aussi possible et moins couteux. Et si le système ne possède pas de telles propriétés, la demande de modularité peut être comblée par l'installation de passerelles protocolaires. Toutefois, comme il est montré dans la section 4.2.2, les passerelles protocolaires s'avèrent couteuses ou peu pertinentes dans certaines situations.
- **Découverte protocolaire réactive**. L'entrée et le départ d'un équipement doit être notifié sur la plateforme de composition. Un équipement apportant des fonctions requises mais qui est accessible selon un nouveau protocole déclenche le processus d'installation du pilote pour celui-ci.

4.3.2 Communication interpersonnelle

Les scénarios de communication personnelle enrichie (cf. Figure 11) exposent des exigences techniques particulières. L'exigence suivante concerne l'hétérogénéité :

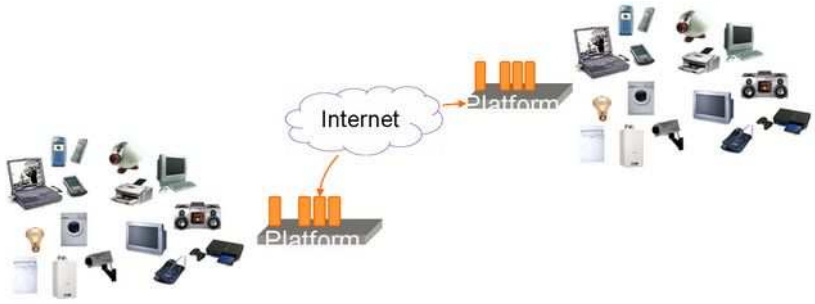


Figure 11 Communication interpersonnelle entre deux réseaux domestiques

- **La négociation et l'adaptation de flux initiés par les applications de contrôle**. Afin de faire inter-opérer les équipements du réseau domestique avec les terminaux de communication interpersonnelle, les applications doivent non seulement faire face à l'hétérogénéité des protocoles de contrôle mais aussi négocier et parfois adapter les formats et protocoles de flux multimédia selon les cas. La situation typique est celle des équipements UPnP/DLNA qui utilisent exclusivement le protocole de flux HTTP alors que les équipements SIP utilisent exclusivement RTP. UPnP/DLNA est le standard principal des équipements multimédia de la maison tandis que SIP est majoritairement accepté comme le protocole de la communication interpersonnelle sur IP.

4.3.3 Administration d'équipements

Deux objectifs distincts sont rencontrés sur le thème de l'hétérogénéité des protocoles du réseau domestique pour l'administration. Premièrement, combler le besoin de pont entre un protocole d'administration tel TR-069 [TR6904] et les protocoles de contrôle tel UPnP, ce qui est l'objectif de travaux comme [NPD07]. Cela permet de tester les services domestiques existants à partir d'un serveur d'administration. Ce besoin est similaire à l'exigence d'**interfaces uniformes** afin de représenter tous les équipements quels que soient leur protocole de contrôle. Cette fois, le protocole commun est celui du protocole d'administration. L'enjeu sera alors de spécifier un modèle de conversion entre chaque protocole domestique et le protocole d'administration.

Le deuxième objectif est de définir une interface générale d'administration pour les équipements en général. C'est l'objectif notamment du Comité de Travail UPnP Device Management (ex-Execution Platform) [2007-9] (cf. Chapitre V.6). Cette fois, le besoin est de définir un protocole qui s'appliquera de manière uniforme sur chacune des technologies. Les fonctions doivent pouvoir s'adapter à chaque plateforme à l'implémentation.

4.4 Etude de diverses passerelles

4.4.1 Contrôle d'équipements

Indiss [BrI05] est un projet de la même équipe que MSDA (cf. section 4.5.1). Il partage la vision que l'interopérabilité peut être apportée par l'édification externe de passerelles pour l'interaction entre fournisseurs et demandeurs de service déjà existants. L'architecture d'Indiss permet la traduction de messages émanant d'équipements SLP en messages UPnP et vice-versa.

4.4.2 Administration d'équipements

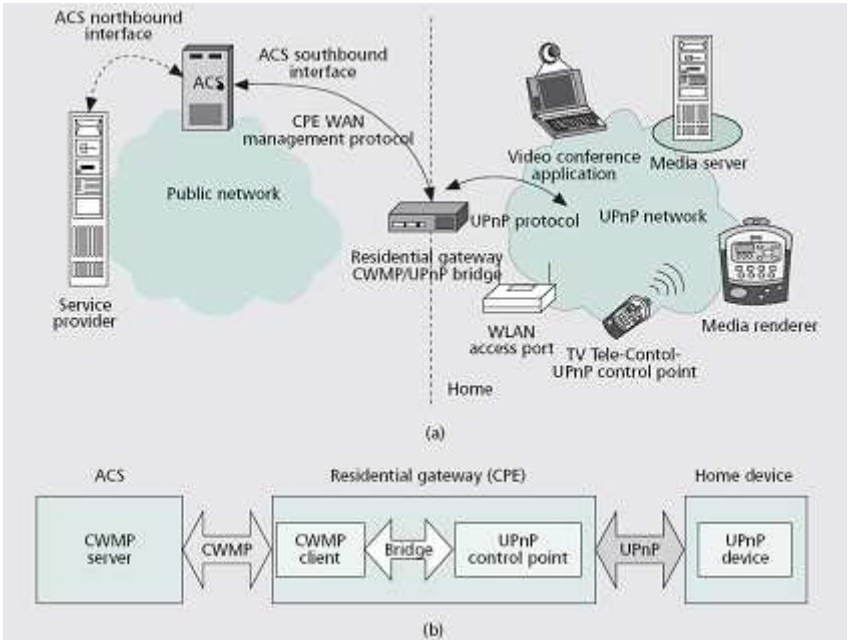


Figure 12 Passerelle et adaptation logicielle entre un protocole d'administration et un protocole de contrôle d'équipements [NPD07]

[NPD07] délivre les clefs de la spécification d'une passerelle entre le protocole d'administration TR-069 du Broadband Forum (cf. Chapitre III.3.2.1) et le protocole de contrôle UPnP (cf. Chapitre III.1.1.1). L'intérêt est ici le contrôle d'équipements standard UPnP par un serveur d'administration sur Internet. UPnP ne proposant que peu de profils d'équipements tournés vers l'administration (la publication du travail du Comité UPnP Device Management n'est attendue qu'en 2009), la passerelle est relativement peu utile pour les besoin d'administration actuels (auto-configuration et gestion du cycle de vie logiciel). Il permet toutefois de tester les fonctions des applications UPnP et de paramétrer les équipements établissant des règles de QoS via UPnP.

Les protocoles d'administration définissent des opérations de manipulation à distance d'un modèle de données hiérarchique. Le pont entre les deux protocoles consiste par conséquent en la représentation des équipements UPnP découverts par un point de contrôle dans un modèle de données à la syntaxe définie par une spécification du Broadband Forum, le TR-106. La gestion de ce modèle de données est commune avec un client TR-069 (cf.

Figure 12). Comme les événements UPnP peuvent être très fréquents, la passerelle spécifiée par [NPD07] possède aussi des fonctions de filtrage.

4.5 Etude de diverses approches protocolaires

4.5.1 Contrôle d'équipements

MSDA [RCCI05] est un intergiciel permettant l'interaction transparente d'applications préexistantes exposant et requérant des services dans des protocoles distincts. Chaque réseau est administré par un Gestionnaire (Network Manager) lié à autant de modules de découvertes que de protocoles pouvant être utilisés sur le réseau comme UPnP, SLP… Ces entités relaient les évènements de découverte et les appels protocolaires entre modules de découverte sur un même réseau et des ponts permettent de les relayer entre les réseaux. La vision introduite par MSDA est d'apporter de l'interopérabilité sur le réseau sans toucher aux logiciels déjà déployés. Tous les fournisseurs et demandeurs de services préexistent et sont développés avec leurs protocoles propres. L'architecture MSDA agit telle une passerelle entre les réseaux topologiques et entre les technologies de découverte. La transparence repose sur l'utilisation massive des capacités réseaux. Plus qu'une passerelle entre 2 protocoles, MSDA établit toutefois un protocole pivot afin de réduire le nombre d'adaptateurs à implémenter.

4.5.2 Contrôle à distance d'équipements à partir d'Internet

Les travaux d'une équipe de Telcordia dans les années 2000 et 2001 [TMM01][MMT01][MMG02] – nommément Simon Tsang, Stan Moyer and Dave Marples – montraient l'intérêt et la faisabilité de l'utilisation de SIP pour le contrôle à distance d'équipements domestiques. SIP est initialement spécifié pour des applications de communication interpersonnelle. Selon ces travaux, SIP apporte les niveaux nécessaires de sécurité, de robustesse, de passage à l'échelle, d'indépendance vis-à-vis des protocoles du réseau domestique et de nommage pour un objectif de passerelle entre équipements du réseau domestique et les terminaux du réseau Internet. Toutefois, le pont proposé nécessite une extension du protocole SIP [TMMS00]. Les URLs de SIP sont jugées pertinentes pour adresser les équipements domestiques localisés tandis qu'une nouvelle méthode – "DO" – et un nouveau type MIME appelée DMP [KGD00] (Device Message Protocol) sont proposés pour le passage de commandes de contrôle.

En réalité, au lieu de permettre à des terminaux SIP et des équipements domestiques existants d'inter-opérer, ces travaux inventent un nouveau protocole de contrôle qui nécessite le redéveloppement des clients et serveurs de part et d'autre du lien de communication. En effet, la sémantique des applications visées est complètement nouvelle pour les clients SIP qui n'agissent que dans le domaine de la communication interpersonnelle et le nouveau type MIME proposé à la sémantique de commandes d'équipements n'est utilisé dans aucune application connue aujourd'hui. D'autres travaux reposent sur une proposition similaire de protocole hybride basé sur SIP [BLBK04][BKBL06].

Il parait évident que les protocoles utilisés aujourd'hui sur le réseau domestique sont techniquement plus à même de conquérir le marché du contrôle à distance. UPnP (Chapitre III.1.1.1) possède l'atout d'être le protocole le plus répandu sur le réseau domestique mais le désavantage de l'être peu sur des terminaux sur Internet. L'effort technique du Comité de Travail UPnP Remote Access (publication prévue en 2009) répondra en partie au besoin de transparence de l'accès distant aux moyens de découverte et de contrôle UPnP sur les réseaux locaux. DPWS (cf. Chapitre III.1.1.3) est moins répandu sur le réseau domestique mais possède l'atout de reposer sur la technologie des Web Services très répandue sur Internet aujourd'hui. [KuR06] est une étude qui précède la création du Comité UPnP Remote Access. Elle montre une approche similaire à celle de l'équipe de Telcordia avec un approfondissement technique sur la manière de représenter les équipements domestiques à

l'extérieur du réseau. Elle montre en particulier le besoin de représentation de mandataires. Elle étend le protocole SIP avec le protocole UPnP. L'usage des registrars SIP est probablement un atout que n'a pas la spécification UPnP Remote Access en cours.

4.5.3 Communication interpersonnelle

SSIP [CPH07] est une tentative de division d'une session SIP en plusieurs flux destinés à des équipements distincts. Le papier montre une discussion intéressante sur la comparaison des possibilités de mobilité offertes par les mécanismes REFER (Call Transfer Protocol, cf. Chapitre III.2.1.2) et 3PCC (Third Party Call Control, cf. Chapitre III.2.1.2). Les auteurs rejettent le dernier sur l'argument qu'il implique l'intégration d'une entité centrale dans l'architecture. Pourtant, cet équipement peut être premièrement opportunément mêlé à l'infrastructure SIP (une passerelle SIP est nécessaire sur le LAN, au moins pour la réécriture des adresses dans les messages entre LAN et WAN). Deuxièmement, cette activité centrale peut aussi être transférée par utilisation de la SIP REFER.

Ce choix de REFER par les auteurs dans des situations non adaptées les fait proposer une extension du protocole SIP. L'extension proposée est un nouvel en-tête dans tous les messages SIP : chaque méthode SIP (par exemple INVITE, cf. Chapitre III.2.1.1) est indiquée par cet en-tête comme restreinte à une partie de la session. De cette façon, un équipement SIP peut transférer une partie de la session à un autre équipement SIP par restriction d'un message REFER à un des flux de cette session. La session SIP se dédouble à chaque appel de REFER et devient multiple. Chaque interaction par flux est associée en finale à une session SIP entre les mêmes interlocuteurs. Cette approche est définitivement erronée puisqu'elle couple le protocole de signalisation au protocole de flux. Elle vient alors en contradiction avec les objectifs mentionnés :

- Le système ne peut pas adresser l'hétérogénéité des protocoles puisque chaque flux n'est dirigé que vers un équipement SIP.
- L'extension proposée vient modifier un protocole qui permet déjà l'implémentation des scenarios mentionnés de mobilité et d'éclatement de session. Les sessions multi-flux sont possibles grâce à l'utilisation de la négociation par le protocole SDP associé à SIP (cf. Chapitre III.2.1.1). La reconfiguration d'une session multi-flux est permise par la méthode RE-INVITE.
- Avec SSIP, l'appelé est gère de manière non transparente plusieurs sessions simultanée. Chaque équipement devient possède une partie du contrôle, ce qui complique l'évolution de l'appel. Le couplage est fort entre les équipements. Au contraire, le modèle SIP 3PCC sépare la gestion de la session centrale de la gestion locale effectuée de part et d'autre grâce à un patron de conception de Façade.

4.5.4 Administration d'équipements

SCOMO [SCOM08] (cf. la présentation détaillée du Chapitre III.3.3.5) vise la définition d'une interface générale pour le contrôle du cycle de vie de plateformes d'exécution variées. SCOMO est une spécification protocolaire indépendante de tout langage de programmation. SCOMO définit deux entités logicielles : le "Delivery Package" (DP) et le "Deployment Component" (DC). Ces concepts correspondent aux concepts d'installeurs et d'unités de déploiement. La seule dépendance spécifiée entre ces unités est le fait qu'un DP contienne un ou plusieurs DCs. Les deux entités sont décrites selon un ensemble de paires noms-valeurs.

Une première originalité enviable de SCOMO réside dans la définition d'un état inactif d'une unité de déploiement. Cela permet de montrer un état dans lequel l'unité n'est

pas utilisable, soit parce que ses dépendances de ressources ne sont pas résolues, soit parce que l'unité est sur le point d'être désinstallée.

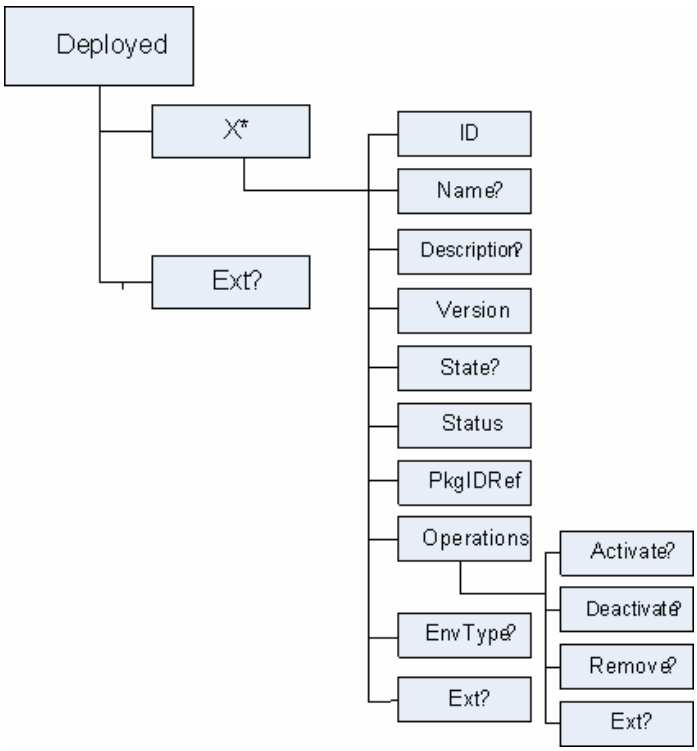


Figure 13 Modèle de données des composants de déploiement SCOMO

Une deuxième originalité, cette fois non naturelle, est que les actions sur le cycle de vie sont effectuées par des opérations sur les nœuds de l'arbre de données de l'équipement (cf. le nœud "Operations" de la Figure 13). Les opérations de téléchargement, d'installation, de désinstallation, d'activation et de désactivation sont des effets de bord d'opérations de configuration. Une spécification plus naturelle aurait défini des opérations sur l'interface de l'équipement. La spécification aurait gagné en lisibilité et en simplicité.

4.6 Etude de diverses architectures d'adaptation logicielle

L'adaptation logicielle est une approche du Génie Logiciel qui utilise les protocoles tels qu'ils sont et tente de les intégrer de manière opportuniste en insérant l'intelligence au sein des plateformes logicielles hébergeant les applications. L'approche Home SOA (cf. Chapitre V) fait partie de ces intergiciels qui prennent le meilleur de chaque protocole détecté afin de satisfaire l'activité de l'utilisateur.

4.6.1 Contrôle d'équipements

REMMOC [GBS03] est un modèle pionnier dans l'utilisation transparente d'intergiciels orientés services distincts. Tout protocole de découverte et de communication distante peut être pris en compte dans l'architecture modulaire de REMMOC. La reconfiguration dynamique de la plateforme en fonction des protocoles effectivement utilisés sur le réseau est l'aspect principal mis en avant dans ces travaux. Un module de l'architecture est responsable de la "découverte" des protocoles utilisés. La reconfiguration réagit sur les événements émis par ce module. La vision de l'interopérabilité est similaire à celle des travaux de cette thèse, nommée Home SOA : les fournisseurs et demandeurs de services sont écrits indépendamment des protocoles de découverte et de communication effectivement utilisés et une syntaxe de description générique joue le rôle de pivot commun. Cependant, l'adaptation dynamique de la disponibilité des services requis, la continuité entre

communication locale et distante, la nécessité d'une médiation technique plus riche ne sont pas adressées dans le projet REMMOC.

Les nombreux travaux au-dessus de la plateforme OSGi montrent la possibilité de masquer l'hétérogénéité des protocoles de découverte grâce à l'utilisation du registre de service OSGi et du service d'événements associé. Le registre de service OSGi implémente le patron de conception "Dynamic Service Locator" [Fow04] qui découple les applications des entités pervasives utiles à l'exécution. L'article [DFKM02] montre cette gestion partielle de l'hétérogénéité par l'utilisation de cette interface unique qu'est le registre de services OSGi.

4.6.2 Communication interpersonnelle

Les auteurs de [CCV06][VCC07] décrivent leur cadriciel qui éclate un terminal de communication interpersonnelle entre un téléphone SIP, un serveur de contenu et une télévision UPnP. L'architecture proposée suit une approche d'adaptation logicielle puisque les trois équipements formant le système de Vidéophonie sur IP suivent des interfaces standards : SIP User Agent, UPnP Media Server, UPnP Media Renderer. Toutefois, les équipements UPnP fournissent des flux RTP, ce qui est contradictoire avec la recommandation de facto pour le protocole de flux HTTP. Ils sont des prototypes inventés qui simplifient le problème. La gestion de l'hétérogénéité des flux n'est donc pas adressée. De surcroit, le système est écrit dans le langage C et ne montre aucune modularité logicielle nécessaire pour toute évolution future. Enfin, leur approche n'est pas générale face à l'hétérogénéité des protocoles multimédia et de communication personnelle puisque seuls deux protocoles sont utilisés : SIP et UPnP[1].

4.6.3 Hétérogénéité sémantique

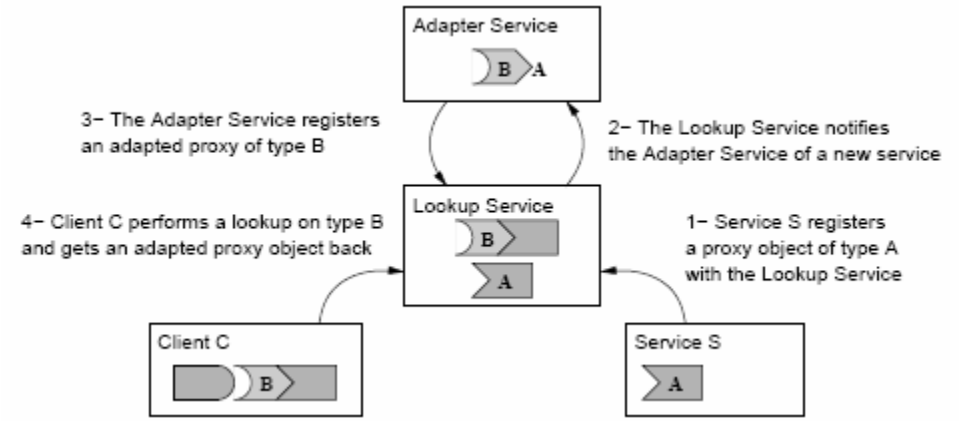


Figure 14 Service d'adaptation et répertoire d'Adapteurs [Vay01]

Les techniques d'adaptation logicielles sont utiles afin de faire face à la fragmentation d'interfaces [Vay01][Law00], c'est-à-dire de représenter les différents objets d'interfaces sémantiquement proches dans une interface commune. Afin d'adapter automatiquement les objets découverts, il peut être proposé un répertoire de multiples adapteurs simples [Vay01][PoF03]. Un gestionnaire présent sur une plateforme de services peut alors, dès qu'un service est enregistré sur la plateforme, enregistrer tous les adapteurs

[1] Au passage, l'article tente de montrer la pertinence d'une normalisation UPnP éventuelle d'un "VVoIP device" mais la définition de l'interaction s'avère incomplète : En effet, le "VVoIP Control Point" et le "VVoIP device" se trouvent sur la même plateforme et l'ensemble d'actions et d'événements protocolaires qui définissent cette interaction n'est pas définie avec détails.

dont l'interface d'entrée correspond à l'interface enregistrée. Ces adapteurs peuvent former des chaines d'adaptations permettant aux demandeurs insatisfaits pour un type de service d'utiliser les services d'un type sémantiquement proche. Dans [Vay01], la chaine d'adaptation est chargée et instanciée dynamiquement sur la plateforme et enregistrée comme nouveau service dans le registre de service Jini (cf. Figure 14). Cette technique est raffinée par Home SOA. Des outils de générations spécifiques et une tentative de génération d'adapteurs sémantiques sont aussi proposés dans la section Chapitre V.4.

Le répertoire d'adapteurs proposé par [Vay01] et [PoF03] nécessite le développement d'adapteurs pour tous les types de situations d'offre de services que peut rencontrer une application. Certains travaux proposent une génération semi-automatique d'adapteurs. [KeB03] propose par exemple un système de requête élaboré de services remplaçant celui de la plateforme OSGi. Lorsque celui-ci est appelé, il présente à l'utilisateur un panneau graphique qui lui permet de choisir les services à utiliser. Parmi les services choisis, si la correspondance syntaxique n'est pas directe, le système introspecte grâce à la réflexion Java les méthodes accessibles de ces services et du service requis, pour les présenter à l'utilisateur. Ce dernier peut alors indiquer les correspondances entre les méthodes. Si les types d'entrée et de sortie ne sont pas identiques, l'utilisateur doit alors développer les conversions de types. A la fin de ce travail, qui peut être très laborieux, l'outil génère alors les adaptateurs nécessaires.

L'équipe de Keith Edwards œuvre sur des concepts d'"informatique recombinante" [KNS01][NTKR06]. Elle a proposé récemment un cadriciel, appelée uMiddle, permettant d'établir des ponts programmatiques entre plateformes intergicielles définies comme étant du type d'UPnP, Bluetooth, Jini. uMiddle est une approche programmatique spécifiant un pivot d'interface commune et un grain fin de composition défini comme un type d'entrée et de sortie d'un service. uMiddle vise la composition d'entités réseau hétérogènes dans les environnements pervasifs. Trois contributions présentées dans l'article [NTKR06] :

- "Service Shaping" : c'est une technique de composition de services décomposés en ports (types d'arguments et de flux). Un service est défini par un ensemble de types de données d'entrée et de sortie. L'API d'uMiddle permet aux applications de chercher dynamiquement les services requérant ou fournissant des types d'arguments donnés.
- USDL (Universal Service Description Language) : USDL est un format XML pour la génération dynamique d'adapteurs. Il permet aux concepteurs de décrire leurs services dans le système de ports d'uMiddle. Des modèles USDL sont écrits pour les événements, les états et les actions UPnP par exemple. Cela permet à un traducteur UPnP générique de générer des mandataires uMiddle pour tout équipement UPnP.
- Un mécanisme dynamique de liaison d'équipements : chaque nouveau service apparaissant sur la plateforme uMiddle est décomposé en ports. Ces ports sont dynamiquement comparés et liés aux ports déjà présents sur la plateforme. La décomposition en types d'entrée et de sortie singuliers augmente les possibilités de compositions chainées.

5 SELECTION, SUBSTITUTION ET CONTINUITE DE SERVICE

Le caractère pervasif des environnements exacerbe le besoin des applications en dynamisme et de comportement autonomiques. Ces environnements sont caractérisés par la variabilité et la mobilité des entités actives environnantes. Composer dynamiquement des applications tout en garantissant la continuité de service aux utilisateurs est un défi important de l'Informatique Pervasive.

L'auto-organisation des équipements d'un réseau et la composition de fonctionnalités offertes par l'environnement dans des services de haut niveau à l'utilisateur sont les objectifs des projets de l'Informatique Pervasive. L'orientation service est à la base de plusieurs travaux de part l'adéquation entre l'ouverture des environnements pervasifs et le couplage lâche apporté par les approches à services. C'est pourquoi les tentatives d'atteindre l'auto-organisation se concentre souvent sur la recherche de méthodes de composition de services.

Les intergiciels protocolaires orientés service existants sont bâtis sur des protocoles de découverte de services, de description de services, de contrôle de services et offrent des fonctionnalités de qualité de service et de sécurité [ZMN05]. La sélection de services et le classement des services selon des critères contextuels ne sont toutefois pas complètement adressés par ces ensembles protocolaires. Les mécanismes nécessaires sont complexes en raison de la richesse et de la dynamique du contexte ouvert des environnements pervasifs tels que le réseau domestique. Ces taches de sélection sont le plus souvent l'objet d'une configuration manuelle. La sélection automatique de services est une tache difficile que de nombreux travaux tentent de traiter [BSD03][FMK06].

La composition de service au sein des applications pervasives est forcément dynamique afin de répondre à un environnement changeant. Des mécanismes de reconfiguration dynamique rendent possible la réaction adéquate aux stimuli de l'environnement. Dans les applications à service, il s'agit de délier et de lier à nouveau le demandeur de service au fournisseur le plus adéquat à l'instant de la reconfiguration. La reconfiguration dynamique d'application est ici considérée en tant que reliaison dynamique de service.

Cette reliaison dynamique pose souvent le défi de la garantie de continuité de service. Garantir la continuité de service est aisé dans le cas où les services utilisés n'ont pas d'état, c.-à-d. que chaque opération appelée ne dépend pas des précédentes. Dans le cas de services à états, l'état du service utilisé précédent doit être passé au service nouvellement lié durant la reliaison. La complexité du passage d'état tient de la spécificité de l'état de l'application au type d'application donné.

Cette partie tente d'aborder les problèmes intergiciels auxquels sont confrontés les concepteurs d'applications attentives au contexte. Avant d'adresser la reliaison dynamique de service, le "contexte" est défini dans les applications pervasives. La littérature scientifique est emplie d'approches qui définissent et décrivent le contexte applicatif de manière générale. Ces définitions sont une première donnée pour ce travail. Une deuxième donnée provient des nombreux travaux sur les approches composants et leur cycle de vie. Ces travaux permettent d'appréhender la place du transfert d'état durant la composition.

La solution idéale visée dans cette thèse est un système qui se reconfigurerait de manière proactive. La reliaison proactive de service vient de l'idée que la composition de service puisse être changée par le système durant l'exécution par la seule connaissance des facteurs contextuels et sans action explicite de l'utilisateur. Le système est proactif lorsqu'il prend des décisions de reliaisons en prévoyant le transfert d'état nécessaire. Le système décide qu'un autre fournisseur de service est meilleur que le fournisseur courant remplace ce dernier par le premier en maintenant la continuité de service.

Une définition du "contexte" est présentée dans la section suivante. Des scenarios intéressants démontrent ensuite le besoin de reconfiguration dynamique dans les applications pervasives. Enfin, une section décrit les exigences techniques nécessaires pour l'automatisation de la description du contexte applicatif, de la sélection de services conséquente et de la reliaison durant l'exécution.

5.1 Le contexte

L’attention au contexte (context-awareness) est un thème de recherche partagé par un grand nombre de travaux tant dans le Génie Logiciel que dans les sciences humaines. Anind K. Dey est probablement l’auteur le plus cité de la littérature technico-scientifique associée.

Dey [DSA01] définit le contexte, c.-à-d. le contexte applicatif, selon trois catégories : les personnes, les objets, les lieux. La définition tripartite suivante du contexte est similaire :

- Le contexte de l’utilisateur : activité, localisation, préférences, liens sociétaux, description physique et médicale, humeur, etc.
- Le contexte du réseau d’équipements : présence, localisation, paramètres réseaux, fonctionnalités, qualités de service disponibles, etc.
- Le contexte physique : lieu, date, paramètres météorologiques, etc.

5.1.1 Le contexte de l'utilisateur

La satisfaction de l'utilisateur est souvent l'objectif premier des applications dans les environnements pervasifs tels que le réseau domestique. Satisfaire l'utilisateur est l'assister dans ses activités avec le service le plus approprié. Afin qu'un système puisse décider quel service est le plus approprié, le système nécessite la connaissance de paramètres dont la pertinence varie selon l'application.

L'application de messagerie ambiante est une des applications pervasives les plus représentatives [CSLB06] (cf. section 5.2.1). Cette application nécessite la connaissance de l'activité de l'utilisateur afin de lui procurer le service du niveau d'attention souhaité. La localisation dans la maison ou à l'extérieur est un des critères les plus importants afin que l'utilisateur puisse suivre les échanges à l'endroit où il se trouve. Les préférences de l'utilisateur influent sur le choix du meilleur équipement de restitution et le choix du rendu sensoriel. Les liens sociétaux permettent de donner des priorités d'une activité par rapport à une autre et des règles pour le partage du système par les utilisateurs. Par exemple, si l'utilisateur laisse entrer un voisin qu'il connaît peu, le système pourra supprimer l'affichage des échanges de la messagerie ou si l'utilisateur accueille un ami connaissant l'interlocuteur de la messagerie, le système pourra inviter l'ami dans la conversation et gérer ce second utilisateur comme le premier.

Ce contexte repose sur la remontée d'informations captées par différents équipements. Le clavier et la souris peuvent être d'excellents capteurs de contexte dans des applications classiques où ils sont utilisés. Dans une vision puriste de l'Informatique Pervasive – voire de l'Informatique Ambiante – ces interfaces classiques ne doivent pas être les interfaces d'entrées principales puisqu'elles sont l'exemple des interfaces génériques, localisées et complexes d'aujourd'hui. Afin que la remontée d'information soit plus "ambiante", l'information doit être captée par des équipements complémentaires : capteurs de pression, interfaces tactiles, micros, cameras, etc.

5.1.2 Le contexte de l'équipement

Le paramètre le plus important à prendre en compte dans un système composant les fonctions de plusieurs équipements est la présence de ceux-ci. Dans un système distribué, la possibilité que les équipements puissent ne pas être accessibles rend nécessaire des mécanismes détection de panne et de reconfiguration dans les applications. L'Informatique Pervasive prend pour hypothèse que toute entité peut se connecter et quitter le réseau à tout moment. Des protocoles de découverte complètent alors les mécanismes applicatifs afin de renseigner l'application sur l'état de présence des équipements [ZMN05].

Ensuite, les paramètres utiles pour la composition applicative sont le type de fonctions et de protocoles que fournit l'équipement. Ceux-ci ne varient habituellement pas au cours de l'utilisation de l'équipement, contrairement aux paramètres de présence et aux autres paramètres techniques comme ceux de qualité de services, de réseau, etc. Les paramètres mesurant la qualité de service – bande passante utilisée, niveau d'utilisation des capacités de calcul, etc. – sont l'objet de négociation de la qualité du service que doit fournir l'équipement à l'application demandeuse.

5.1.3 Le contexte physique

Le contexte physique comporte les paramètres qui n'appartiennent à aucun utilisateur et à aucun équipement. Les paramètres caractérisant les lieux où se situent l'action, les paramètres météorologiques, les objets non-électroniques, etc. peuvent être des informations pertinentes pour caractériser des situations applicatives.

5.2 Scénarios

Plusieurs scénarios illustrent les objectifs de reconfiguration dynamique. Ils s'appliquent à des situations où le service proposé à l'utilisateur est une composition de fonctionnalités procurés par des entités présentes dans l'environnement.

5.2.1 Scénarios domestiques

Les applications de Suivez-Moi (Follow-Me) audio-vidéo consistent à changer dynamiquement de lecteur multimédia au fur et à mesure que l'utilisateur change de position dans la maison (cf. Figure 15). [KiN05] définit une architecture simple d'application attentive à la localisation de l'utilisateur. Un point de contrôle lie une source auditive donnée au puits auditif – players de la Figure 16 – le plus proche de l'utilisateur. Les auteurs ont inventés leurs propres interfaces protocolaires pour les puits et les sources multimédia alors que le standard UPnP AV [UAV02] connaissait déjà un certain succès sur le marché des équipements électroniques.

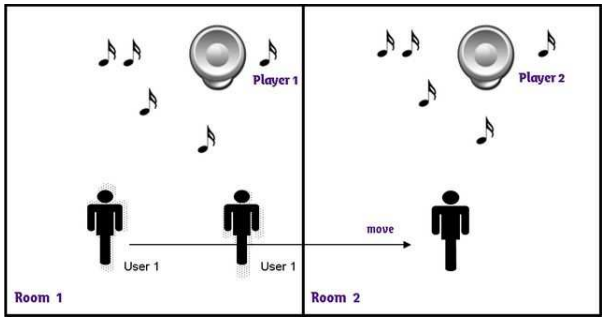


Figure 15 Une radio qui suit l'utilisateur lorsqu'il change de pièce

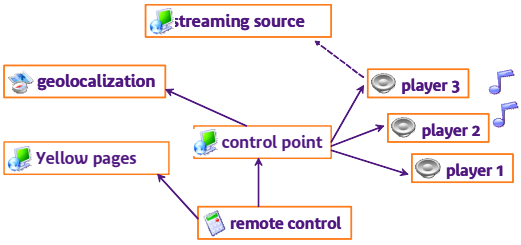


Figure 16 Architecture d'une radio qui suit l'utilisateur dans la maison

Une application pervasive plus complète est la messagerie ambiante, c'est-à-dire la messagerie instantanée attentive au contexte. La messagerie ambiante adapte dynamiquement

les services d'entrée-sortie – clavier, souris, écran, haut-parleurs, microphone – les plus adéquats à l'activité de l'utilisateur. Lorsque ce dernier est assis devant la télévision, le système redirige la communication sur une partie de la télévision en utilisant un système Picture-Into-Picture. S'il fait une sieste sur le canapé, l'état des communications lui est transmis sous forme d'indications lumineuses et colorées sur une lampe ambiante. S'il vient à prendre une boisson dans le réfrigérateur, le système abandonne la télévision pour diffuser les messages écrits de ses interlocuteurs par les haut-parleurs via un système de diction de texte (text-to-speech) si aucun écran n'est disponible dans la cuisine. Lorsqu'il vient à son bureau, la messagerie instantanée lui est à nouveau visible par des interfaces habituelles.

La complexité de ces applications de communication ambiante se mesure à la variété des interfaces d'entrée-sortie gérées par le système et à la variété des situations que reconnaît le système [CSLB06]. La conception de la messagerie instantanée devient encore plus difficile si l'on ajoute une adaptation dynamique au nombre et à la qualité des utilisateurs présents. D'autres travaux montrent des scénarios domestiques simples, par exemple un système de suivi de taches quotidiennes et d'alerte de l'utilisateur en cas de situations dangereuses [VFL05].

5.2.2 Les décisions de reliaison proactive dans ces scénarios

Dans ce type de scénarios, les décisions du système sont des réactions aux changements de localisation et de la propension de l'utilisateur à communiquer.

La localisation de l'utilisateur peut être représentée sous des formes différentes selon l'application, par exemple une pièce d'un bâtiment ou la position géométrique dans un référentiel donné. La localisation des équipements suit aussi cette logique. La forme et l'orientation de ceux-ci ont aussi leur importance. En effet, un écran est dans le voisinage de l'utilisateur si l'utilisateur est localisé dans une zone définie devant lui. Les niveaux d'adéquation se mesurent dans ce cas par la distance de l'utilisateur à l'écran, l'orientation de celui-ci et la capacité visuelle de l'utilisateur.

La propension à communiquer de l'utilisateur est un paramètre-clef que tentent de mesurer les systèmes de messagerie ambiante par la détection de l'activité de l'utilisateur et par raisonnement sur ces préférences. Malheureusement, ce critère est très difficile à mesurer. Il n'est effectivement pas une conséquence directe des actes de l'utilisateur. Ce critère ne peut être évalué aujourd'hui que par des informations partielles. Le service IBM Grapevine [CSLB06] est un exemple de messagerie ambiante. Ces concepteurs notent de manière éloquente qu'il existe un fossé sémantique entre l'information que les capteurs de bas niveau et les programmes peuvent détecter et l'information de haut niveau qui caractérise la capacité et la propension de la personne à communiquer avec une autre personne.

Des conflits avec d'autres utilisateurs présents peuvent aussi intervenir. Les liens sociétaux entre les différents utilisateurs et la possibilité de coexistence des activités de chacun entrent dans la liste des préoccupations du système.

L'activité de communication débute lorsque le système est actif et qu'un utilisateur local ou distant tente d'entrer en communication par un des modes permis par l'application. Tant qu'elle est active, l'application évalue la localisation et la propension à communiquer de l'utilisateur grâce à un réseau de capteurs distribué dans la maison. Les capteurs de localisation peuvent être des caméras, des détecteurs infrarouges, des antennes wifi, des tapis RFID, etc. Les capteurs d'activité peuvent être des caméras, des surfaces sensitives dissimulées sur les portes et meubles que l'utilisateur touche fréquemment. Des capacités d'analyse importantes doivent être attachées à ce réseau de capteurs. Des règles complexes sont à définir afin d'évaluer l'activité et par inférence, la propension à communiquer de l'utilisateur.

Les événements pertinents provenant de la collecte de ces informations déclenchent la reconfiguration de l'application. Dans les situations visées par cette thèse, la reconfiguration est la liaison des meilleurs équipements d'entrée-sorties à l'application de communication vers l'extérieur. Dès que la localisation de l'utilisateur change au point qu'un équipement devienne plus pertinent qu'un autre, le système réagit pour lier l'équipement le plus proche. Dès que l'utilisateur change d'activité et qu'un seuil de propension à communiquer est franchi, l'équipement le plus adéquat remplace l'équipement en cours.

5.2.3 Autres scenarios

Les Environnements de Développement Intégrés (IDE en anglais) – comme le célèbre environnement eclipse – sont des sources de scénarios où de nombreuses taches de reconfiguration sont effectuées pour assister l'activité de l'utilisateur. Par exemple, eclipse définit la notion d'*espace de travail* (workspace) à partir d'une liste de documents visible selon un certain filtre et selon différents arrangements de fenêtres appelés *perspectives*. Toute perspective peut être considérée comme une composition de services, toute vue (fenêtre) comme un composant offrant des services. Le passage d'une perspective à une autre est alors une tache de reliaison. Un utilisateur d'eclipse tel qu'un développeur Java demande fréquemment le changement de vues et de perspectives afin de visualiser de la meilleure façon les projets à télécharger d'un répertoire SVN, les fichiers Java à éditer dans son projet de développement, les fichiers XML de configuration de son application, les étapes d'un déverminage pas à pas, etc. Une tache de reliaison intervient dès que l'utilisateur lance une action qui est vue d'une meilleure manière après un changement de perspective.

Posséder un système qui évalue et adopte le meilleur contrat d'accès téléphonique disponible à tout moment de son abonnement est un autre scénario qui demande la reliaison dynamique d'un client et d'un service. Le service recherché est qualifié selon des critères comme le prix, la durée d'abonnement minimale, etc. Les préférences de l'utilisateur sont la durée moyenne des appels passés dans les mois précédents, l'horaire habituel des appels passés, etc. [BFTZ05] décrit un tel scenario.

5.3 Techniques et définitions

La sélection de service consiste à sélectionner le fournisseur de service le plus adéquat selon les besoins du demandeur de service. Un service est décrit par un type – ou interface de service – qui identifie un ensemble d'opérations – aussi appelées méthodes ou actions – et par des propriétés – aussi appelées métadonnées – qui qualifient le service. Les propriétés de service sont des données contextuelles comme la localisation, la qualité des fonctionnalités offertes, les capacités disponibles, etc.

Les propriétés de services sont définies par des paires attributs-valeurs dans plusieurs technologies orientées services (OSGi [OSGi05], SLP [GPVD99], UPnP [UPnP06], Web Services [GDSD04] lié à ws-metadata-exchange et ws-policy, DPWS [DPWS06], IGRS [IGRS06], Apple Bonjour [StC05], Jini [Wal99], Salutation [SALU99]). Les services de la plateforme OSGi et des intergiciels protocolaires tels que SLP et UPnP se voient attachés un ensemble arbitraire de paires attributs-valeurs. WS-Policy définit une grammaire extensible pour exprimer les capacités du fournisseur de service et les exigences d'un demandeur de service. Un ensemble multiple de paires attribut-valeur et la représentation des exigences sous forme d'une fonction d'utilité sont des définitions communes dans le e-commerce (cf. [LAOS06] pour une liste de technologies).

La recherche de couplage lâche est l'explicitation des informations demeurant implicites entre un client et un serveur. Force est de reconnaitre qu'aujourd'hui, la plupart des architectures orientées service garde implicite le partage d'un même protocole de communication et d'une même sémantique métier. Seules les propriétés non-fonctionnelles du

service - par exemple, la QoS, la sécurité - sont l'objet de flexibilité apportée par la sélection de services pour un type de service recherché [CuM03]

5.3.1 Outils de sélection de services

Plusieurs concepts permettent des possibilités de sélection de service dans les intergiciels existant et dans la littérature scientifique :

- Un **domaine** (scope) définit le périmètre d'une requête de service. Les domaines sont habituellement définis selon des critères topologiques, selon le rôle de l'utilisateur, ou plus généralement selon le contexte applicatif [ZMN05]. Par exemple, les méthodes multicast limitent naturellement la découverte de service UPnP à une topologie de réseau encadrée par des switchs. Les domaines de la technologie SLP sont librement assignés par les fournisseurs de service à leurs services lors de leur enregistrement. Les demandeurs de service doivent mentionner ces domaines lors de la recherche comme premier niveau de filtre. L'utilisation d'une composition hiérarchique de domaines est promue par certaines architectures à composant [CeH04].
- Un **filtre** restreint l'ensemble des services disponibles selon les besoins du demandeur de service. Un filtre définit un seuil d'acceptation. Dans les architectures orientées service le filtre principal concerne le type de service – son interface. Un demandeur de service peut requérir des types de services précis pour une application donnée. Dans les modèles définissant des propriétés de service en tant que paires attribut-valeur [CeH04], l'opérateur d'égalité et de seuil sont les opérateurs les plus basiques. Une distinction est quelques fois indiquée entre **filtre obligatoire** et **filtre optionnel**. Un filtre obligatoire est toujours évalué avant la liaison de service et élimine les services qui ne le vérifient pas. Les filtres optionnels ne sont évalués que sur les services ayant déjà passé le filtre obligatoire et si le nombre de ces derniers excède le nombre de services requis. Ces filtres raffinent le premier afin de restreindre la sélection de services [FMK06].
- L'objectif d'une **fonction d'utilité** – aussi appelée fonction de score, fonction objective, etc. – est d'ordonner tous les services possibles dans un ordre total. Cette fonction évalue le degré d'adéquation des services fournis aux besoins du demandeur de service. Plusieurs approches de sélection de service sont basées sur la théorie d'utilité [KeN75]. Cette dernière définit les propriétés comme un ensemble de paires attribut-valeur qualifiant des ressources – ou services fournis dans le contexte de la composition de services. La fonction d'utilité est une somme pondérée d'inconnues. Les inconnues sont à remplacer par les scores d'évaluation des propriétés de chaque service potentiellement intéressant pour la liaison en relation avec les besoins du demandeur de service. La fonction d'utilité permet au demandeur d'ordonner les services fournis disponibles. Le fournisseur de service ayant obtenu le meilleur score est considéré comme celui qui optimise la composition [BSD03].
- Une **politique** définit des algorithmes de classement et peut être implémenté avec des fonctions d'utilité [LAOS06]. Une politique possède une signification très générale. Quelques définitions embrassent les mécanismes basés sur des règles comme les règles Evénement-Condition-Action. Par exemple, des politiques d'adaptation réactives sont définies dans l'article [DaL06].

5.3.2 L'application de fonctions d'utilité

Bâtir une solution de sélection de services sur des fonctions d'utilité revient à définir les deux modèles suivants :

- Une échelle de valeur commune afin d'évaluer les propriétés contextuelles attachées aux services à composer
- Un langage dans lequel programmer les fonctions d'utilité

5.3.2.1 Affectation de propriétés et évaluation de scores de services

L'évaluation de l'adéquation des propriétés des services disponibles en regard des besoins de l'application cliente est une tache spécifique du domaine d'application. Affecter un score aux valeurs des attributs exposés dans les descriptions de service dépend de la sémantique choisie pour définir le contexte applicatif. Par exemple, le prix d'un service peut être défini dans plusieurs devises différentes. Il peut être acceptable dans une fourchette de valeurs pour certaines applications et dans une autre fourchette de valeurs pour une autre application. C'est pourquoi les entités attachant les scores et les entités les évaluant partagent une même échelle de valeurs.

Distribuer cette tache d'évaluation sur tous les fournisseurs et demandeurs de services implique un couplage allant à l'encontre des principes de composition de services dans les environnements pervasifs. Ainsi, il est plus raisonnable qu'une infrastructure distincte des fournisseurs de services attache des scores à des représentations de services et évalue la fonction d'utilité sur ces représentations. Ces représentations sont habituellement enregistrées dans un registre de services dans les différentes technologies orientées services citées.

Il est naturel d'implémenter les méthodes de sélection de service dans les entités qui sont les demandeurs de service. Dans ce cas, ces entités seront couplées à l'infrastructure de sélection de service. Toutefois, dans une tentative de découplage, des techniques de génie logiciel permettent de séparer la logique métier de ces entités de la logique de sélection de services. [DaL06] montre un modèle à composant où les politiques de recomposition – ou reconfiguration – sont attachées au composant de manière extérieur à ceux-ci. Des règles implémentées dans un langage de script distinct du langage de programmation du composant sont affectées par l'infrastructure sur le composant à l'aide de méthodes de contrôle extérieures définies dans le modèle du composant. [2007-7] montre un travail similaire où la contextualisation des requêtes de service est effectuée par un contrôleur – appelé handler dans la taxonomie d'iPojo – du modèle de composants à services iPojo [EHL07].

5.3.2.2 Description des fonctions d'utilité

Les fonctions d'utilité peuvent faire appel à des fonctions complexes. [FSLM07] montre un exemple de définition de fonctions d'utilité dans une syntaxe XML complexe. [DaL06] utilise un langage de script appelé FScript défini par la communauté du modèle à composants Fractal [BCLQ06]. La complexité d'une fonction d'utilité peut requérir la puissance d'expression d'un langage de programmation. Les exemples suivants sont classés du plus simple au plus complexe. Les deux premières fonctions sont aisément décrites par utilisation de fonctions mathématiques usuelles. La troisième utilise une fonction décrite par un modèle sémantique spécifique. Les deux dernières font notamment référence à une constante déterminée par une valeur maximale, valeur qui dépend de l'application :

- Recherche d'un prix minimum après avoir filtré les prix inférieurs à 100 dollars avec le filtre suivant (&(price<100)(currency=dollar) :

$$(100 - price) / 100$$

- Recherche de la distance minimum de l'utilisateur à des équipements par utilisation de coordonnées géométriques :

$$(MaxDistance - (\sqrt{((x_{device}-x_{user})^2+ (y_{device}-y_{user})^2+ (z_{device}-z_{user})^2}))/ MaxDistance$$

- Recherche de la distance maximale entre l'utilisateur et les équipements par utilisation de coordonnées définies dans un langage ensembliste :

$$distance(room_{device} , room_{user}) / MaxDistance$$

5.3.3 Niveaux syntaxique et sémantique de sélection de services

Dans les architectures orientées services, le couplage entre fournisseur et demandeur de services est réduit au partage de la connaissance d'une interface de service. A des fins de description et de sélection, cette interface est définie non seulement par un ensemble d'opérations mais aussi par un ensemble de propriétés descriptives. La mention de ces propriétés permet au fournisseur de qualifier son service d'après des critères et une échelle de valeurs connue par le demandeur. Cette qualification particulière par le fournisseur peut alors être sujette à la sélection du demandeur.

Les moyens de sélection de service des technologies actuels reposent sur l'adéquation syntaxique exacte entre l'interface de service offerte par les fournisseurs et l'interface de service requise par le demandeur. La découverte, la description et la sélection de services de ces technologies sont dites syntaxiques.

De nombreux projets dénoncent la faiblesse des technologies syntaxiques au regard du défi que pose la composition de services sur le web et dans les environnements pervasifs. En effet, dans une vision jusqu'au-boutiste, ces environnements ouverts posent le défi de la mise en relation d'entités ne partageant aucune interface commune. Ils prennent par conséquent pour hypothèse que la sélection de service s'effectue sur des bases purement sémantiques. La recherche de services devient alors plus complexe puisqu'elle se doit d'évaluer la distance sémantique entre interfaces requises et fournies. Afin de répondre à des requêtes de services par des offres de service sémantiquement proche mais syntaxiquement différentes, de larges dictionnaires appelées ontologies sont élaborés et utilisés. Un groupe de travail important autour de la technologie WSMO [WSMO06] propose en particulier une vision du Web Sémantique pour la composition de Web Services [FRPD05]. Dans le e-commerce, Lamparter et al. proposent aussi l'utilisation d'ontologies [LAOS06]. Avec la promotion de ces approches, les concepteurs applications doivent élaborer un compromis nécessaire entre expressivité et efficacité du langage sémantique. L'utilisation de procédés sémantiques se doit d'être justifié lorsque les études montrent que la profondeur des graphes ontologiques a un impact exponentiel sur les temps de raisonnement [ABR06].

5.3.4 Service à état et continuité de service

Les applications sont dites "à état" lorsque leur comportement dépend d'un contexte d'utilisation passé et évoluant. Les applications dont la composition évolue selon le contexte et la disponibilité des services pose le défi de garantie de services. En effet, lorsqu'un service se substitue à un autre dans la composition, l'état éventuel du service substitué doit être transféré au nouveau service. Le transfert d'état intervient lorsque le service substitué est un service à état (stateful service), c'est-à-dire que l'état est détenu par le service utilisé. Afin de garantir la continuité de services, les applications peuvent mettre en œuvre des politiques variées qui dépendent souvent du domaine d'application et du contexte :

- Certaines applications critiques tentent d'éviter la recomposition de service si l'avantage apporté par la recomposition est relativement moins grand que les défauts de la composition actuelle et le coût de la recomposition. Les algorithmes de décision affectent un coût à la recomposition afin de n'opter pour celle-ci que dans des conditions acceptables. Dans le cas de compositions dirigées par une fonction d'utilité, un poids important sera donné au service en cours d'utilisation.

- Certaines applications tentent de placer les opérations de recomposition à un moment où le service n'est pas utilisé. C'est le cas du cadriciel d'OS embarqué Think développé par Juraj Polacovic [POS06] qui détecte les états saufs (safe state), état où la reconfiguration est la moins couteuse. Ces états correspondent à des états dormants, qui peuvent correspondre par exemple au moment où aucun thread n'est en cours de fonctionnement pour un système multi-threadé. Le coût de recomposition varie lui-même dans le temps.
- Certaines applications utilisent des services redondants afin qu'une recomposition faisant intervenir simultanément une dé-liaison et une nouvelle liaison n'ait pas lieu. Un cas particulier est l'utilisation de 2 services de restitution auditive lors du passage de l'utilisateur d'une pièce à une autre dans une application de "suivez-moi auditif" ("audio follow-me") [KiN05]. Une superposition de 2 systèmes sonores est mise en place aux changements de pièce dans ces applications afin que l'utilisateur ne perçoive pas de discontinuité sonore. Un autre cas d'application est celui d'une application web qui utiliserait simultanément 2 instances d'un serveur web au moment de mettre à jour la version de celui-ci. Dans ce cas, l'ancienne version demeure utilisée jusqu'à ce que les sessions clientes se terminent sur celui-ci alors que les nouvelles sessions ne sont initiées que sur la nouvelle version du serveur.

Le transfert d'état dans ces applications est effectué au niveau applicatif. Il peut être automatisé de manière générique si certains principes peuvent être suivis. Un état (ou une session) est le résultat d'opérations appelées sur un service. Par conséquent, un état de service est équivalent à l'information de tous les appels successifs invoqués et de leurs réponses. Dans certains cas, il est alors raisonnable de transférer un état en jouant à nouveau sur le service à lier la même séquence d'appels effectués auparavant sur le service à délier. La solution n'est pas envisageable dans le cas de service où les séquences d'appels à jouer ont des conséquences visibles pour l'utilisateur (exemples : longue durée de rejeu, coût important du rejeu). Dans l'application de suivez-moi auditif, l'application doit non seulement rejouer les paramètres de volume sonore, l'identité du morceau de musique joué mais elle doit aussi requérir la position courante du jeu en cours afin de ne pas laisser l'utilisateur réécouter le début du morceau lorsqu'il passe d'un poste de musique à un autre. Cet exemple montre que les paramètres d'état sont spécifiques aux applications et que le simple rejeu des appels invoqués sur le service précédent n'est pas toujours suffisant pour que l'état soit complètement transféré.

Une autre approche permet de concevoir des applications uniquement à base de services sans état (stateless services). C'est d'ailleurs une recommandation de l'approche appelée REST (Representational State Transfer) [Fie00]. Les styles architecturaux REST de Roy Fielding s'appliquent premièrement à des services pouvant gérer des sessions multiples. Ces services doivent avoir les capacités de mettre en file d'attente des appels et d'y répondre de manière (quasiment) simultanée afin que l'état "occupé" ne soit pas visible. REST implique deuxièmement que l'état d'une session puisse être transmis intégralement dans les arguments de chaque appel de sorte qu'aucun état ne soit stocké sur le serveur, même partiellement. Ces conditions compliquent la gestion des appels d'opérations et sont par conséquent parfois inapplicables. Par exemple, ils sont difficilement applicables dans des applications embarquées aux capacités de calculs restreintes. Ils ne sont pas applicables non plus sur des équipements qui ne peuvent servir qu'un client à la fois, par exemple une machine à laver dans un réseau domestique.

5.4 Exigences techniques

Rendre possible les reliaisons dynamiques dans ces scénarios demandent des réponses logicielles à plusieurs exigences techniques.

La plupart des intergiciels orientés services ne fournissent que des moyens simples de sélection de service. La sélection repose souvent seulement sur la définition de domaines (scopes) et de filtres en raison de la simplicité de ces mécanismes. Les domaines et les filtres restreignent l'ensemble des liaisons disponibles mais ils laissent le demandeur de service considérer les possibilités restantes comme identiques. Les intergiciels existant n'évitent pas l'ambigüité de service. L'ambigüité de service apparaît lorsque deux fournisseurs de service remplisse les mêmes requis. Elle mène à l'imprédictibilité de la composition de services résultante : dans des situations identiques, la composition de l'application sera différente, ce qui peut perturber l'utilisateur. Ce problème soulève le besoin d'introduire des algorithmes de classement parmi les mécanismes de composition de services.

Dans les applications attentives au contexte, les algorithmes vise le classement des fournisseurs de service selon les besoins contextuels de l'application à l'exécution. Des mécanismes de haut niveau sont requis afin de séparer la définition du comportement contextuel et la logique du cœur de l'application. La problématique est décomposable en 4 sujets distincts qui sont discutés dans la section suivante :

- **Qualification contextuelle des propriétés de services** : le contexte du fournisseur de service est acquis par lui-même ou de manière extérieure. Par exemple un centre multimédia pourra lister les codeurs-décodeurs (codecs) qu'il permet alors qu'un système extérieur, tel un système de gestion de localisation, seul pourra renseigner la pièce qu'il occupe. Dans les deux cas, cette information doit être ajoutée dynamiquement à la description des services fournis.
- **Qualification contextuelle des requêtes de services** : les préférences statiques et l'activité dynamique de l'utilisateur peuvent être acquises par des entrées internes au demandeur de services ou de manière extérieure. Par exemple, les paramètres entrés par l'utilisateur au travers d'interfaces graphiques peuvent être considérés comme acquis par des entrées internes si le demandeur de services délivre ces interfaces ou par des entrées externes si les interfaces graphiques appartiennent à une autre entité.
- **Classement dynamique de services** : les besoins des demandeurs de service doivent être projetés sur les différentes techniques de sélection : domaines, filtres obligatoires, filtres optionnels, algorithmes de tri afin de classer dynamiquement les services selon les besoins de l'application donnée. Un changement dans le classement propre à l'application déclenche des actions de reliaison.
- **Sujétion à des politiques de (re-)liaison** : le concepteur peut raffiner l'ensemble des actions de dé-liaison et de reliaison afin de garantir un niveau de continuité de service. Ces taches dépendent souvent spécifiquement de l'application donnée, ce qui rend difficile la tentative d'automatiser les réactions de liaison. Les actions de continuité de service sont le plus souvent l'objet d'une optimisation spécifique.

5.5 Etude de diverses architectures existantes

5.5.1 Contextualisation par un système de gestion de contexte

L'originalité de l'architecture de [PaT06] réside en partie dans les politiques de sélection sémantique (cf. section 5.5.3) mais surtout dans la séparation des applications et des sources de contexte. L'article décrit un annuaire de service centralisant les informations provenant d'un registre de service et de sources de contexte réparties. Une des particularités de l'architecture est que les fournisseurs se services s'enregistrent auprès de l'annuaire en

mentionnant les sources de contexte pertinentes pour qualifier leur propre service. Les demandeurs de service émettent de même leurs requêtes en les associant à des sources de contexte. La requête et l'offre de service sont donc contextualisées par des sources d'information externes aux services et leurs clients. Les requêtes sont évaluées de manière synchrone (découverte active) et asynchrone (découverte passive). Lorsque la requête est traitée de manière asynchrone, le demandeur est notifié des changements de résultat au cours du temps. Les réponses asynchrones permettent aux demandeurs de service de réagir aux changements de contexte. Ces réactions conscientes du contexte ne sont toutefois pas explicitées dans la proposition.

La séparation du système de gestion de contexte et du middleware de services est non seulement requise par le principe de séparation des préoccupations mais aussi pour des raisons d'efficacité. Le découplage temporel entre requêtes de service et raisonnement sur le contexte est aussi promu par [ABR05]. Dans ces systèmes, le raisonnement sur le contexte est délégué à une infrastructure qui traite les données de manière asynchrone aux demandes des demandeurs de services.

5.5.2 Fonctions d'utilité

Caroline Funk propose DSCL [FMK06], un langage pour la description d'une composition de services. DSCL est un acronyme pour Dynamic Service Composition Language. Ce langage décrit la composition d'entités à requérir selon une interface de service, des filtres obligatoires, des filtres optionnels, une fonction objective et des règles conditionnelles. Le filtre obligatoire est un premier filtre composé de paires attributs-valeurs. Les filtres optionnels sont présents pour le raffinement de la sélection des services passant le premier filtre. La fonction objective classe enfin les services selon l'ordre donné par l'évaluation de la fonction sur les qualités des services disponibles ayant passé les filtres précédents. Ces moyens de sélection de service sont complétés de surcroit par des règles conditionnelles afin de réagir au contexte ambiant. Aucun service n'est proposé au demandeur si ces règles conditionnelles ne sont pas vérifiées. Un exemple est la condition que l'utilisateur se trouve à un endroit donné. DSCL peut donc être considéré comme l'amorce d'un langage de sélection de service complet. [FSLM07] montre que DSCL peut être utilisé dans des compositions complexes. Toutefois, le contexte ne semble provenir que des équipements et de ce que l'utilisateur a inscrit sur l'interface d'une application. Un demandeur de service intermédiaire entre l'utilisateur et les équipements terminaux ne semble pas pouvoir influer sur la composition de services durant l'exécution.

Les auteurs de [MiT04] montrent le défi que pose l'affectation de valeurs numériques à des propriétés de services (priorités locales). Ils promeuvent l'emprunt de la logique floue et proposent des modèles de transformation mathématiques pour simplifier le calcul de fonctions d'utilité. La logique floue permet aux demandeurs de services d'interpréter les valeurs proposées de manières variées en intégrant l'aspect vague inhérent de la réflexion humaine ("the vagueness of the human thinking").

5.5.3 Politiques de sélection sémantique

Le projet DReggie [CPAJ01] propose une solution de découverte de service syntaxique étendue par des politiques de sélection sémantiques. DReggie est une extension de l'implémentation de Sun, appelée Reggie, du registre de service Jini (Jini Lookup Service). Cette extension ajoute à la recherche de service Jini des fonctionnalités de description et de sélection issue de la technologie DAML (http://www.daml.org), une technologie du Web Sémantique. Dans cette architecture, les fournisseurs de service attachent des documents descriptifs à leurs services publiés dans le registre de service Jini. Les demandeurs de services peuvent de même ajouter des informations sémantiques à leur requêtes de service afin d'obtenir des services plus adaptés. DReggie utilise un moteur à base de règles – définies dans le langage Prolog – pour déterminer le fournisseur de service le plus adéquat.

L'extension de DReggie définit par conséquent un modèle de politique de sélection de services. Les auteurs décrivent l'exemple d'un client requérant un service d'impression couleur selon la technologie laser dans une situation où les où deux services d'impression sont disponibles. Le premier est un service noir et blanc de la technologie laser alors que le second délivre de la couleur avec la technologie à jet d'encre. Le second est retourné par DReggie car celui-ci infère que l'attribut de couleur est prépondérant sur l'attribut de technologie à partir de la requête sémantiquement enrichie.

L'architecture proposée par les auteurs de [PaT06] décrit de même une solution de découverte de service syntaxique Jini étendue par des politiques de sélection sémantique utilisant la technologie OWL [OWL04] qui dérive de DAML. [ABR05] propose une approche similaire et souligne l'importance du coût du raisonnement des systèmes d'inférence. L'architecture de [LAOS06] présente aussi l'extension d'un annuaire de services standard. L'annuaire UDDI [UDDI02] est lié à une base de connaissance où les demandeurs de services publient leurs requêtes. UDDI est utilisée pour la découverte de services alors qu'une base de connaissance séparée réagit aux événements de services pour le compte des demandeurs. L'algorithme de recherche de services est effectué en deux étapes. La première consiste à effectuer une recherche classique de type de service dans l'annuaire UDDI. La deuxième est un appel selon le protocole WS-Metadata-Exchange [MEX06] sur les fournisseurs de services disponibles afin de connaître le service qui est le plus proche de la politique enregistrée par le service dans la base de connaissance. La base de connaissance est responsable de l'envoi de la liste classée des services disponibles au demandeur de service.

[DaL06] décrit l'implémentation de mécanismes auto-adaptifs au-dessus du modèle à composant Fractal [BCLQ06]. Les auteurs définissent des politiques d'adaptation réactive grâce à un système de règles Evénement-Condition-Action. Ces règles sont affectées aux composants de manière externe. Des événements internes et externes au composant peuvent déclencher les réactions. Le code de la réaction est tissé dans le composant à l'exécution grâce à des techniques de programmation orientées aspects.

5.5.4 Continuité de service

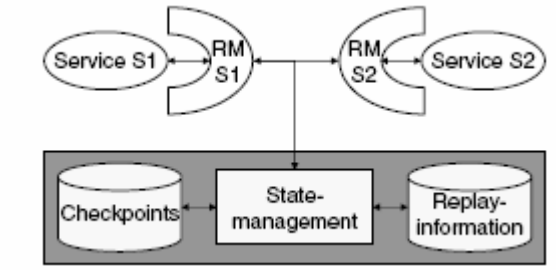


Figure 17 Gestion de transfert automatique d'état dans le système de [FEL07]

Caroline Funk a tenté d'automatiser le transfert d'état entre composants dans un système décrit dans [FEL07]. Ce système utilise le langage DSCL décrit dans la section 5.5.2 pour composer et recomposer dynamiquement les applications. Ce système impose l'association d'un proxy à la représentation de chaque service à état sur le système. Ce proxy expose la même interface que le service et implémente l'interface ReplayManager (RM – cf Figure 17) – gestionnaire de rejeu – dans le langage de programmation sur le système du client. Ce proxy est chargé d'enregistrer la séquence d'appel de méthodes effectuée sur ce service. Si l'instance de service est remplacée par une autre, le gestionnaire de rejeu est utilisé pour rejouer la séquence sur la nouvelle instance de service afin que celui-ci retrouve l'état de l'instance précédente. Les gestionnaires de rejeu sont liés à un système de gestion d'état (State Management – cf Figure 17) qui stocke les séquences de rejeu et les états stables (checkpoints – cf Figure 17) à partir desquels débuter les séquences de rejeu.

6 SYNTHESE

L'Informatique Pervasive est un domaine technico-scientifique rassemblant de nombreuses disciplines. Cette thèse s'inscrit parmi les projets contemporains de Génie Logiciel. Les trois défis importants qu'adresse cette thèse sont la gestion de la répartition, la gestion de l'hétérogénéité et la gestion de la dynamique des réseaux pervasifs dans les applications embarquées.

Face aux défis de l'Informatique Pervasive, le paradigme service apporte le couplage lâche nécessaire à la composition de fonctions offertes dans un environnement ouvert. Les équipements de l'environnement visé sont en effet hétérogènes, se connectent et se déconnectent du réseau suivant des actes d'administration divers.

Depuis la fin des années 90, des intergiciels orientés services sont apparus avec une promesse de composition Plug-n-Play des équipements sur les réseaux locaux et Internet. Tous ces intergiciels présentent des couches protocolaires communes : Adressage, Nommage, Découverte, Description, Invocation, Notification et différentes couches dites non-fonctionnelles. Cette liste de couches protocolaires génériques et les critères associés sont repris dans la section Chapitre III.1.1 afin de comparer les différents protocoles Plug-n-Play existant sur les réseaux locaux, en particulier le réseau domestique.

L'état de l'art de la gestion des aspects distribués dans les réseaux pervasifs montrent un domaine mûr où des réponses complètes sont déjà apportées quant à la transparence des appels de procédures et de méthodes distantes pour le développeur d'applications. Certaines techniques s'appliquent durant la phase de développement, d'autres adressent même la répartition d'applications dans une phase post-développement. Toutefois, la transparence de la découverte d'entités distribuées n'est qu'incomplètement résolue et l'avènement de technologies de déploiement modulaires avec partage de code entre modules telles que les plateformes .NET et OSGi ré-ouvrent le domaine d'investigation.

De nombreux projets se montent face à la demande de composition de services hétérogènes dans les environnements ouverts. Face à l'hétérogénéité des protocoles, des langages de programmation et des interfaces programmatiques, les approches peuvent être catégorisées selon les critères dichotomiques suivants : passerelles ou pivot commun, approche protocolaire ou programmatique, approche syntaxiques ou sémantique.

La gestion de la dynamique du contexte de l'environnement dans la composition de service est le dernier sujet d'investigation de cet état de l'art. Le contexte se définit selon trois catégories dans la littérature : contexte utilisateur, contexte équipement et contexte physique. Tous les éléments décrivant ces contextes influence la composition de services dans les applications pervasives. Les ingénieurs du Génie Logiciel utilisent des techniques de sélection de services qui vont du simple filtre à la fonction d'utilité complexe. Cette dernière est adéquate pour des besoins de raffinement précis de la sélection. La reconfiguration des applications après sélection des services les plus adéquats lève la préoccupation de la continuité de service. Ce besoin ne semble satisfait par des solutions qu'au cas par cas.

Avant de présenter plus précisément la problématique adressée par ces travaux de thèse dans le Chapitre IV, l'état de l'art est complété par un panorama des technologies normalisées interagissant avec les équipements de réseaux locaux, en particulier le réseau domestique. Les technologies touchant au contrôle d'équipements, à la communication personnelle et à l'administration d'équipements sont l'objet de descriptions et de comparaisons dans le Chapitre III suivant.

Chapitre III. LE RESEAU DOMESTIQUE, ENVIRONNEMENT PERVASIF PAR EXCELLENCE

Le réseau domestique est un environnement pervasif par excellence. Son ouverture met en exergue les principaux défis de l'Informatique Pervasive. Distribution, dynamique et hétérogénéité de l'environnement. Le réseau est de plus en plus distribué à mesure que les équipements électroniques deviennent disponibles au grand public et prennent place dans chaque pièce de la maison. La dynamique de l'environnement domestique repose, elle, sur l'activité de l'utilisateur et les mesures de sauvegarde d'énergie des équipements. L'hétérogénéité provient du manque de convergence des différentes technologies existantes. Protocoles réseaux, plateformes d'exécution logicielles et matérielles demeurent hétérogènes.

Mark Weiser [Wei91] avait déjà annoncé ce paradoxe dans les années 90 : les ordinateurs devront être nombreux et interconnectés afin de disparaître de la conscience de l'utilisateur. L'objectif de l'Informatique Pervasive est de réaliser cette vision d'un environnement quotidien empli d'équipements électroniques où l'interface avec l'utilisateur devient répartie dans l'environnement. Plus les capacités de calculs et de communication envahissent l'environnement, plus le contrôle des applications peut se fondre dans celui-ci. L'utilisateur a de moins en moins besoin de s'adresser à des interfaces localisées et complexes à mesure que l'interface est répartie et capte les désirs de l'utilisateur des façons les plus variées.

Aujourd'hui, il semble que le marché se situe dans un état transitoire. Le nombre et la variété des équipements électroniques capables d'être connectés en réseau explose : assistant personnels, téléphones mobiles nouvelles générations, centres multimédia, consoles de jeux, etc. Les équipements deviennent effectivement de moins en moins chers, de plus en plus petits et apparaissent de plus en plus nécessaires aux utilisateurs. Toutefois, la vision de Mark Weiser n'est encore pas exaucée : les capteurs, les ordinateurs, les écrans ne sont toutefois pas présents dans toutes les pièces d'une maison. De surcroit, l'auto-organisation des équipements au service des utilisateurs n'est encore qu'une utopie. La configuration des équipements demeure manuelle et technique. Les interactions entre équipements domestiques restent rares et isolées.

La distribution des équipements de l'environnement domestique est d'abord liée au nombre d'activités menées dans la maison. En effet, les équipements électroniques apportent assistance et confort dans les différentes taches et loisirs domestiques. Plus les personnes qui vivent dans la maison seront nombreuses et actives, plus le nombre d'objets sera grand.

Ensuite, la distribution est liée à la volonté de répartir les activités dans cet espace. Un étudiant vivant dans un studio utilisera un ordinateur qui lui servira à la fois d'outil de travail et de télévision. Au contraire, une famille vivant dans une maison affectera des ordinateurs dans les pièces de bureau et des écrans de télévision dans une ou plusieurs pièces de détente. Enfin, la distribution est liée à l'évolutivité offerte par le partage de fonctions dans des équipements différents. Un écran dédié seulement à la télévision offrira un confort plus grand qu'un écran partagé par d'autres applications. Un autre exemple est donné par les systèmes Home Cinema où le tout-en-un n'est qu'un compromis selon des critères de qualité-prix par rapport à un système où les fonctions d'amplification, les haut-parleurs et appareils de lecture sont assemblés en vue d'optimiser la qualité du son dans une pièce de vie. Les tendances à la profusion d'équipements dans l'environnement domestique sont appuyées par le progrès technique et la baisse des coûts de l'électronique.

Le réseau domestique est rendu dynamique premièrement par la mobilité de certains équipements électroniques – téléphones, assistants personnels, disques amovibles, etc. – et par les mesures de sauvegarde d'énergie des équipements. L'entrée et la sortie des équipements mobiles, l'allumage et l'extinction des équipements électroniques sont accompagnés de la connexion et de la déconnexion de ces équipements au réseau domestique. Deuxièmement, la mobilité et les changements d'activité des utilisateurs font varier la pertinence des fonctions procurées par les équipements de l'environnement. La satisfaction de l'utilisateur est souvent l'objectif des applications sur le réseau domestique (cf. section Chapitre II.5.1.1).

Figure 18 Le réseau domestique vu par le projet IST Amigo

L'hétérogénéité du réseau domestique provient de la variété des activités domestiques et du grand nombre d'acteurs dans ce domaine. En effet, de la variété des activités dérive la diversité des domaines d'applications : multimédia, information, communication interpersonnelle, surveillance domotique, confort domotique, cuisine, nettoyage, télétravail. Et en raison du grand nombre d'acteurs, de nombreux protocoles partagent déjà chaque domaine d'application (cf. Figure 18) alors que les équipements domestiques commencent seulement à devenir connectés. Si les appareils multimédias sont connectés en raison de l'explosion du marché des contenus numériques, d'autres commodités comme les produits blancs – réfrigérateur, lave-linge, etc. – ne le sont pas encore de manière répandue. De surcroit, si le réseau domestique devient une réalité pour la connexion d'équipements à l'Internet, peu d'équipements interagissent dans une collaboration interne au réseau domestique. Et lorsque cette collaboration est réelle, elle n'existe qu'au sein d'un même

domaine d'application et au travers d'un protocole commun. Les sections suivantes font état des domaines d'applications et des technologies concurrentes dans chaque domaine. Ces domaines et ces technologies sont l'objet d'expérimentations décrites dans ce document. Certaines technologies ont même été l'objet de propositions d'amélioration dans des organismes de normalisation : Alliance OSGi [2007-5][2007-8][2006-3], Forum UPnP [2007-9], Comité ISO [2006-7].

1 CONTROLE D'EQUIPEMENTS

Cette section sur le contrôle d'équipements comprend la description et la comparaison d'intergiciels utilisés par des équipements sur les réseaux locaux, principalement le réseau domestique. Les protocoles opérant les équipements d'un réseau local à partir de l'extérieur ne sont pas visés par cette section (cf. les sections suivantes pour les protocoles de communication interpersonnelle et les protocoles d'administration).

Des intergiciels de communication répartie sont utilisés dans le domaine du contrôle d'équipements. Plusieurs ensembles protocolaires sont pourvus de protocoles de découvertes, ce qui en fait des protocoles dits Plug-n-Play. Ces protocoles sont utilisés dans des applications sur des réseaux de proximité, dits PAN (Personal Access Network, cf. section Chapitre III.1.3) ou sur des réseaux locaux, dits LAN (Local Access Network). Les équipements convoyant des ressources numériques importantes utilisent des protocoles IP, décrits dans la section suivante. Les équipements pour lesquels la complexité de la pile IP apparaît superflue, utilisent des bus de terrain qui sont définis dans la 2ème section de cette partie. Ceux-ci définissent des couches réseau particulières sur des couches physiques variées.

1.1 Protocoles Plug-n-Play IP

La pile IP est une couche réseau pertinente et d'un coût acceptable lorsque les équipements convoient d'importantes ressources numériques. Les équipements multimédia, d'impression et les équipements de connectivité IP sont les classes principales d'équipements (si l'on excepte les assistants personnels et ordinateurs) visées. Le marché multimédia est par exemple divisé par l'utilisation de trois standards protocolaires distincts principaux : UPnP [UPnP06] appuyé par l'organisme de recommandations DLNA (Digital Living Network Alliance, www.dlna.org), Apple Bonjour [StC05] à la base de l'application iTunes notamment, et le standard chinois IGRS [IGRS06]. Le domaine de l'impression – imprimantes, scanners – est partagé par les mêmes standards en supplément du nouveau standard poussé par Microsoft : DPWS (Devices Profile for Web Services [DPWS06]). Dans le domaine de la connectivité IP – routeurs, points d'accès Wifi – UPnP a pris le monopole. Et avant que ces protocoles ne gagnent des parts de marché, de nombreuses autres technologies ont été pionnières : Jini [Wal99], SLP (Service Location Protocol [GPVD99]), Salutation. SLP était utilisé par les ordinateurs et équipements Apple jusqu'à la version MacOS 10.2 à partir de laquelle Apple a imposé Bonjour.

Ces protocoles Plug-n-Play IP partagent de nombreux points communs :

- L'utilisation de la pile protocolaire IP et en particulier les mécanismes d'adressage DHCP et AutoIP,
- l'utilisation d'un protocole de découverte qui les fait appartenir à la classe des intergiciels orientés service,
- l'utilisation d'un numéro unique d'identification,
- la normalisation de propriétés d'équipements – numéro de série, nom de modèle, etc.,

- et la définition de description d'équipements spécifiques.

S'ils partagent des points communs, ils montrent aussi des différences saillantes qui les rendent difficilement interopérables. Un résumé est donné dans le Tableau 1 :

Tableau 1 Comparaison des différentes spécifications protocolaires Plug-n-Play

Technologie / Couche technique	UPnP	IGRS	DPWS	Bonjour	SLP	Jini	Salutation
Nommage	Optionnel, DNS	Optionnel, DNS	Optionnel, DNS	Obligatoire, DNS ou mDNS	Non spécifié	Non spécifié	Non spécifié
Répertoire de service	Moyens multicast	Moyens multicast	2 modes : Discovery Proxy et moyens multicast	2 modes : DNS Server et moyens multicast	2 modes : Discovery Agent et moyens multicast	Lookup Service	Salutation Manager
Mode de découverte	Actif et passif	Actif et passif	Actif et passif	Actif et passif	Actif	Actif et passif	Actif et passif
Description de service	Nommage, Opérations, événements	Nommage, Opérations, événements	Nommage, Opérations, événements	Nommage	Nommage	Nommage, Opérations, événements	Nommage
Méta-données	Attributs-valeurs	Attributs-valeurs	Affectation et sélection riches.	Attributs-valeurs et filtres de service	Attributs-valeurs et filtres de service	Attributs-valeurs et filtres de service	Attributs-valeurs et filtres de service
Couches techniques supérieures	Sécurité, QoS, administration, etc.	Sécurité et spécifications à venir	Sécurité, QoS, administration, etc.	Agnostique	Agnostique	Spécifications Java : sécurité, etc.	Agnostique
Langage de programmation	Agnostique	Agnostique	Agnostique	Agnostique	Agnostique	Java	Agnostique

- L'importance de la couche de nommage : seule la technologie Bonjour la rend indispensable et spécifie un protocole multicast de nommage, mDNS, dans le cas où le réseau est "non-géré", c'est-à-dire qu'aucun serveur DNS n'est présent.
- La notion de répertoire de service : certaines technologies comme UPnP et IGRS spécifient seulement des moyens multicast de découverte, certaines comme Jini et Salutation seulement des moyens unicast avec utilisation d'un répertoire de services, enfin d'autres comme SLP, Bonjour et DPWS définissent les deux modes et le passage réversible d'un mode à l'autre.
- Le mode actif ou passif de la découverte : tous les protocoles mentionnés spécifient les deux modes sauf SLP qui ne spécifie que le mode actif.
- Les descriptions de service : alors que la plupart des ensembles protocolaires définissent un schéma d'interface de service avec les notions d'opération et d'événement, SLP, Bonjour et Salutation ne définissent que la syntaxe des noms d'interface.
- L'affectation de métadonnées : si tous les protocoles partagent la possibilité de qualifier les entités découvertes avec une liste de paires attributs-valeurs, la flexibilité et la richesse de l'affectation de métadonnées est variable. UPnP et IGRS ne permettent pas la définition de filtre sur ces métadonnées lors de la recherche de service. Par ailleurs, seul DPWS est associé à des protocoles

standards de définitions de métadonnées riches – WS-MetadataExchange [MEX06] et WS-Policy [WSP07].

Tableau 2 Couches techniques des standards répandus dans le réseau domestique

Couches essentielles pour la découverte					
Technologie / Couche technique		UPnP	IGRS	DPWS	Bonjour
Adressage	Serveur DHCP	DHCP	DHCP	DHCP	DHCP
	Non-géré	AutoIP	AutoIP	AutoIP	AutoIP
Nommage	Serveur DNS	DNS		DNS	DNS
	Non-géré				Multicast DNS
Découverte		SSDP	SSDP	WS-Discovery	DNS-SD
Couches essentielles pour le contrôle					
Description		UPnP (XML) Template Language	WSDL	WSDL WS-Addressing WS-Policy WS-Mex	DNS SRV (RFC2782)
Contrôle		SOAP	SOAP	SOAP + MTOM	
Eventing		GENA	GENA	WS-Eventing	
Exigences non-fonctionnelles					
Securité		UPnP Security Ceremonies	IGRS Security	WS-Security	
QoS		UPnP QoS		WS-Policy	
Administration		En cours de définition dans le Comité DM		WS-Management	
Accès distant		UPnP Remote Access		Discovery Proxy[1] (?)	
Interfaces déportees		UPnP Remote IO			
Etats électriques		UPnP Low Power			
Equipements et applications normalisés					
Equipements et applications normalisés		Internet Gateway, WLAN Access Point, Media Server, Media Renderer, Printer, Scanner, Lighting controls, Security Camera, HVAC System	Spécifications correspondantes à celles d'UPnP	Printer Scanner	iTunes, File Sharing, iChat AV, iPhoto, Safari Browser, Airport Station

[1] L'accès distant aux équipements d'un réseau local semble techniquement faisable à l'aide du Discovery Proxy et de WS-Addressing. Toutefois, le standard DPWS n'est pas explicite.

- La définition de couches techniques supérieures à la couche de description : le Tableau 2 compare les couches protocolaires de 4 standards dominants sur les réseaux locaux. SLP et Bonjour considèrent la spécification des couches fonctionnelles de contrôle et non-fonctionnelles comme étant du ressort des domaines d'application spécifiques. Les organismes de standardisation d'UPnP, de DPWS et d'IGRS s'engagent dans la définition de protocoles additionnels de portée générale. Jini bénéficie des atouts de Java et de RMI (Sécurité, transactions, etc.).
- L'adhésion à un langage de programmation : la spécification Jini adhère au langage Java.

D'autres classifications sont apportées dans la littérature technico-scientifique. [MHS05] étend une classification similaire à l'étude de nombreuses solutions intergicielles académiques sur des réseaux IP locaux, sur le réseau Internet et sur les réseaux ad hoc. Feng Zhu et al. [ZMN05][ZZMN05] étendent la classification à des critères de sécurité et d'intimité. Le 2ème chapitre de la thèse [Tho06] apporte des critères supplémentaires pour la description et la publication des services dans les environnements pervasifs sur une liste relativement large d'intergiciels protocolaires. [Kei06] compare quelques autres protocoles de découverte intéressants.

1.1.1 UPnP/DLNA

Force est de reconnaître que la technologie UPnP (Universal Plug aNd Play [UPnP06], www.upnp.org) est l'intergiciel protocolaire standard le plus répandu sur le réseau domestique aujourd'hui. Son essor sur les équipements multimédia a été appuyé par l'organisme de recommandations DLNA (Digital Living Network Alliance, www.dlna.org). DLNA raffine, entérine les profils d'équipements spécifiés par UPnP et parfois fournit de nouvelles exigences techniques pour des versions ultérieures. UPnP est déployé de manière importante sur deux marchés principaux :

- Le marché multimédia : progressivement, tous les téléviseurs et les équipements qui peuvent s'y raccorder deviennent UPnP : les consoles de jeux (Microsoft XBox, Sony PlayStation Portable, etc.), les systèmes Home Cinéma, les Set-Top-Box, les chaines Hi-Fi notamment. Au travers de leur connexion. D'un point de vue utilisateur, ils sont capables de découvrir les équipements de stockage de contenus multimédia connectés au réseau domestique, de proposer la navigation dans ces contenus à l'utilisateur et de jouer les contenus choisis sur la télévision ou la chaine Hi-Fi raccordée. D'un point de vue technique, ces équipements sont généralement des points de contrôle UPnP AV, parfois certifiés DLNA Media Player. Les équipements de stockage sont, eux, des Media Servers UPnP, parfois certifiés DLNA Media Server. La certification DLNA requiert la spécification UPnP et le nombre d'équipements de ces catégories ne cesse d'augmenter sur les listes d'équipements certifiés DLNA (cf. http://certification.dlna.org/products/) et des équipements certifiés UPnP (cf. http://www.upnp-ic.org/default.asp).
- Le marché de la connectivité IP : la majorité des passerelles Internet domestique implémente le profil UPnP Internet Gateway Device. Cela permet à toute application du réseau domestique de découvrir la passerelle et de configurer les ouvertures de ports sur l'extérieur pour ses besoins. Les applications de messagerie instantanée et de partage de contenus Peer-to-Peer proposent généralement à l'utilisateur de configurer le réseau automatiquement via UPnP. La passerelle domestique est donc configurée automatiquement sans que l'utilisateur n'ait à savoir ce qu'est une IP, à trouver celle de la passerelle, et à

comprendre ce qu'est le "port-mapping". Cela permet aussi d'éviter des erreurs de configuration qui peuvent nuire à la sécurité du réseau. En revanche, d'aucuns reprochent le manque de sécurité du profil qui pourrait être accédé par des applications malveillantes [Hem06]. UPnP a pourtant spécifié un protocole de sécurité appelé "UPnP Security Ceremonies". Le débat demeure technique et pragmatique : les moyens de sécurité (authentification) nuisent à l'aspect Plug-n-Play et le danger réellement dû à la spécification reste à évaluer.

UPnP est présent aussi sur d'autres marchés avec toutefois moins de succès :

- Le marché des cameras IP : certaines caméras autonomes sont raccordables au réseau domestique grâce à un câble Ethernet ou à une liaison Wifi. La plupart n'implémente que la partie découverte du protocole UPnP afin d'être détectées aisément mais n'implémente aucune description standard. Parfois, les caméras du commerce implémentent la spécification UPnP Media Server et non le profil Digital Security Camera Device. La raison est probablement la publication plus tardive du dernier profil et l'existence de points de contrôle pour le premier profil sur beaucoup d'équipements commerciaux aujourd'hui. L'utilisateur peut donc découvrir la caméra via sa télévision DLNA Media Player et visualiser le bébé qui dort dans la chambre et qui est surveillé par la caméra DLNA Media Server.
- Le marché de l'impression : certaines imprimantes commerciales annoncent leur présence via UPnP.

Tableau 3 Pile protocolaire UPnP

<table>
<tr><td colspan="4">UPnP Vendor</td></tr>
<tr><td colspan="4">UPnP Forum</td></tr>
<tr><td colspan="4">UPnP Device Architecture</td></tr>
<tr><td colspan="2">SSDP</td><td>GENA</td><td>SOAP</td></tr>
<tr><td>HTTPU</td><td>HTTPMU</td><td colspan="2">HTTP</td></tr>
<tr><td colspan="2">UDP</td><td colspan="2">TCP</td></tr>
<tr><td colspan="4">IP</td></tr>
</table>

UPnP et une spécification technique double. UPnP est d'abord un intergiciel de communication répartie orienté service générique. UPnP est un intergiciel pré-Web Services. Il est né dans la mouvance des protocoles dits humainement lisibles (human-readable). UPnP est un patchwork de technologies Web - IP, TCP, UDP, HTTP, XML, SOAP – et de standards issus de l'IETF – SSDP et GENA pour la découverte et les événements (cf. Tableau 2 et Tableau 3). Les protocoles HTTP sur UDP et sur IP Multicast respectivement appelés HTTPU et HTTPMU sont des particularités ad hoc définies par UPnP. Les modèles de description XML des équipements (devices) et des services sont une autre particularité : l'UPnP Template Language.

L'UPnP Device Architecture [UPnP06] définit l'usage de ces protocoles dans le contrôle d'équipements. La spécification spécifie les échanges entre les clients, appelés Control Points, et les équipements, appelés Devices, qui fournissent des services (cf. Figure 19). Un Device est une entité logicielle. En ce sens, il peut héberger d'autres Devices et chaque Device publie un ou plusieurs services, un service étant décrit par une liste de variables d'états et d'actions. Chaque action est une opération request-response (cf. Chapitre

II.2.5) au-dessus du protocole SOAP. Chaque variable d'états peut être déclarée comme source d'événements. Un point de contrôle peut souscrire aux événements d'un service. Dans ce cas, il sera notifié de tous les changements d'états des variables déclarées source d'événements.

UPnP est ensuite une collection de spécifications de profils d'équipements spécifiques ou de couches techniques non-fonctionnelles. Ces spécifications sont appelées DCPs (Device Control Protocols). Cette collection augmente au fur et à mesure des publications des différents Comités de Travail (Working Committees) à l'œuvre dans le Forum UPnP. Elle correspond à la couche appelée UPnP Forum dans le Tableau 3. Chaque DCP définit un type de device particulier et un nombre de services obligatoires et optionnels fournis par celui-ci. Chaque type de service spécifié dans un DCP normalise une liste de variables d'états, une liste d'actions à implémenter de façon obligatoire ou optionnelle. Pour implémenter un profil d'équipements UPnP standard, chaque éditeur d'application UPnP doit se conformer à l'UPnP Device Architecture et implémenter tous les services et leurs actions obligatoires. L'éditeur est incité à implémenter les actions optionnelles. Il est libre d'étendre la liste des actions et de services pour ses besoins. Ces définitions étendues sont illustrées par la couche appelée "Device Vendor" dans le Tableau 3.

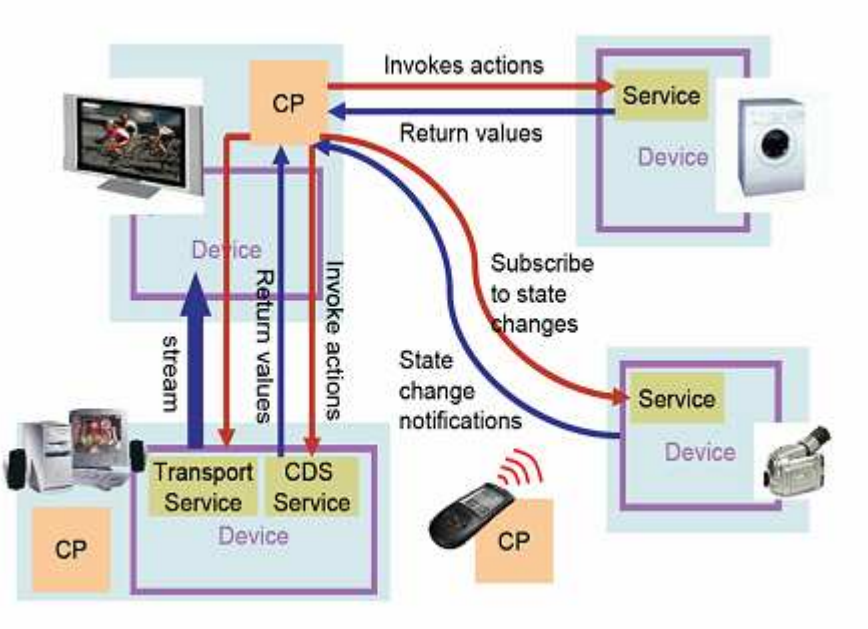


Figure 19 Points de contrôle, équipements et services UPnP

Le succès d'UPnP ne cache pas une multitude de faiblesses techniques notables du standard. La spécification a utilisé des spécifications très diverses et a édifié des modèles ad hoc comme l'UPnP Template Language. Il est donc premièrement à regretter que les messages protocolaires ne soient pas alignés entre eux comme DPWS bénéficie de l'alignement de ses protocoles sur SOAP. Deuxièmement, certaines des fondations d'UPnP n'étaient que des "drafts" en 1999 : SSDP et GENA. SSDP n'est certainement pas un modèle de protocole de découverte même après les raffinements apportés par le Forum UPnP. Il est notamment regrettable que le passage à l'échelle et la fiabilisation de la découverte ne soient pas apportées par la définition d'un répertoire de services. Ce répertoire de service aurait pu être optionnel comme dans les spécifications SLP et WS-Discovery. De surcroit, le protocole de découverte est verbeux, ce que n'a pas hésité à déclaré Jeffrey Schlimmer, auteur des spécifications UPnP et DPWS. Enfin, le protocole ne permet pas de sélection de service sur des métadonnées. Troisièmement, les protocoles UPnP souffrent de leur simplicité : la souscription au niveau service et non au niveau de chaque événement est un défaut. La notion de variable d'état et l'obligation d'attacher chaque argument d'action à une variable d'état nuit

à la simplicité de l'UPnP Template Language. Des références à des types complexes manquent aussi à la clarté des descriptions qui cachent parfois des espaces de nom vastes. Par exemple, le format XML DIDL-Lite utilisé dans l'architecture UPnP AV n'est pas visible dans la description XML des services. D'autres erreurs techniques existent, certaines d'entre elles sont l'objet de corrections dans les Comités de Travail du Forum.

Le succès d'UPnP tient probablement de la simplicité de ces spécifications au dessus de technologies web qui se sont répandues, de la justesse et du nombre de profils d'équipement spécifiés depuis sa fondation en 1999. Les modèles de descriptions de services UPnP illustre la simplicité d'UPnP. Loin de la généricité de WSDL (Web Services Description Language), UPnP définit un langage XML peu hiérarchisé pour définir des listes d'opérations et des listes de variables d'états au-dessus d'une liste de types limitée à des types simples. Le Forum UPnP a spécifié de nombreux profils depuis 1999 dont les plus prometteurs sont ceux de l'Architecture Audio-Video UPnP [UAV02]. La distinction de 3 entités – Point de Contrôle, Media Server et Media Renderer – et l'aspect agnostique de la spécification aux protocoles de flux multimédia (streaming) déclarés "protocoles en dehors du champ" (out-of-band protocols) ont donné un crédit technique aux Comités de Travail du Forum. Le travail a pu être entériné par la suite par DLNA notamment. Evidemment, le succès d'UPnP peut aussi être expliqué par des événements moins techniques : les fabricants d'équipements influents étaient certainement à la table du Comité Directeur (Steering Committee) depuis le début : Intel, Philips, Siemens, Sony, Samsung, LG, Panasonic, Pioneer.

1.1.2 IGRS

Les équipements de la norme IGRS (Internet Grouping and Resource Sharing [IGRS06]) débutent leur essor sur le marché chinois. Cette norme, encouragée par le gouvernement chinois, semble se répandre en Chine et pourrait avoir des retombées plus larges en Asie à l'avenir. La fondation du consortium IGRS en juillet 2003 est relativement récente et le consortium semble vouloir rattraper le succès d'UPnP sur les mêmes domaines d'applications.

L'effort de normalisation des IGRS Labs – www.igrs.org – revisite tous les standards UPnP et publie des spécifications similaires. La spécification IGRS a notamment gardé de la spécification UPnP tous les protocoles des couches adressage, nommage, invocation, notification tout en modifiant la syntaxe des messages protocolaires issus des spécifications des couches invocation et notification. Malheureusement, tous les messages protocolaires spécifiés sont très différents de ceux d'UPnP et les deux types d'équipements ne sont pas interopérables malgré la volonté initiale visible dans les premières versions de la spécification. Pire, avant sa proposition en comité ISO/IEC en 2006, la spécification IGRS ne permettait pas aux équipements UPnP et IGRS de coexister sur un même réseau local sans interférences. En effet, le standard avait gardé une même adresse multicast partagée par tous les équipements IGRS et UPnP pour les messages de découverte. Les équipements IGRS et UPnP auraient interféré par échange de messages incompatibles. La coexistence des équipements a été rendue possible par l'acceptation de la modification de l'adresse multicast pour la découverte des équipements IGRS [2006-7].

La partie techniquement originale de la spécification se présente selon trois traits principaux :

- Le remplacement de l'UPnP Template Language par l'emploi du standard WSDL (Web Services Description Language). Cela permet à IGRS de reprendre un standard très répandu, d'accéder à la généralité des descriptions des Web Services, en particulier à la description des types complexes.
- L'ajout d'une couche technique appelée Device Grouping. L'intérêt de cette couche est de pouvoir isoler des groupes d'équipements pour des raisons de

sécurité. La découverte dans un groupe centralisé passe par un équipement appelé Maitre, les autres membres étant nommés esclaves.

- L'ajout de paramètres de sécurité parmi les paramètres de description des équipements et des services. La sécurité est intimement mêlée à la spécification générale IGRS alors que les Cérémonies de sécurité UPnP (UPnP Security Ceremonies) étaient gardées séparées de la spécification UPnP tout comme WS-Security est séparée de la spécification DPWS. Ce mélange et le fait additionnel que les protocoles de sécurité imposés n'existent qu'en Chine rend cette différence techniquement regrettable.

Enfin, les descriptions d'équipements spécifiques IGRS profitent aussi des spécifications UPnP. Les IGRS Labs revisite et publie progressivement des spécifications d'équipements similaires aux profils UPnP (ou UPnP DCPs, cf. section 1.1.1 précédente et Tableau 2). Un exemple est donné par la Figure 20 qui montre l'interaction entre un point de contrôle IGRS (Controller), des serveurs multimédia et un lecteur multimédia hébergeant des services similaires en nombre et en nommage à ceux d'UPnP. Les spécifications étant très récentes et les produits n'étant pas commercialisés en Europe, il n'est pas aisé d'évaluer le succès commercial éventuel à venir de celles-ci.

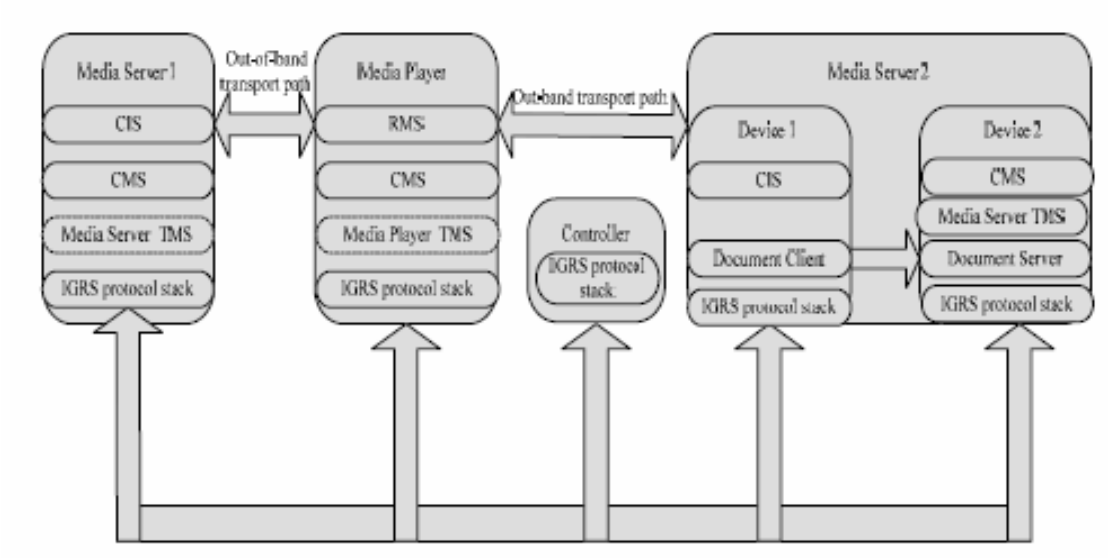


Figure 20 IGRS AV [IGRS06], une architecture similaire à celle d'UPnP AV

1.1.3 DPWS

DPWS (Devices Profile for Web Services [DPWS06]) est une spécification récente qui bénéficie techniquement de son alignement sur les Web Services et commercialement de la force commerciale de son principal promoteur : Microsoft. DPWS se trouve déjà implanté dans quelques rares produits commerciaux, notamment le serveur d'exploitation Microsoft Vista et quelques imprimantes. Des fabricants de terminaux électriques seraient en partenariat avec Microsoft sur l'emploi de la technologie dans le domaine du contrôle domotique. Schneider Electric est un acteur qui a poussé la technologie dans plusieurs projets européens, les Projets ITEA Sirena [JaS05], Anso auquel le laboratoire MAPS/AMS de France Telecom R&D a participé [2008-2], Soda (www.soda-itea.org), et le projet national du Pole de compétitivité Minalogic (www.minalogic.com) nommé Smart Electricity.

L'originalité de DPWS tient sur l'alignement de la technologie sur les protocoles des Web Services. Tous les protocoles d'échanges entre clients et serveurs DPWS sont basés sur les spécifications SOAP, WSDL et XML Schema Figure 21. DPWS bénéficie de la maturité des spécifications comme WS-Discovery, comme WS-Eventing et de la généralité du

standard de description WSDL attaché protocole d'échange de métadonnées WS-MetadataExchange et de politiques WS-Policy. Voici la description détaillée des couches techniques originales de DPWS :

- Découverte : WS-Discovery est un protocole de découverte qui permet les modes actif et passif avec ou sans répertoire de services. Tout équipement annonce sa présence par un message multicast appelé Hello à la connexion et Bye à la déconnexion. Ces messages sont envoyés par IP Multicast même en présence d'un répertoire de services. Le répertoire de services est appelé Discovery Proxy. Ce dernier s'annonce sur le réseau comme service et répond à toute requête multicast par un message Hello unicast. Dès que les clients DPWS reçoivent un message Hello d'un Discovery Proxy, toutes les requêtes actives de découverte suivantes sont adressées par messages unicast au Discovery Proxy. Si le Discovery Proxy vient à quitter le réseau, les clients DPWS envoient les requêtes actives suivantes par messages multicast.
- Description : la généralité de WSDL permet la définition de types complexes. WS-MetadataExchange permet une description de métadonnées riches tandis que WS-Policy permet la définition d'alternatives de services par expression de capacités et de requis. Les alternatives peuvent être exprimées selon des critères de sécurité, de QoS, etc.
- Notification : WS-Eventing permet la souscription à un ou plusieurs événements présentés par un service.

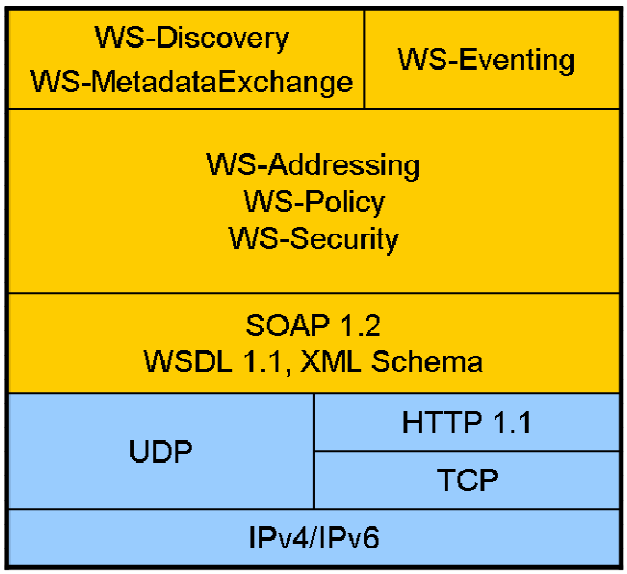


Figure 21 Pile protocolaire DPWS

Microsoft et les acteurs de DPWS ont publié des spécifications additionnelles (cf. Tableau 2). Des spécifications existent pour les équipements du type imprimante et scanner. DPWS bénéficie aussi de l'alignement sur les Web Services pour les protocoles non-fonctionnels suivants :

- WS-Security : sur les mêmes bases que les Cérémonies de Sécurité UPnP, WS-Security permet l'authentification mutuelle d'un serveur et d'un client et le cryptage des données échangées. L'authentification s'effectue par échange de clefs PKI qui doivent être reconnues par un tiers de confiance.
- WS-Management : l'administration d'équipements selon des protocoles Web Services est spécifiée par ce protocole de l'organisme DMTF (Distributed Management Task Force, www.dmtf.org). Il permet l'expression d'un modèle de

données hiérarchique de configuration et sa manipulation par des serveur d'administration.

La maturité et la généralité de la spécification DPWS sont toutefois accompagnées de contreparties pour le domaine des réseaux locaux. La généralité des protocoles va de pair avec une complexité de la lecture des spécifications abstraites et sujettes à interprétations multiples [ZBBG07]. L'écriture et la manipulation des documents WSDL et des messages SOAP d'invocation et de notification sont aussi plus complexes et plus verbeux, ce qui induit des lourdeurs peu adéquates dans le domaine de l'embarqué [2008-5].

Le succès de la spécification est déjà visible dans la littérature scientifique et dans les communautés open source où plusieurs piles protocolaires sont déjà disponibles (www.soa4d.org, www.ws4d.org, gforge.inria.fr/frs/shownotes.php?release_id=1804). Il est dû à son alignement sur les Web Services et au caractère abouti des spécifications. En revanche, le succès commercial n'est pas encore important même s'il est attendu sur le marché de la domotique.

1.1.4 Apple Bonjour/iTunes

Apple Bonjour (ex-Rendezvous [StC05]) est la base protocolaire de l'application iTunes qui est déployé sur de nombreux PC, de nombreux équipements de stockage sur le réseau domestique et quelques équipements de lecture multimédia comme les appareils de radio de marque Roku. Il partage donc le marché du multimédia avec les protocoles UPnP. Apple Bonjour est aussi la base protocolaire de nombreuses applications Apple (cf. Tableau 2). Il est par ailleurs utilisé par quelques imprimantes.

L'originalité technique d'Apple Bonjour est son alignement sur le protocole DNS. Apple profite ainsi de la connaissance répandue, des piles protocolaires disponibles du protocole DNS et des serveurs DNS déjà déployés. Le protocole est étendu afin de définir un protocole de nommage multicast appelé mDNS et afin de définir un protocole de découverte appelé DNS-SD (ou DNS Service Discovery). La découverte est à la fois active et passive. Si un serveur DNS est présent, les clients et serveurs Bonjour s'adressent à ce serveur par messages unicast. Afin de garder une découverte passive en présence d'un serveur DNS, des requêtes persistantes (DNS Long-Lived Queries) sont spécifiées. Si aucun serveur DNS n'est présent, les requêtes et les annonces sont multicast.

DNS-SD permet seulement un filtre de services dans les requêtes selon des noms de types et de sous-types. Il permet toutefois l'affectation d'une liste libre de paires attributs-valeurs à un service fourni que les clients obtiennent dans les réponses à des requêtes de découverte. L'invocation, la notification et les autres couches techniques supérieures à la description sont laissées libres aux éditeurs logiciels. Par conséquent, la description de service demeure restreinte à une chaine de noms de types et de sous-types accompagnée d'une liste de paires attributs-valeurs.

Les applications utilisant la pile protocolaire Bonjour définissent des protocoles d'invocation et de notification complémentaire aux couches d'adressage, de nommage et de découverte de Bonjour. Par exemple, iTunes utilise un protocole propriétaire et confidentiel à Apple, appelé DMAP. Le succès de Bonjour tient plus dans l'attrait des produits Apple que dans les avantages techniques du standard protocolaire lui-même.

1.1.5 SLP

Le principal promoteur de SLP (Service Location Protocol [GPVD99]) a été Apple. Jusqu'à la version 10.2, MacOS utilisait SLP pour échanger des services, notamment de stockage, sur le réseau. Aujourd'hui, Apple emploie Bonjour plutôt que SLP dans ses applications. SLP est toujours utilisé par certaines imprimantes. La simplicité de SLP et son

succès précoce explique probablement que le protocole demeure malgré son abandon industriel une technologie reprise par certains projets académiques [RCCI05][RAR07].

SLP possède des traits similaires à ceux de Bonjour : reprise d'un standard ouvert pour protocole de découverte, découverte multicast avec passage à l'échelle possible par définition d'un répertoire de service optionnel nommé Directory Agent, description des services par un nommage de types et de sous-types de services, affectation d'une liste d'attributs-valeurs aux fournisseurs de services. Les différences principales sont la définition du seul mode passif, le courtage des attributs de services dans les requêtes de découverte.

1.1.6 Jini

Jini [Wal99] a connu un grand succès technico-scientifique au tournant des années 2000. De nombreux chercheurs ont vu dans Jini la technologie phare de l'Informatique Pervasive [GTA02]. Une grande communauté de développeurs s'est créée mais l'engouement s'est éteint progressivement et les équipements embarquant la technologie sont demeurés marginaux. De surcroit, aucun profil standard d'équipement ne semble avoir été spécifié.

L'originalité technique de Jini tient dans sa relation avec Java RMI. Jini profite de l'efficacité de la sérialisation binaire de RMI [DaP02] par rapport au formatage SOAP et du chargement dynamique de classes Java pour le téléchargement à chaud de proxys Java (drivers).

Le protocole de découverte ajouté par Jini définit un répertoire de services appelé Lookup Service (LUS). Ce répertoire peut être connu des clients et serveurs Jini par configuration ou par requête multicast. Les requêtes et les publications de service s'effectuent alors par message unicast auprès du LUS. Les services sont publiés avec une liste libre de paires attributs-valeurs qui peuvent être l'objet d'un filtre de service.

Un service est décrit par une interface Java, soit un ensemble de méthodes. L'interface est définie dans un espace de nom appelé package. Les arguments d'entrée et de retour des méthodes Java sont des types simples ou des objets complexes définis par des classes dans le langage orienté objet Java. Des événements peuvent être définis grâce au patron de conception Observateur usuel en programmation orientée objet. Jini procure des interfaces (net.jini.core.event.RemoteEventListener) qui permettent de surveiller les erreur provenant de la connexion distante de l'observateur et de l'observé. Enfin, Jini bénéficie de la sécurité et des transactions de Java. Toutefois, l'authentification d'une entité distante n'est pas une fonctionnalité procurée par Java.

1.1.7 Salutation

Salutation est un intergiciel de communication répartie qui a eu une influence visible dans la littérature technique [Pas01][Ric00][Hel02]. Le consortium est né en 1995 et a été dissolu en 2005.

L'originalité de cet intergiciel est d'être agnostique non seulement à tout langage de programmation mais aussi à tout protocole de communication. Salutation est principalement la définition de l'interface abstraite du courtier de service appelé Salutation Manager et des projections de cette interface sur les protocoles de découverte et de communication existants.

Le Salutation Manager est un répertoire de service qui se synchronisent avec les répertoires disponibles selon les différents protocoles de découvertes permis par les Transport Managers installés (cf. Figure 22). Chaque équipement est muni d'un Salutation Manager. Les clients et Services de cet équipement interrogent la liste de services et souscrivent aux événements disponibles sur le Salutation Manager.

La notion de pilotes protocolaires est importante dans Salutation. Chaque pilote fournit un Transport Manager (cf. Figure 22). Ce dernier est un traducteur protocolaire pour

le Salutation Manager. Les pilotes sont téléchargeables sur un répertoire appelé "Doc Storage".

Les services de Salutation sont identifiés par les "unités fonctionnelles" (Functional units) qu'ils fournissent. Les types d'unités fonctionnelles sont nommés en utilisant le standard ISO 8824 ASN.1.

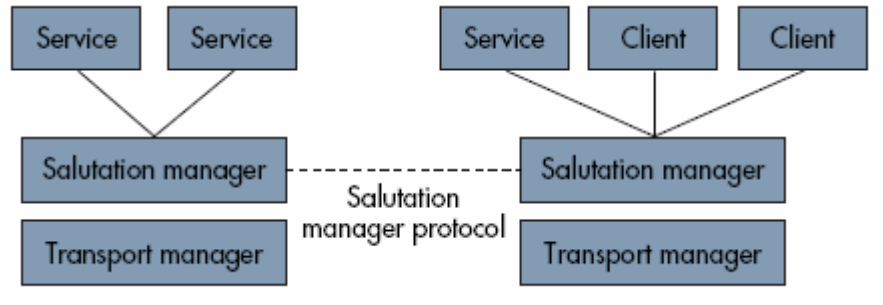


Figure 22 Salutations managers, transport managers, services and clients [Ric00]

L'originalité de Salutation a été certainement une des raisons de son abandon. Etant agnostique à la fois aux langages de programmation et aux protocoles de découverte et de communication existant, l'intergiciel est fondé sur un haut niveau d'abstraction. Cette abstraction pourrait avoir sa place dans les interfaces programmatiques des solutions d'aujourd'hui qui adressent l'hétérogénéité des protocoles.

1.2 Bus de terrain

Lorsque les applications requièrent peu de volume d'échange et que l'exigence première est le prix et le niveau de consommation électrique, la pile IP devient superflue et les bus de terrain deviennent adaptés. Ils sont principalement utilisés dans le domaine de la domotique : surveillance du domicile, systèmes d'alerte pour incendies ou effractions, gestion des lumières, gestion des volets, les systèmes de chauffage, de ventilation et d'air conditionné appelés systèmes HVAC (Heat Ventilation and Air Condition), etc. Les particularités de chaque bus de terrain créent des opportunités au sein des environnements pervasifs [DSTG07].

Comme les protocoles Plug-n-Play IP, les bus de terrains montrent des similarités et aussi de grandes différences entre eux. Dans le cas des bus de terrain, même la couche physique OSI diffère d'un bus à l'autre. ZigBee (www.zigbee.org) est un ensemble protocolaire distribué sur liaison radiofréquence. Il est optimisé pour des taux de consommation bas et offre un protocole de découverte. L'ensemble protocolaire KNX (www.knx.org) – la convergence des trois technologies EHS (European Home Systems Protocol), EIB (European Installation Bus), Batibus – est connu pour son adaptation dans le bâtiment. KNX est majoritairement déployé sur paire torsadée, mais il peut aussi l'être sur courants porteurs, sur liaison radio, sur liaison infrarouge, et même au-dessus d'Ethernet. CEBus (Consumers Electronics Bus, protocole défini par l'Electronic Industries Alliance, www.eia.org) est un concurrent de KNX majoritairement sur les courants porteurs. X10 (www.x10.com) est un protocole de contrôle sans découverte dont l'installation vise les amateurs de bricolage depuis les années 70. Le protocole définit une seule interface de 16 opérations – lightOn, LightOff, etc. – qui peut être adressés sur des équipements localisés sur 256 adresses possibles. Le marché "Do It Yourself" américain présente une offre importante d'équipements X10. Echonet (Energy Conservation and HOmeCare NETwork www.echonet.gr.jp/english) est une spécification asiatique normalisant des interfaces

protocolaires et les interfaces programmatiques associées. De nombreux autres ensembles protocolaires existent comme LON, C-Bus, Z-wave, Insteon, EnOcean, etc.

1.3 Protocoles de la sphère personnelle

On appelle protocoles de la sphère personnelle les systèmes distribués sur de courte portée permettant des hauts débits de communication. USB (Universal Serial Bus, www.usb.org) et FireWire (ou IEEE 1394, www.1394ta.org) sont des normes permettant des taux de transferts de quelques centaines de $Mbit.s^{-1}$ sur des cables dédiées. La détection des équipements USB et Firewire est automatique, ce qui fait d'eux des technologies Plug-n-Play. Bluetooth est un ensemble protocolaire au-dessus d'une liaison radio. Elle permet aussi une découverte automatique avec les fonctionnalités avancées de recherche de types de services et de filtre de services par paires attributs-valeurs. Ces atouts le rendent comparable à des intergiciels Plug-n-Play de la sphère IP [Hel02]. Le Bluetooth SIG (Special Interest Group, www.bluetooth.com) a normalisé de nombreux profils d'équipements : Headset (HSP), Hands-Free (HFP) et Audio/Video Remote Control (AVRCP) qui sont les profils des oreillettes sans fils répandues, Basic Imaging Profile (BIP) qui permet le tranfert d'images, File Transfer Profile (FTP) qui permet le transfert de fichiers, etc. Le succès de Bluetooth se voit notamment avec celui des oreillettes pour téléphones portables.

2 COMMUNICATION INTERPERSONNELLE

La communication interpersonnelle regroupe ici les applications de communication électroniques entre personnes. Les applications de Voix sur IP VoIP – dont on peut définir une variante, la VVoIP (Video et Voix sur IP) [VCC07] – sont l'objet de cette section. La téléphonie sur IP [OuP07] est un ensemble de technologies qui permet de communiquer par la voix et/ou l'image via Internet.

La téléphonie sur IP définit un protocole d'échange entre 2 terminaux au travers d'une infrastructure Internet relativement complexe. La différence majeure entre les protocoles de contrôle domestiques (cf. section 1) et les protocoles d'établissement de session entre terminaux est la localisation des échanges. Les applications de contrôle d'équipements ciblent principalement la découverte et le contrôle d'un équipement par un autre sur le même résaue local alors que la téléphonie sur IP cible principalement des communications entre équipements situés sur des réseaux différents souvent séparés par le réseau Internet. Les défis sont alors l'adressage de terminaux parmi un nombre gigantesque et l'établissement de session au travers de topologies variées.

Une autre caractéristique remarquable, qui rejoint cette fois les caractéristiques de certains protocoles de contrôle sur les réseaux locaux (e.g., UPnP AV [UAV02], iTunes [StC05]), est la séparation entre protocoles de signalisation et protocoles d'échanges de flux de données. Les préoccupations d'établissement de session et d'échanges de flux peuvent être considérées comme des préoccupations orthogonales. La signalisation est l'ensemble des étapes indépendantes de l'échange par la voix et/ou l'image pour inviter un interlocuteur à débuter une conversion (initialisation de session), fournir des informations sur l'état des terminaux (suivi de session, e.g., correspondant occupé, sonnerie du téléphone), négocier le type, le format de flux multimédia échangés et terminer une conversation (terminaison de session). Les protocoles d'échanges de flux offrent une variété de solutions d'échange de flux face au besoin de compression, de qualité, de contrainte de ressources, etc.

2.1 SIP, principes et fonctions avancées

SIP [SIP02] (Session Initiation Protocol), est un protocole de signalisation spécifié par l'IETF (Internet Engineering Task Force). Ce protocole est utilisé pour créer, modifier,

terminer des sessions entre un ou plusieurs participants. La spécification définit premièrement des principes et une architecture pour gérer l'établissement de sessions entre participants. Des fonctions avancées permettent la mobilité des sessions.

2.1.1 Principes et architecture

Une infrastructure est définie afin de localiser les participants et relayer les communications quelque soit la topologie du réseau. SIP définit un réseau homogène se superposant aux différentes technologies complexes sous-jacentes (overlay network). Les messages SIP transitent à travers des serveurs Proxy en charge de les relayer vers la destination finale. Les serveurs Proxy utilisent les serveurs d'enregistrement, appelés Registrar, afin de résoudre les adresses SIP des participants. SIP utilisent le format URI (Unique Resource Identifier) pour les adresses des personnes associées à des terminaux. Ces adressent suivent le même format que les adresses e-mail : user@domain. Les messages de signalisation sont peu nombreux (invite, ok, ack, bye, options) et sont échangés de manière asynchrone entre clients SIP, appelés User Agent (UA).

Le protocole SIP n'étant qu'un protocole de signalisation, il ne prend en charge ni la qualité de service (QoS), ni le transport des flux audio/vidéo échangés par les participants de la session; en général, ce transport est assuré par le protocole RTP (Real-time Transport Protocol, IETF RFC 3550). SIP s'intègre également à d'autres protocoles tels que :

- RTCP (Real-time Transport Control Protocol, IETF RFC 3550), qui fournit des informations dynamiques sur l'état du réseau.
- RTSP (Real-time Streaming Protocol, IETF RFC 2326), pour contrôler la diffusion de flux multimédias en temps réel.
- RSVP (Resource reSerVation Protocol, IETF RFC 2205), pour obtenir des garanties de qualité de service et effectuer des réservations de ressources.

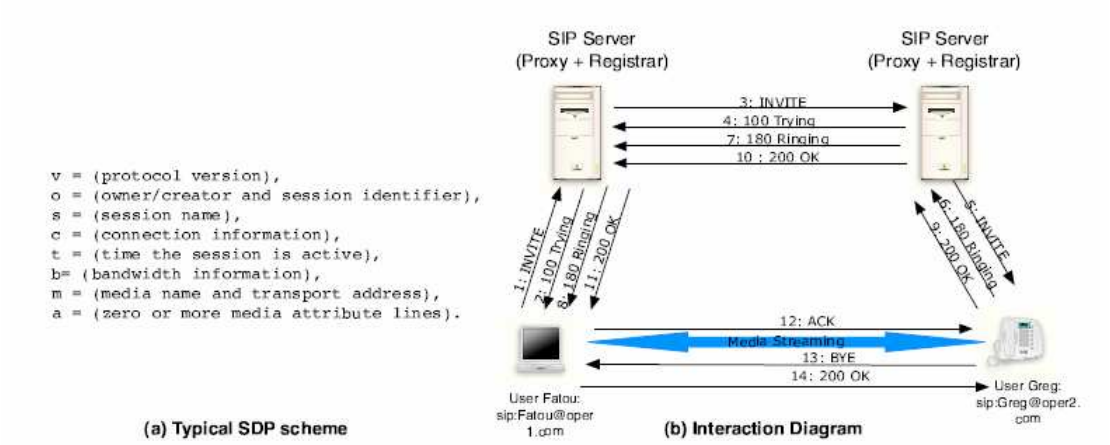


Figure 23 Principes et architecture d'une session SIP

La partie droite de la Figure 23 décrit les interactions nécessaires à l'établissement d'une communication par SIP. Dès que l'appelé est localisé les UAs échangent des messages de signalisations pour négocier les paramètres de la session à établir. La localisation du destinataire et le routage des messages jusqu'à lui est géré de manière transparente par l'infrastructure SIP (Serveurs Proxy et Registrar). La négociation de la session s'effectue par échange de propriétés entre terminaux. Les paramètres possibles sont décrits dans le format SDP (Session Description Protocol, IETF RFC 2327). L'objectif de la négociation est d'établir une session dont les paramètres conviennent pour chaque partie. Ce contrat décrit les paramètres possibles pour chaque type de flux multimédia (e.g., video, audio), le protocole de transfert de flux (e.g., RTP, HTTP, H.320) et le format du flux (e.g., MPEG, H.261), les

adresses IP et les ports d'émission ou de réception. La partie gauche de la Figure 23 montre le format d'une description SDP.

2.1.2 Fonctions avancées pour la mobilité de session

Dans cette section sont exposés les atouts intéressants des spécifications protocolaires attachées au protocole SIP qui sont utiles à la composition contextuelle de services multimédia : les méthodes Re-INVITE et REFER, les mécanismes 3PCC et le contrôle de transcodeurs sont décrits. Ces méthodes sont utilisées dans une des expérimentations de cette thèse et dans les travaux concurrents (cf. section Chapitre II.4.6.2).

Le contrat d'établissement de session peut être renégocié à tout moment grâce à la méthode Re-INVITE. Cette méthode est une méthode INVITE intervenant dans une communication déjà établie. Elle répond aux cas d'utilisations demandant des changements de configuration : ajout ou suppression d'un flux video, mobilité de terminal (en liaison avec l'utilisation du mécanisme 3PCC décrit ci-dessous). Quelques exemples de renégociation de session par utilisation de Re-INVITE avec proposition de paramètres de session différents sont proposés dans le RFC IETF 4317 intitulé "Session Description Protocol (SDP) Offer/Answer Examples". Ce dernier montre que les configurations d'une session gérant de multiples flux est possible. Cette possibilité de session multi-flux est particulièrement pertinente dans les scénarios d'éclatement du terminal de communication en un système constitué de plusieurs équipements domestiques (cf. Chapitre II.4.1.2). Cela nécessite toutefois que les équipements de part et d'autre soient compatibles avec une description SDP à flux multiples. Habituellement, les équipements SIP gère une session constitué d'un flux audio et d'un flux vidéo. Plusieurs flux d'un même type ne sont généralement pas possibles.

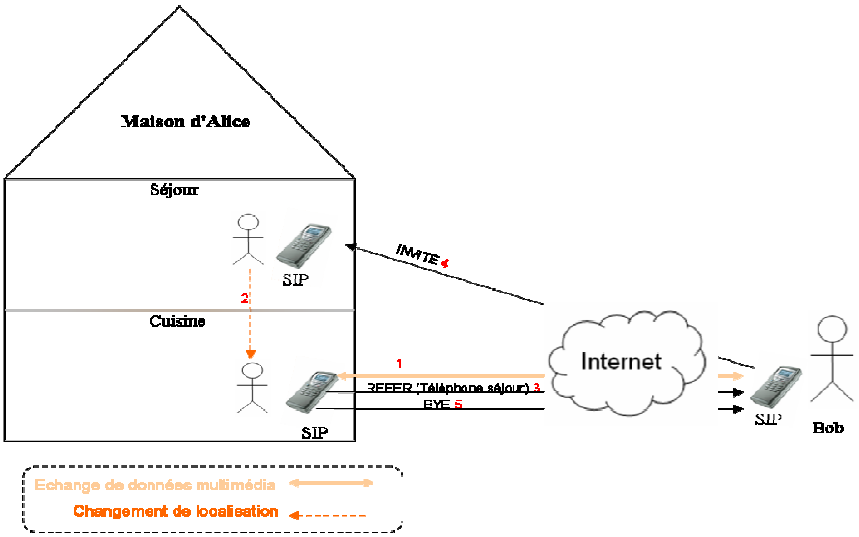


Figure 24 Premier mode d'utilisation de la méthode REFER

La méthode REFER (IETF RFC 3515 appelé Call Transfer Protocol) constitue un premier moyen pour réaliser la mobilité de session SIP. La méthode peut être utilisée de deux manières différentes. La différence repose sur le choix du participant pour le rôle de "transféré", soit le participant qui reçoit la requête de transfert d'appel (méthode REFER). Dans le sénario, Alice possède une maison composée d'au moins les deux pièces suivantes : le séjour et la cuisine. Un téléphone SIP est posé dans la cuisine où se trouve Alice au début du scénario. Le séjour est équipé d'un système sophistiqué, constitué d'une télévision et d'un système "home cinema", qui se comporte comme un téléphone SIP sur le réseau. Quant à Greg, il dispose d'un téléphone SIP.

Alice et Bob sont deux participants en cours de communication (1). Alice utilise d'abord le téléphone SIP dans la cuisine. Dans le premier mode d'utilisation de la méthode

REFER (cf. Figure 24), lorsqu'Alice passe dans le séjour (2), une demande de transfert d'appel (méthode REFER) vers l'équipement SIP du séjour est envoyée à Bob par le téléphone de la cuisine (3). Bob envoie donc un message d'invitation à l'équipement du séjour (4), une session est initialisée entre les deux et enfin, le téléphone de la cuisine met fin à sa session avec Greg. Ici le téléphone de Bob joue le rôle de transféré.

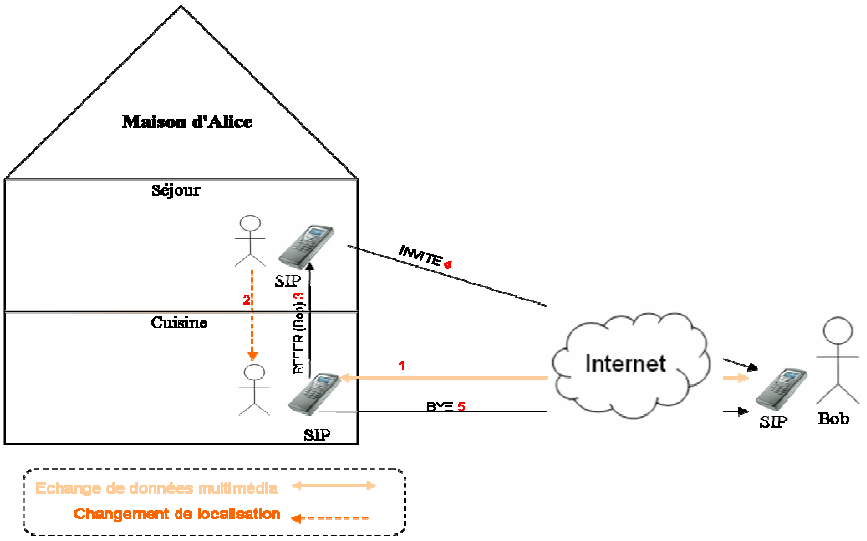


Figure 25 Deuxième mode d'utilisation de la méthode REFER

Dans le deuxième cas (cf. Figure 25), lorsqu'Alice passe dans le séjour (2), une demande de transfert d'appel vers le téléphone de Bob est envoyée à l'équipement du séjour par le téléphone de la cuisine (3). Bob est donc invité à initier une session par l'équipement du séjour, et enfin, le téléphone de la cuisine met fin à sa session avec Greg.

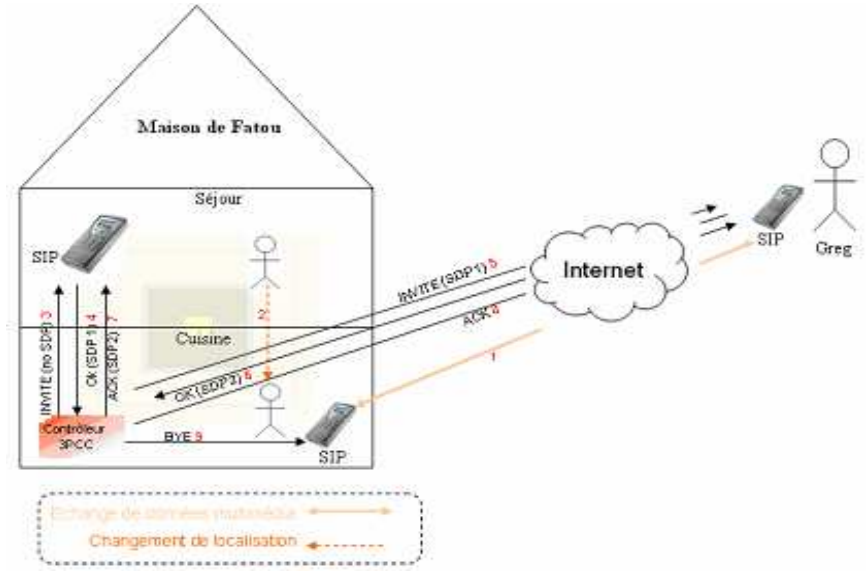


Figure 26 Mobilité de session par utilisation du mécanisme 3PCC

La Figure 26 décrit le même scénario de mobilité par utilisation du mécanisme nommé 3PCC (Third-Party Call Control, IETF RFC 3725). Alice et Bob sont en cours de communication (1), ils ont été mis en relation par le contrôleur 3PCC, installé dans la maison de Alice qui est à présent dans la cuisine. Elle utilise donc le téléphone de la cuisine. Lorsqu'elle se déplace dans le séjour, le contrôleur transfère la communication sur l'équipement du séjour (2). Ce dernier procède comme suit : il envoie un message d'invitation (requête INVITE) à l'équipement du séjour (3). Ce message d'invitation ne contient aucune

description de session. A ce message d'invitation il reçoit une réponse OK qui contient la description de session de l'équipement du séjour (4). Le contrôleur envoie un nouveau message d'invitation à Bob avec la description de la nouvelle session envoyée par l'équipement du séjour (5). Ce nouveau message d'invitation correspond à la méthode re-INVITE. Le téléphone de Bob répond OK à la réinvitation avec de nouveau ses paramètres de session (6). Le contrôleur envoie un message d'acquittement (requête ACK) au téléphone de Alice avec les paramètres de session de Bob (7). Ensuite, il envoie un message d'acquittement au téléphone de Bob également (8). Si tout s'est bien passé, le contrôleur envoie enfin une requête BYE au téléphone de Alice situé dans la cuisine(8). La conversation est transférée sur l'équipement du séjour.

Une autre spécification de l'IETF – RFC 4117, Transcoding Services Invocation in the Session Initiation Protocol Using 3PCC – adresse la gestion des différences de capacités des terminaux multimédia. Elle complète le protocole de négociation des paramètres de session attaché au protocole SDP dans le cadre de la gestion de l'hétérogénéité des protocoles de transfert de flux et de format des flux. En effet, si la négociation échoue et montre que 2 terminaux ne possèdent pas de protocoles de transfert et de format des flux communs pour au moins un type de ligne multimédia (e.g., audio, video), il est naturel de mettre en œuvre les capacités de transcodage disponibles sur le réseau. Le RFC 4117 repose sur la définition du contrôle d'un transcodeur par le protocole SIP. La Figure 27 décrit le diagramme d'interaction entre deux participants A et B d'une communication et un transcodeur où l'un des participants est contrôle le transcodeur T. A envoie un message INVITE à B (1) qui, constatant que la description SDP de A n'est pas compatibles avec ses capacités, envoie une INVITE à T avec sa description SDP et celle de A (2). T répond à B avec un message OK lui décrivant sa réponse SDP pour la session entre T et B et la réponse SDP que B doit envoyer à A (3). La première réponse SDP permet la session multimédia entre Tet B, B envoie alors une réponse ACK à T (4). B envoie la 2ème réponse SDP dans un message OK à A (5) qui répond ACK à B (6). Les paramètres SDP donnés à A dirige les flux de A à l'adresse de T. Les flux établis finalement entre A et B passent par les traitements du transcodeur T.

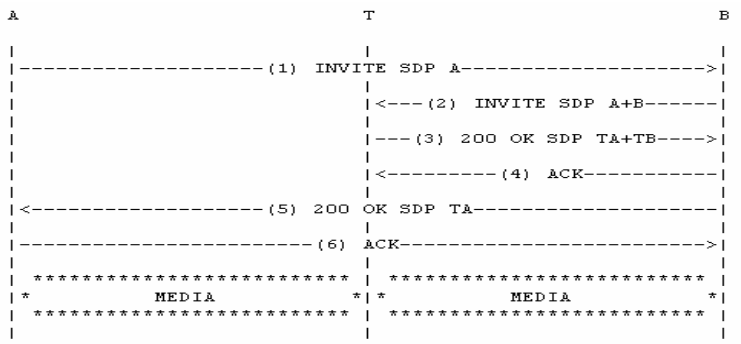


Figure 27 Interaction avec un transcodeur contrôlable par le protocole SIP

2.2 Jingle, H.323 et autres protocoles concurrents

Ici, les protocoles de signalisation dont les noms suivent sont décrits brièvement : Jingle (Jabber/XMPP), H.323. D'autres protocoles existent : MSN, Yahoo!, IRC.

2.2.1 Jingle

Jingle est le protocole de signalisation destiné à l'initialisation, le suivi et la terminaison de session multimédia entre clients Jabber. Jabber est un système standard ouvert de messagerie instantanée et de présence. Jabber est également une application de téléphonie sur IP qui supporte la VoIP et devrait normalement, dans un futur proche, supporter la VVoIP.

Le protocole Jingle prend non seulement en charge la signalisation mais aussi le transport de données multimédia. A ce jour, le protocole Jingle demeure encore assez jeune. Toujours en cours d'élaboration, les différents éléments du protocole sont repartis en trois catégories. En premier lieu, le XEP 166 constitue la brique de base du protocole, il ne définit que la sémantique d'une session. Ensuite, pour chacune des méthodes de transport des données reconnues, il existe un protocole XEP dédié. Enfin, pour chacun des formats de données reconnus, il existe de même un format XEP dédié.

2.2.2 H.323

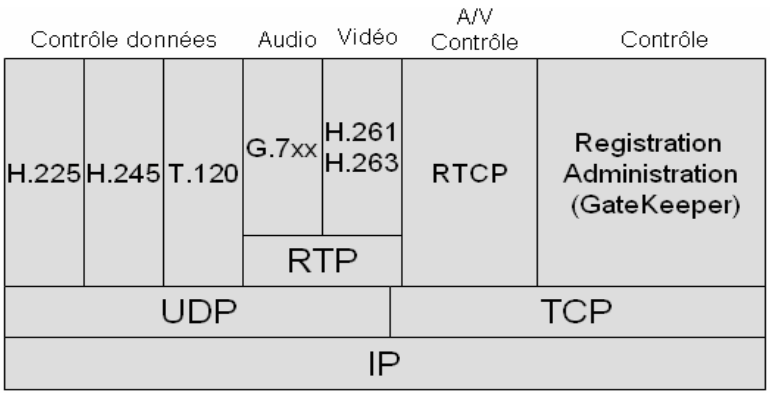


Figure 28 Agencement des protocoles utilisés par le protocole H.323

H.323 est un ensemble protocolaire couvrant la communication par la voix et l'image sur IP. Cet ensemble est développé par l'UIT-T (Union Internationale des Télécommunications). De nombreuses offres de téléphonie sur IP grand public s'appuient sur H.323.

Le protocole H.323 gère toutes les étapes d'une communication sur IP en s'appuyant sur d'autres protocoles. Il assure la signalisation, la négociation et le transport de données. La Figure 28 illustre l'agencement de ces protocoles pour former le protocole H.323.

3 ADMINISTRATION D'EQUIPEMENTS

3.1 Définitions

L'administration d'équipements – appelé "Device Management" en anglais – recouvre les activités de mise en cohérence du logiciel des équipements avec l'environnement et les désirs de l'utilisateur. L'administration des équipements est une activité de provisionnement logiciel et de suivi parallèle à l'application des logiciels déployés. L'administration des équipements est une activité de contrôle particulière. Ces deux aspects applicatifs proches, dont les moyens associés au second servent parfois au premier, sont le plus souvent considérés comme des préoccupations orthogonales. Dans le modèle de composants Fractal par exemple [BLC02], l'interface de contrôle du cycle de vie est un connecteur "vertical" dit non-fonctionnel sur les composants alors que les interfaces dites fonctionnelles sont décrites par des connecteurs horizontaux.

3.1.1 Les fonctions de l'administration d'équipement

L'administration d'équipements regroupe les fonctions suivantes :

- Déploiement logiciel : le déploiement concerne autant la mise à jour d'images micro-logicielles (firmware) que la gestion du cycle de vie de modules logiciels constituant une application. Le provisionnement – "provisioning" en anglais –

est la première étape de déploiement effectuée alors qu'aucune unité logicielle n'était présente auparavant.

- Configuration : la configuration des équipements et de leurs applications est le premier usage des protocoles d'administration qui est demandé et spécifié dans les organismes de normalisation : Broadband Forum (ex-DSL Forum www.broadband-forum.org) et le protocole TR-69, OMA (Open Mobile Alliance www.openmobilealliance.org) et le protocole OMA DM, etc.
- Diagnostique et supervision de performance : les opérations de diagnostique et de supervision de performance permettent à l'administrateur de détecter et les erreurs et leur cause sur les équipements. Le diagnostique et la supervision de performance se distinguent principalement par l'acteur initiant les opérations : le diagnostique recouvre un ensemble de requêtes ponctuelles actives de la part de l'administrateur alors que la supervision de performances est habituellement une collecte passive d'information sur une longue durée au cours de l'exécution des applications sur les équipements.

3.1.2 Couches protocolaires

Les solutions d'administration d'équipements s'appuient sur des moyens similaires à ceux des intergiciels distribués :

- Découverte : afin d'administrer un parc d'équipements, l'administrateur se doit d'en faire l'inventaire. Un protocole d'adressage et un protocole de découverte sont donc requis (cf. Chapitre II.2).
- Description : l'inventaire des équipements s'accompagne de la description des capacités des équipements et de la description de leurs données de configuration.
- Administration active et événementielle : les actions de déploiements, de configuration, de diagnostique et de supervision reposent sur l'utilisation d'un protocole de contrôle. L'administration active se distingue de la supervision par événement par l'envoi de requêtes à l'initiative du superviseur.
- Aspects non-fonctionnels : des considérations non-fonctionnelles sont associées à l'administration d'équipements. La cohérence des logiciels effectivement déployés et des données effectivement configurées lors d'une session d'administration requiert le suivi de propriétés transactionnelles : atomicité, cohérence, isolation, durabilité [OzV99]. Les actions de déploiement logiciel et de configuration apportent des changements critiques aux applications actives sur les équipements. Par conséquent, des moyens de sécurité assureront l'authentification, la gestion des autorisations et le contrôle de l'intégrité des données en rapport avec l'aspect critique des cas d'utilisation.

3.1.3 Opérations d'administration et modèles de données

La particularité des protocoles d'administration est leur Orientation Donnée (data-oriented protocols). Tous les protocoles existants d'administration – Broadband Forum TR-69 [TR6904], OMA DM [OMAD05], DMTF WS-Management [WSMa06], IETF Netconf [Netc06], OASIS MUWS [MUWS05], SNMP [SNMP02], JCP JMX [JMX02] – ne spécifient qu'un ensemble simple et réduit d'opérations dans une interface partagée par tout serveur et client. L'ensemble minimal d'opérations vise la manipulation générique d'un modèle hiérarchiques de données dont certaines hypothèses sont imposées sur la syntaxe générale. Chaque standard normalise ou référence une syntaxe propre. Quatre opérations sont généralement définies avec différentes variantes syntaxiques selon les standards :

- Get : une opération est définie pour l'obtention des valeurs des paramètres du client administré.
- Set : une opération est définie pour l'affectation de valeurs à des paramètres du client administré.
- Create : une opération est définie pour la création de paramètres ou pour la création d'instances d'objet dans des tables multi-objets. Par exemple, ajouter un serveur DNS dans une liste de serveurs DNS requiert l'utilisation de l'opération "AddObject" dans le protocole TR-69.
- Delete : une opération est définie pour la suppression de paramètres ou pour la suppression d'instances d'objet dans des tables multi-objets.
- GetDataModels : une ou plusieurs opérations sont définies pour l'obtention du nom des modèles de données et de l'ensemble des paramètres supportés par le client administré.

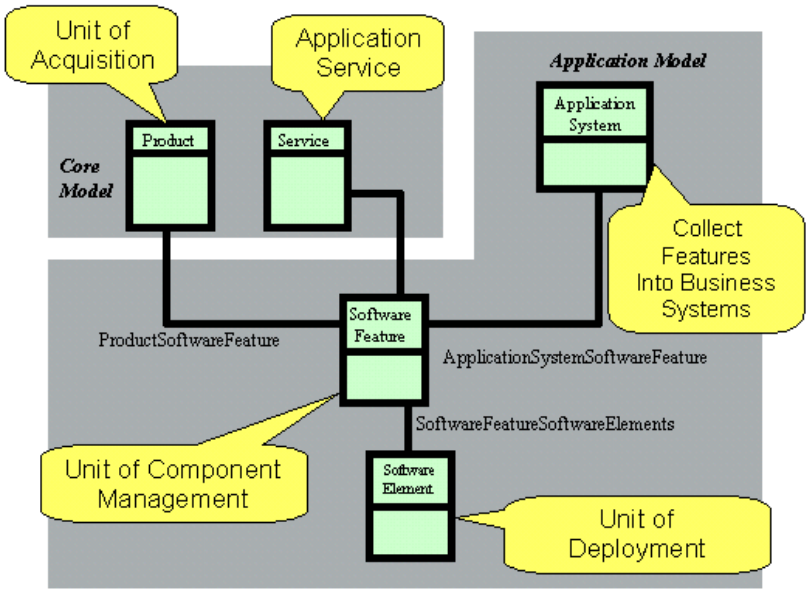


Figure 29 Extrait du modèle de données DMTF CIM

Le modèle hiérarchique de données permet à tout client ou type de client de se décrire spécifiquement selon une syntaxe normalisée. Toutes les données administrées propres à chaque ensemble de client – et parfois chaque client – peuvent être l'objet de standards additionnels – par exemple, la spécification TR-140 du Brodband Forum pour les serveurs de stockage résidentiels (NAS ou Network-Attached Storage) ou les spécifications OMA SCOMO [SCOM08] et DMTF CIM qui reprennent une syntaxe établie pour décrire des modèles généraux pour la gestion de modules logiciels. La Figure 29 montre un extrait du modèle objet représentant le modèle d'administration d'application DMTF CIM. Ce modèle objet a une correspondance dans un arbre de données hiérarchique en syntaxe XML.

Les protocoles d'administration appliquent ces opérations génériques sur des données ayant trait à la configuration, à la gestion de cycles de vie logiciels, au diagnostic et à supervision de performances. Parfois même, en particulier pour la gestion de cycles de vie logiciels, l'opération Set (ou Create) est utilisée dans un sens proche de l'exécution de commandes. Les données inscriptibles semblent alors correspondre à des arguments d'entrées et les données en lecture seule à des arguments de sortie dont l'application par l'opération Set (ou Create) déclenche, par un effet de bord voulu, une commande sur l'équipement. Par exemple, la spécification TR-106 du Broadband Forum définit quelques opérations de diagnostique de la façon dont est construite l'opération de Ping dans le Tableau 4.

".LAN.IPPingDiagnostics." définit un nœud de l'arbre de données, les autres lignes du tableau correspondant aux paramètres-feuilles de l'arbre. Dans ce modèle, l'opération Set mettant la valeur du paramètre nommé "DiagnosticsState" à "Requested" ordonne à l'équipement d'effectuer une opération de "Ping" à destination de l'hôte nommé "host" avec un nombre de répétitions, un temps maximum, une taille de données et un code DSCP donnés par les autres paramètres inscriptibles (W : Writable).

Tableau 4 L'objet .LAN.IPPingDiagnostics. de la spécification TR-106 du Broadband Forum

Name	Type	Write	Description
.LAN.IPPing-Diagnostics.	object	-	This object defines access to an IP-layer ping test for the default IP interface.
Diagnostics-State	string	W	Indicates availability of diagnostic data. One of: “None”, “Requested”, “Complete”, “Error_CannotResolveHostName”, “Error_Internal”, “Error_Other”, …
Host	string(256)	W	Host name or address of the host to ping.
NumberOfRe-petitions	unsignedInt[1:]	W	Number of repetitions of the ping test to perform before reporting the results.
Timeout	unsignedInt[1:]	W	Timeout in milliseconds for the ping test.
DataBlockSize	unsignedInt [1:65535]	W	Size of the data block in bytes to be sent for each ping.
DSCP	unsignedInt [0:63]	W	DiffServ codepoint to be used for the test packets. …
SuccessCount	unsignedInt	-	Result parameter indicating the number of successful pings (those in which a successful response was received prior to the timeout) in the most recent ping test.
FailureCount	unsignedInt	-	Result parameter indicating the number of failed pings in the most recent ping test.
AverageRes-ponseTime	unsignedInt	-	Result parameter indicating the average response time in milliseconds over all repetitions with successful responses of the most recent ping test. …
MinimumRes-ponseTime	unsignedInt	-	Result parameter indicating the minimum response time in milliseconds over all repetitions with successful responses of the most recent ping test. …
MaximumRes-ponseTime	unsignedInt	-	Result parameter indicating the maximum response time in milliseconds over all repetitions with successful responses of the most recent ping test. …

3.1.4 Approches génériques ou spécifiques aux équipements

Si les opérations d'administration sont peu nombreuses, leur généricité permet la gestion d'ensembles variés de données. Les standards existants restreignent le type de modèles de données manipulé plus ou moins fortement. Certains standards comme SNMP, TR-69, OMA DM, définissent leur propre syntaxe alors que Netconf et WS-Management ne vont imposer que la conformité des modèles de données à la syntaxe XML. Les premiers suivent une approche dite "générique" alors que les seconds suivent une approche "spécifique équipements". La généricité contraint les équipements à convertir leurs modèles dans la syntaxe dite générique. Dans le cas générique, un serveur peut ne pas connaître le détail du contenu mais reconnaitra toujours la syntaxe des données échangées (cf. les nœuds génériques "parameter", "name", "value" de la Figure 30). Au contraire, Netconf et WS-Management tentent de demeurer agnostiques aux modèles de données tant que ceux-ci demeurent XML. Les données échangées sont traitées comme des données opaques dont la syntaxe n'est reconnue que si le serveur et le client partagent les mêmes modèles de données (cf. Figure 31). A des fins de validation de l'extrait XML passé dans le message de réponse, des espaces de noms (namespaces) sont passés en premier argument de retour (valeur xmlns).

```
Get(“/Device/ProcessStatus/Process”)
GetResponse("
```

```
<parameter>
        <name>ManageableDevice.ProcessStatus.Process.3.PID</name>
        <value>1233</value>
</parameter>
<parameter>
        <name>ManageableDevice.ProcessStatus.Process.3.Name</name>
        <value>process_1233</value>
</parameter>
<parameter>
        <name>ManageableDevice.ProcessStatus.Process.5.PID</name>
        <value>4555</value>
</parameter>
<parameter>
        <name>ManageableDevice.ProcessStatus.Process.5.Name</name>
        <value>process_4555</value>
</parameter>
```

Figure 30 Réponse conforme à un format générique à une requête de type Get

```
Get(“/Device/ProcessStatus/Process”)
GetResponse("xmlns="urn:example.org/device/",
        "<process>
                <PID>1233</PID>
                <Name>process_1233</Name>
        </process>
        <process>
                <PID>4555</PID>
                <Name>process_4555</Name>
        </process>
")
```

Figure 31 Réponse au format original de l'équipement à la même requête

Certains modèles de configuration sont hybrides. Ils définissent un cadre générique où des objets de l'arbre sont de contenu libre. C'est le cas par exemple du modèle de configuration définit par UPnP pour les équipements de stockage audio-video. En effet, le modèle de données DIDL-Lite définit des nœuds aux noms génériques ("container" et "item") au niveau le plus haut, et les éléments XML à l'intérieur du nœud "item" sont pour partie librement définissables par des acteurs externes. Si des types d'éléments sont ajoutés, des préfixes (dc, upnp, etc.) identifient les espaces de noms les définissant (cf. Figure 32).

Plus le format imposé est simple, plus il sera aisé de spécifier les opérations de manipulation des données sans ambigüité. Au contraire, plus il est ouvert, plus la spécification sera complexe afin de couvrir toutes les situations. Alors que les opérations sur le système de paires noms-valeurs du Broadband Forum est relativement simple, les opérations sur les fragments XML définis par Netconf et WS-Management ne sont pas intuitives d'un premier abord.

```
CreateObject ("10", "
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
 xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
        urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
         http://www.upnp.org/schemas/av/didl-lite.xsd
        urn:schemas-upnp-org:metadata-1-0/upnp/
         http://www.upnp.org/schemas/av/upnp.xsd">
        <item id="" parentID="10" restricted="0">
                <dc:title>New Song</dc:title>
```

```
                <dc:date>1990-01-01</dc:date>
                <upnp:class>
                        object.item.audioItem
                </upnp:class>
                <res protocolInfo="*:*:*:*"/>
        </item>
</DIDL-Lite>")
```

Figure 32 La demande de création d'un objet au format DIDL-Lite

A contrario, les approches génériques imposent la conversion des modèles de données dans la syntaxe imposée afin que les données soient échangeables. Les approches spécifiques aux équipements, aux modèles plus ouverts, permettent de garder les modèles de données des équipements intacts (ou quasi-intact).

3.1.5 Gestion du cycle de vie logiciel

La gestion des entités logicielles d'un équipement fait partie de son administration. Les entités administrées peuvent être variées, elles peuvent correspondre à des applications, des interfaces graphiques, des pilotes, des plugins d'un navigateur, etc. Les équipements les plus contraints ne définiront qu'une image logicielle, appelée micrologiciel ou firmware, qui ne pourra être que mise à jour entièrement. D'autres équipements procureront les moyens d'administrer plus finement les modules logiciels qui sont chargés au-dessus d'un serveur d'exploitation (Windows, Linux, etc.) ou d'une machine virtuelle (Java, .NET, etc.).

Les environnements d'exécution d'applications définissent habituellement les termes suivants. Le Tableau 5 compare différentes technologies selon certains de ces critères :

- Paquetage de déploiement : une unité binaire empaquetant une à plusieurs unités de déploiement permettant généralement l'installation d'une application complète. Un paquetage peut être téléchargé et retiré de la plateforme.
- Unité de déploiement : une unité binaire qui peut être déployée dans l'environnement – installé, désinstallé, mis à jour.
- Unité d'exécution : une unité logique qui peut être démarrée ou arrêtée.
- Dépendances : des dépendances de ressources – librairies, fichiers, etc. – peuvent être définies entre des unités de déploiement et entre unités de déploiement et unité d'exécution. Les unités d'exécution s'appuient sur des unités de déploiement et sont souvent contenus par ces unités de déploiement – les bundles OSGi, les assemblies .NET, les MIDlet Suites peuvent contenir des unités que l'on peut démarrer et arréter.
- Métadonnées : des données qui décrivent les entités logicielles – environnement d'exécution, dépendances, configuration, etc.
- Opérations de gestion de cycle de vie
  - des paquetages de déploiement : téléchargement, suppression, installation.
  - des unités de déploiement : installation, désinstallation, mise à jour, etc.
  - des unités d'exécution : démarrer, stopper, mettre en pause, reprendre.
- Etats
  - des paquetages de déploiement : téléchargé, supprimé.
  - des unités de déploiement : installé, désinstallé, résolu.
  - des unités d'exécution : démarré, arrêté. Dans certains modèles particuliers, un état de pause peut être défini.

- Evénements : ils sont conséquents aux changements d'états. Les événements peuvent être notifiés à des gestionnaires particuliers ("Java Application Manager" de MIDP), aux entités d'un domaine d'application (.NET) ou à toutes les entités de l'environnement (OSGi).
- Répertoires d'unités de déploiement locaux (.NET Global Assembly Cache) ou distants (OSGi bundle repository, RedHat RPM repositories)
- Répertoires d'unités d'exécution locaux (Répertoire de services OSGi, inventaire OMA SCOMO, etc.)

Tableau 5 Comparaison des technologies de plateformes d'exécution

Technology	Delivery Package	Deployment Unit	Execution Unit	Dependency	Actions	Events
OSGi	Deployment Component (spécification Mobile)	Bundle	Bundle	Bundle, Package, Service	Install, Start, Stop, Update, Uninstall	Installed, Starting, Resolved, Active, Uninstalling
Java MIDP	NA	Midlet Suite	Midlet	Library (MIDP3)	StartApp, DestroyApp	NA
.NET	Windows installer	Assembly	Assembly	Assembly	Download, Load, Unload (AppDomain), Invoke	NA
Linux Debian	Meta-Package	Package	RC Script, possibly others	Package	PackageInstall, PackageUninstall, ServiceStart ServiceStop	Triggering Updates
SCOMO	Delivery Package	Deployment Component	NA	NA	Download, DownloadInstall, DownloadInstallInactive, Install, InstallInactive, Update, Remove, Activate, Deactivate	Operational Results

3.2 Protocoles d'administration d'équipements

De nombreux protocoles d'administration existent : Broadband Forum TR-69 [TR6904], OMA DM [OMAD05], SNMP [SNMP02], DMTF WS-Management [WSMa06], IETF Netconf [Netc06], OASIS MUWS [MUWS05], JCP JMX [JMX02]. Les trois premiers sont probablement les plus importants sur le marché du réseau domestique. Le quatrième est intéressant pour son alignement sur la technologie des Web Services et sa complémentarité avec l'ensemble protocolaire DPWS [DPWS06].

3.2.1 Spécifications du Broadband Forum : TR-069, TR-106, etc.

Le Broadband Forum (ex-DSL Forum, www.broadband-forum.org) est une organisation fondée en 1994 et qui compte plus de 200 membres aujourd'hui. Depuis lors, de nombreuses spécifications se sont répandues, non seulement sur le marché des réseaux et équipements DSL mais aussi sur d'autres types de technologies réseau. Si son succès repose aujourd'hui principalement sur la gestion des passerelles Internet domestiques, ses spécifications touchent la gestion de nombreux équipements du réseau domestiques (Set-Top-Box, applications VoIP, etc.).

Le protocole d'administration qui se répand aujourd'hui est le protocole TR-069 (Technical Report 069) [TR6904] appelé CPE WAN Management Protocol (CWMP, cf. Figure 33). Ce protocole, associé notamment au TR-106, normalise un modèle de données générique et des opérations sur ce modèle de données. Il adresse les fonctions principales de l'administration d'équipements (cf. section 3.1.1) en restreignant la gestion du cycle de vie logiciel à celle du micrologiciel et en ouvrant la supervision à la gestion des fichiers de journalisation.

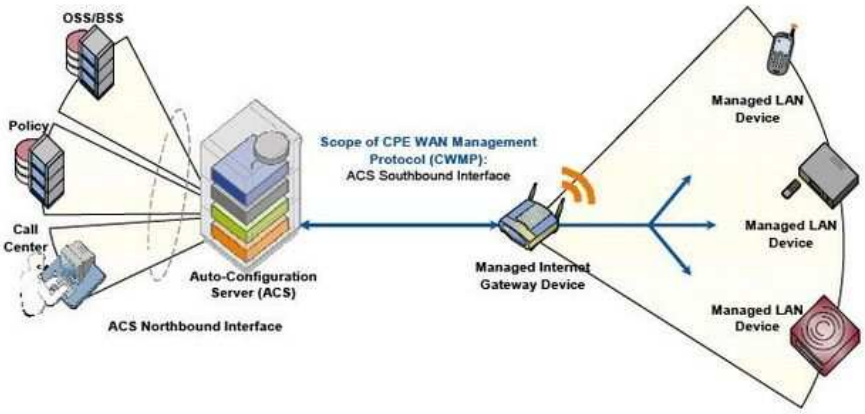


Figure 33 Architecture d'administration de bout en bout du Broadband Forum – www.broadband-forum.org

Les interactions sont définies entre un serveur d'administration distant nommé ACS (Auto-Configuration Server) et l'équipement administré nommé CPE (Customer Premises Equipment). Généralement, le CPE initie la connexion avec l'ACS. L'ACS ne peut initier la connexion que lorsque le CPE possède une adresse visible, ce qui est rarement le cas pour les équipements du réseau domestiques où les adresses ne sont visibles que localement. Les exceptions étant la passerelle Internet et les équipements d'un réseau IPv6. Les connexions permettent l'inventaire des équipements à administrer. Un modèle de données est défini afin de décrire ces équipements. Les connexions périodiques et événementielles entre le CPE et l'ACS permettent les envois d'opérations vers le CPE et la notification de l'ACS d'événements s'effectuant sur le CPE. La sécurité est restreinte à l'authentification d'un unique ACS et du cryptage éventuel des données échangées. L'ACS étant unique pour un CPE, l'aspect transactionnel d'isolation n'est l'objet d'aucune spécification. En revanche, l'atomicité des opérations effectuées est l'objet de la spécification de session entre ACS et CPE.

Le modèle de données suit une approche générique. Chaque équipement est décrit selon un modèle de paires noms-valeurs (cf. le contenu de la réponse décrite dans la Figure 30). La hiérarchie du modèle se vérifie dans la hiérarchie des noms. Chaque paramètre a pour nom la concaténation d'un chemin énumérant les parents jusqu'au paramètre-feuille, le séparateur étant le '.'. Par exemple, le paramètre dont la valeur est le nombre d'octets envoyés depuis le dernier redémarrage de la passerelle Internet a pour nom "InternetGatewayDevice.LAN.Stats.TotalBytesSent".

Les opérations sur ce modèle de données sont :

- GetParameterValues : obtenir les valeurs de paramètres. Le nommage des éléments est toujours absolu (chemins définis à partir de la racine de l'arbre de données). Aucun opérateur de sélection n'est défini en supplément de ce nommage simple.
- SetParameterValues : modifier les valeurs de paramètres.
- GetParameterNames : obtenir les noms des paramètres définis.

- GetParameterAttributes : obtenir les attributs de paramètres. Les attributs principaux des paramètres sont le mode lecture seule ou écriture et le mode de notification de changements).
- SetParameterAttributes : modifier les attributs de paramètres.
- AddObject : ajouter un objet (sous-arbre) dans une table multi-objets.
- DeleteObject : supprimer un objet.
- Download : télécharger un fichier.
- Reboot : redémarrer l'équipement.

Les événements sont remontés selon deux opérations de notification

- Inform : notifier d'un changement de paramètre.
- TransferComplete : informer de l'achèvement d'un transfert.

Le succès des protocoles du Broadband Forum s'expliquent par la simplicité du modèle de données et du protocole TR-069 et évidemment par l'effort uni de lobbying des opérateurs auprès de leurs fournisseurs et partenaires au sein de l'importante organisation du Broadband Forum. Un des défauts de cet ensemble protocolaire est la non-adoption de celui-ci par les fabricants d'équipements domestiques autres que ceux délivrés dans les services des opérateurs (passerelle, Set-Top-Box). Les équipements multimédia et domotique demeurent administrés par d'autres protocoles ou sont exempts de protocoles d'administration. La création du Comité de Travail UPnP Execution Platform résulte en partie de ce manque. L'autre raison est la non-adéquation des protocoles existants tels que ceux du Broadband Forum à une administration des équipements par les équipements du réseau domestique eux-mêmes (applications de self-care).

3.2.2 OMA DM et SCOMO

OMA DM (Device Management) [OMAD05] et SCOMO (Software COmponent Management Object) [SCOM08] sont des spécifications de l'OMA (Open Mobile Alliance), un organisme fondé en Juin 2002 et comptant aujourd'hui plus de 200 membres. L'OMA normalise les protocoles et modèles de données utiles sur le marché du téléphone mobile.

L'OMA DM est un ensemble protocolaire qui permet la configuration, le diagnostic et la supervision de performances d'équipements embarqués. SCOMO ajoute à cet ensemble la gestion du cycle de vie des applications sur ces équipements. Tandis que l'OMA DM normalise un modèle de données générique et les opérations de manipulations de celui-ci, SCOMO normalise des modèles de module logiciel, leurs diagrammes d'états et les opérations de gestion associées.

De manière similaire au protocole TR-069, seul l'équipement administré initie les connexions avec le serveur. Un équipement peut être administré par plusieurs serveurs appelés RM (Remote Manager). Les opérations spécifiées sur le modèle de données permettent le contrôle synchrone des équipements et la notification des serveurs. OMA DM définit le moyen d'authentifier ces RMs, de leur affecter des droits dans des Listes de Contrôle d'Accès (Access Control Lists) et de garantir l'intégrité des messages. Les messages "Atomic" et "Sequence" sont définis afin de définir des séquences d'opérations atomiques et isolées.

Le modèle de données suit une approche générique. Chaque équipement est décrit selon un arbre de données retranscrit dans un modèle générique proche du systèmes de paires nom-valeur de TR-069. Un nœud possède une valeur qui peut être obtenue (Get), remplacée (Replace), ou exécuté (Exec) suivant la sémantique du nœud et des droits associés aux serveurs.

Les opérations sur ce modèle de données sont :

- Get : obtenir la valeur d'un nœud de l'arbre de données (feuille ou sous-arbre).
- Replace : remplacer la valeur d'un nœud de l'arbre.
- Add : ajouter un nœud à l'arbre.
- Delete : supprimer un nœud de l'arbre.
- Exec : exécuter la valeur d'un nœud de l'arbre
- Copy : copier ou déplace un nœud de l'arbre.
- Atomic : spécifier le caractère atomique de l'exécution des opérations subordonnées à cette commande.
- Sequence : spécifier un ordre à l'exécution des opérations subordonnées.

Les événements sont remontés selon l'opération de notification suivante :

- Alert

Le protocole OMA DM est en passe d'être progressivement déployé sur les équipements mobiles. L'effort uni des opérateurs et des fabricants de téléphones portables assure le succès de la spécification. Toutefois, sa provenance d'un protocole plus complexe, SyncML DM, et la relative complexité apparente des messages de configuration n'en font pas une spécification claire. SCOMO est intéressant dans le cadre des travaux du Comité UPnP Device Management car il résulte d'une démarche similaire de représenter un modèle général de modules logiciels qui s'adapterait à la plupart des modèles définis par les technologies de plateformes existantes. Comme le modèle SCOMO peut être comparé à ces modèles existants, SCOMO est décrit avec détails dans la partie intitulée Plateformes d'exécution (cf. section 3.3.5).

3.2.3 SNMP

SNMP (Simple Network Management Protocol) [SNMP02] est une spécification de l'IETF (Internet Engineering Task Force). Cette spécification apparut dans sa première version en 1988. Par la suite, deux autres versions vinrent l'améliorer. Aujourd'hui, de nombreux équipements sont administrables selon ce protocole, notamment les équipements de routages réseau et quelques équipements domestiques de type passerelle Internet et Set-Top-Box.

Les 3 versions de SNMP permettent la configuration, le diagnostic et la supervision de performances des équipements. Il n'adresse pas la gestion de cycle de vie logiciel. SNMP normalise un modèle de données de type générique et un ensemble réduit d'opérations pour la manipulation de celui-ci.

SNMP définit un modèle de description simple d'équipements, des opérations et des événements liés à sa manipulation. La version 3 de SNMP spécifie les aspects d'authentification et d'intégrité des messages. Le protocole permet l'administration d'un équipement par plusieurs administrateurs. En particulier, il définit l'atomicité de l'opération de modification de plusieurs paramètres (Set) et, de plus, un mécanisme de 2 phase commit pour la configuration de plusieurs équipements dans une seule session atomique. En revanche, le protocole n'indique pas la manière de découvrir les équipements décrits selon ce protocole.

Le modèle de données suit une approche générique similaire à celle du protocole TR-069 (cf. section 3.2.1). Chaque équipement est décrit selon un modèle arborescent tabulaire.

Les opérations sur ce modèle de données sont :

- Get : obtenir la valeur d'un paramètre.
- GetNext : obtenir la valeur du paramètre suivant dans un tableau de paramètres.
- GetBulk : obtenir la valeur de plusieurs paramètres dans un tableau.
- Set : modifier la valeur d'un ou plusieurs paramètres.

Les événements sont remontés selon l'opération de notification suivante :

- Trap : remonter une alerte à un administrateur.

De par sa simplicité et son ancienneté, ce protocole est encore utilisé aujourd'hui sur de nombreux appareils. Toutefois, le consensus grandissant autour de TR-069 sur le réseau domestique et d'OMA DM sur le réseau mobile font décliner son importance sur ces environnements.

3.2.4 DMTF WS-Management

WS-Management est un protocole récent qui repose sur de nombreux protocoles de la technologie Web Services. De façon a priori étonnante, ce protocole aligné sur les Web Services n'est spécifié ni par le W3C ni par OASIS. Le DMTF (Distributed Management Task Force) a pris l'initiative de cette spécification afin de consolider la promotion de ces nombreux modèles de données objets dont la traduction en tant que spécifications XML est disponible : CIM (Common Information Model), CDM (Common Diagnostics Model), SMBIOS (System Management BIOS), DASH (Desktop and mobile Architecture for System Hardware), etc.

Ce protocole permet la configuration, le diagnostic et la supervision de performance d'équipements embarqués. Il n'adresse pas la gestion de cycle de vie logiciel. De même que les autres protocoles décrits, WS-Management spécifie un ensemble réduit d'opérations pour la manipulation d'un modèle données.

WS-Management est associé à WS-Eventing pour la souscription et la résiliation de souscription à des événements, et à WS-Security et WS-Trust pour l'authentification et la garantie de l'intégrité des échanges. Il peut être associé à DPWS (cf. section 1.1.3) pour la découverte d'équipements. Rien n'est explicitement spécifié pour l'atomicité et l'isolation des opérations de modifications de l'arbre de données de l'équipement mais la spécification pourrait être associée à WS-AtomicTransaction pour la gestion des aspects transactionnels des échanges (Atomicité, Cohérence, Isolation, Durabilité). WS-AtomicTransaction définit en particulier un protocole "2 phase commit".

Le modèle de données des équipements doit seulement suivre le modèle XML général afin de pouvoir être administrés. L'approche est donc relativement spécifique aux équipements ("device-specific"). Peu de protocoles suivent cette approche (Netconf est un autre exemple) car la spécification et l'implémentation des opérations est plus complexe en raison de la généralité du modèle.

Les opérations sur ce modèle de données sont :

- Get : obtenir la valeur d'un paramètre ou d'un sous-arbre. La sélection du paramètre ou du nœud suit la spécification XPath, qui permet la définition d'un ensemble riche de filtres sur des données XML. La méthode peut aussi être utilisée pour obtenir la liste des espaces de noms XML (namespaces) supportés par l'équipement.
- Put : modifier la valeur d'un paramètre ou d'un sous-arbre.
- Create : créer une instance d'un objet dans une séquence d'objets.
- Delete : supprimer une instance d'un objet dans une séquence d'objets.

- Enumerate, Pull, Release : respectivement initier, continuer, annuler une énumération d'instances d'une séquence d'objets.

WS-Management est encore peu utilisé. Comme DPWS, ce protocole est déjà implémenté dans le serveur d'exploitation Microsoft Vista. La généralité du protocole, son alignement sur la technologie des Web Services et le succès des technologies XML sont ses atouts techniques. La puissance de vente de Microsoft étaie l'importance à venir de ce protocole sur le marché du réseau domestique et de l'entreprise.

3.3 Plateformes d'exécution

Les plateformes d'exécution intéressantes sont celles qui définissent ou du moins permettent le déploiement et l'exécution de modules applicatifs. Cette section décrit les plateformes OSGi [OSGi05], Java MIDP version 2 [MIDP02] et 3, Linux, Microsoft .NET, SCOMO [SCOM08] avec la grille des définitions de la section 3.1.5.

3.3.1 Bundles OSGi

La technologie OSGi définit une plateforme de déploiement d'application modulaire avancée au-dessus de Java. Loin de la cible initiale des passerelles Internet lors de la fondation du consortium en 1999, la définition de la technologie par l'Alliance OSGi elle-même a évoluée vers un objectif plus général : le système modulaire dynamique de Java ("OSGi technology is the dynamic module system for Java" est écrit sur la page d'accueil du site www.osgi.org). La technologie définit des mécanismes de partage et d'isolation de code à grain fin entre modules logiciels, appelés bundles. Grace à ces mécanismes et à un ensemble de bonnes pratiques, il est possible d'envisager des applications constituées de multiples bundles qui coexistent et partagent du code selon leurs affinités.

Voici les caractéristiques du cycle de vie logiciel des entités de la plateforme OSGi version 4.1 [OSGi05] :

- Paquetage de déploiement : un Deployment Package est défini dans le chapitre Deployment de la Spécification Mobile de l'Alliance OSGi ou JSR 232. Il représente un ficher jar contenant un manifest listant différentes ressources pouvant comprendre des bundles OSGi. Il permet de garantir l'installation, la mise à jour et la désinstallation atomique de ressources interdépendantes.
- Unité de déploiement : le bundle OSGi est un ficher JAR qui peut contenir des fichiers Java, d'autres fichiers JAR, un fichier Manifest décrivant les métadonnées du bundle. Il peut aussi contenir d'autres fichiers non spécifiques à OSGi.
- Unité d'exécution : tout bundle OSGi déclarant un fichier appelé "Activator" peut être démarré ou arrêté sur la plateforme. Les autres bundles sont des librairies de ressources inertes.
- Dépendances : le cœur de la spécification OSGi définit la déclaration de packages versionnés de code privés, publics, requis. La plateforme OSGi est responsable du diagnostic de résolution des dépendances de packages entre bundles. Un bundle non résolu ne peut pas démarrer. Des modèles à composants plus sophistiqués (e.g., OSGi Declarative Services) permettent la définition de composants intérieurs aux bundles et de dépendances de services.
- Métadonnées : les propriétés suivantes sont définies pour les bundles OSGi. On peut distinguer :
    - Des propriétés génériques optionnelles décrivant la provenance du bundle : Bundle-Category, Bundle-ContactAddress, Bundle-Copyright, Bundle-

Description, Bundle-DocURL, Bundle-Localization, Bundle-Name, Bundle-Vendor, Bundle-Version.

- Des propriétés spécifiques à la plateforme OSGi utiles au déploiement et à l'exécution du bundle : Bundle-ActivationPolicy, Bundle-Activator, Bundle-Classpath, Bundle-ManifestVersion, Bundle-NativeCode, Bundle-RequiredExecutionEnvironment, Bundle-SymbolicName, Bundle-UpdateLocation, DynamicImport-Package, Export-Package, Export-Service, Fragment-Host, Import-Package, Import-Service, Require-Bundle.

- Opérations de gestion de cycle de vie : tout bundle peut être installé, désinstallé, mis à jour, démarré, stoppé : Install, Uninstall, Update, Start, Stop. Toutefois, le démarrage et l'arrêt d'un bundle ne possédant pas d'activateur n'ont aucun effet. La spécification ne définit pas d'opération de résolution, les mécanismes de résolution sont donc déclenchés par la plateforme elle-même.

- Etats des unités de déploiement et des unités d'exécution : Installed, Resolved, Uninstalled, Active. L'état "Resolved" indique que le diagnostic de résolution de dépendance a été effectué et que les dépendances sont effectivement résolues. Les bundles OSGi passent par des états transitoires définis : Starting, Stopping (cf. Figure 34).

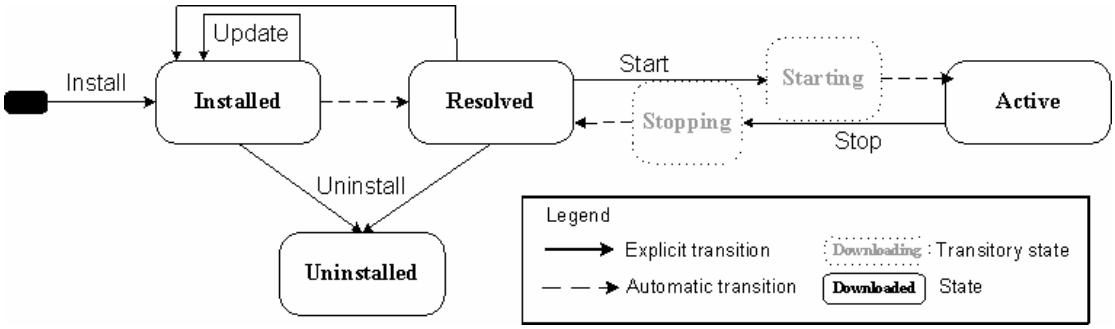


Figure 34 Cycle de vie du bundle OSGi

- Evénements : ils sont conséquents aux changements d'états : "Installed", "Uninstalled". L'activité d'un bundle est visible aussi par la notification des états des services fournis : "Registered", "Modified", "Unregistered".

- Répertoires d'unités de déploiement locaux et distants : bien que le format ne soit pas spécifié officiellement, un standard de facto décrit les répertoires de bundles : OSGi Bundle Repository. Ce format peut aussi être utilisé autant pour un répertoire distant que pour un répertoire co-localisé avec la plateforme OSGi.

- Répertoires d'unités d'exécution locaux : les plateformes OSGi répandues offrent un outil listant les bundles OSGi avec leur état et leurs métadonnées. L'originalité de la spécification OSGi est par ailleurs la définition d'un répertoire de services.

3.3.2 MIDlets Java

Le profil Java MIDP (Mobile Information Device Profile) définit des entités logicielles, pouvant se déployer sur une plateforme de déploiement contrainte, la machine virtuelle embarqué de la configuration Java CLDC (Connected Limited Device Configuration). MIDP a été l'objet de 3 versions successives. Ce qui est indiqué ici est vérifié par MIDP2 [MIDP02] et par MIDP3. Plus restrictif que le modèle OSGi, CLDC/MIDP spécifie un mode tout ou rien pour le partage de code entre entités logicielles. Ce mode est appelé modèle de bac à sable ("sandbox"). A l'intérieur d'une MIDlet Suite, unité de

déploiement MIDP, toute partie du code a visibilité sur le code de la plateforme et sur celui de toutes les autres parties. En revanche, aucune partie n'a de visibilité sur le contenu des autres MIDlet Suites. Si MIDP3 apporte la notion de partage de librairies de code (LIBlets) entre MIDlet Suites, elle respecte le modèle de bac à sable en ne permettant aucun partage d'objets à l'exécution (la librairie "partagée" est chargée en mémoire séparément par chaque MIDlet Suite). Seuls les appels interprocessus (IPC) sont permis entre MIDlet Suites à l'exécution. Chaque MIDlet Suite peut contenir plusieurs unités d'exécution appelées MIDlets. Le cycle de vie de chacune des 2 entités est décrit par la Figure 35.

Voici les caractéristiques de la plateforme CLDC/MIDP [MIDP02] :

- Paquetage de déploiement : aucune entité ne correspond exactement à ce concept. Une MIDlet Suite définissant avec ses LIBlets une application auto-contenue et la MIDlet Suite ayant peu d'interférences avec les autres MIDlet Suite, le paquetage de déploiement peut être confondu avec l'unité de déploiement. MIDP3 spécifie simplement que l'installation d'une MIDlet Suite implique l'installation des LIBlets dont elle dépend.
- Unité de déploiement : une MIDlet Suite est un fichier JAR contenant le code d'une ou plusieurs MIDlets, un fichier JAD (Java Application Descriptor) les décrivant et d'autres ressources non spécifiques à Java MIDP. Le fichier JAD définit aussi les dépendances à des LIBlets, autres unités de déploiement spécifiées.
- Unité d'exécution : une ou plusieurs MIDlets peuvent être contenues dans une MIDlet Suite.
- Dépendances : une MIDlet Suite peut contenir une ou plusieurs MIDlets. Avec MIDP3, toute MIDlet peut requérir des librairies qui se trouvent à l'extérieur de la MIDlet Suite.

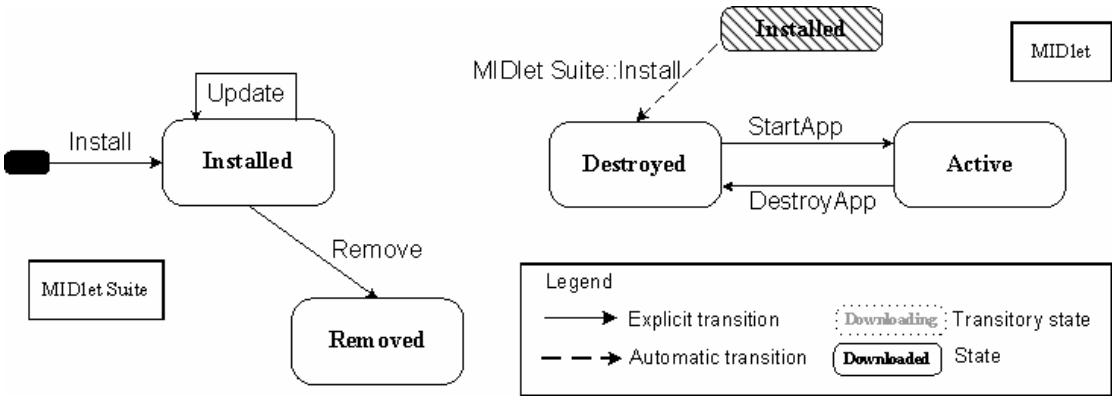


Figure 35 Cycles de vie d'une MIDlet Suite et de ses MIDlets

- Métadonnées : les métadonnées sont écrites dans le fichier JAD.
  - Des propriétés génériques optionnelles décrivant la provenance de la MIDlet Suite : MIDlet-Name, MIDlet-Version, MIDlet-Vendor, MIDlet-Jar-URL, MIDlet-Jar-Size, MIDlet-Description, MIDlet-Icon, MIDlet-Info-URL, MIDlet-Data-Size.
  - Des propriétés spécifiques à la plateforme Java MIDP utiles au déploiement de la MIDlet Suite et à l'exécution de ses MIDlets : MicroEdition-Profile, MicroEdition-Configuration, MIDlet-n, MIDlet-Install-Notify, MIDlet-Delete-Notify, MIDlet-Permissions, MIDlet-Permissions-Opt, MIDlet-Push-n, MIDlet-specific attributes, MIDlet-Jar-RSA-SHA1

- Opérations de gestion de cycle de vie (cf. Figure 35)
  - des unités de déploiement : Install, Remove, Update.
  - des unités d'exécution : StartApp, StopApp. Les 2 premières versions de MIDP ajoutaient l'opération de mise en pause (PauseApp). Cette opération est rendue obsolète dans la 3ème version.
- Etats (cf. Figure 35)
  - des unités de déploiement : Installed, Removed.
  - des unités d'exécution : Destroyed, Active. L'état "Paused" est obsolète depuis la version 3. Les MIDlets apparaissent avec l'état "Destroyed" lorsque la MIDlet Suite associée est installée (cf. partie droite de la Figure 35).
- Evénements : seul le gestionnaire d'application global, appelé JAM (Java Application Manager), est conscient des transitions d'état des MIDlets.
- Répertoires d'unités de déploiement locaux ou distants : il est possible de créer un répertoire de MIDlet Suite avec les informations des descripteurs d'applications.
- Répertoires d'unités d'exécution locaux : les plateformes MIDP offrent généralement un outil listant les MIDlets disponibles et leurs états.

3.3.3 Packages Linux

Linux est certainement l'environnement d'exécution le plus répandu sur les équipements embarqués du réseau domestique. Linux pourrait être érigé en tant que standard de facto s'il ne possédait pas tant de variantes. Les distributions Slackware, Debian et Red-Hat sont les principales distributions dont dérivent les autres avec plus ou moins de succès : Ubuntu, Mepis, Zenwalk, Mandriva, Suze, etc. La structure et le cycle de vie des entités logicielles définies par ces variantes diffèrent de l'une à l'autre. Certains travaux montrent pourtant qu'un système modulaire aux dépendances déclaratives 'à la OSGi' pourrait être aisément normalisé au-dessus d'une distribution Linux. Le prototype nommé Home Gateway Linux [Roy07] a été élaboré au-dessus d'une distribution Slackware. Les dépendances entre paquetages, unités de déploiement des distributions Linux, sont spécifiées par des dépendances vers des noms de paquetages. Ces déclarations de dépendances ne sont toutefois pas de grain aussi fin que les déclarations de paquetage de code Java entre bundles OSGi.

Malgré les variantes offertes par les distributions Linux, une carte d'identité générique peut être tentée :

- Paquetage de déploiement : aucune entité ne correspond exactement à ce concept. Le paquetage Linux peut toutefois être considéré en tant que tel puisqu'il peut être téléchargé sans être installé.
- Unité de déploiement : le paquetage ou package commun aux 3 familles Linux.
- Unité d'exécution : le script RC commun lui aussi à la plupart des distributions. Le processus init montre aussi d'autres variantes d'unités exécutables. [RoF07] énumère quelques systemes Linux importants et comparent la gestion du cycle de vie de telles unités.
- Dépendances : les dépendances entre paquetages sont explicites dans certaines distributions comme celles de la famille Debian où le gestionnaire de paquetages est capable de rapatrier toutes les dépendances d'un paquetage demandé. Entre les paquetages et les scripts RC, les métadonnées manquent afin de lier les scripts aux paquetages requis. C'est ce que palie le prototype de [Roy07].

- Métadonnées : un script RC n'est décrit par aucune métadonnée. Les packages RedHat et les packages Debian possèdent en revanche des métadonnées afin de décrire les dépendances entre eux. Voici la liste des propriétés Debian :
  - Des propriétés génériques optionnelles décrivant la provenance du package : Package, Version, Section, Installed-Size, Maintainer, Description.
  - Des propriétés spécifiques à la plateforme Debian et utiles au déploiement : Priority, Architecture, Essential, Depends, Pre-Depends, Recommends, Suggests, Conflicts, Replaces, Provides.
- Opérations de gestion de cycle de vie (cf. Figure 36)
  - des unités de déploiement : Install, Remove, Upgrade1.
  - des unités d'exécution : Start, Stop. Ce sont les opérations les plus communément supportées par les scripts RC et autres processus init. [RoF07] montre des listes d'opérations différentes disponibles sur quelques distributions en supplément de start et stop (e.g., restart, respawn, init, kill).

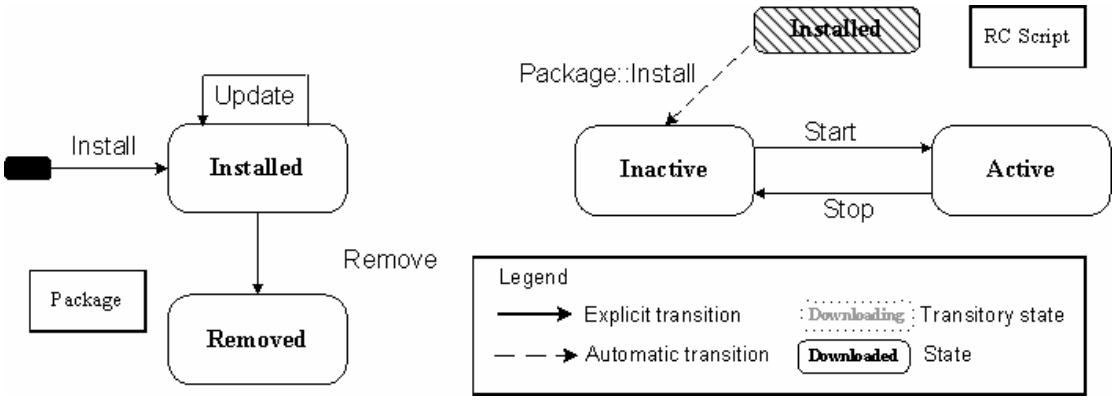


Figure 36 Vision simple des diagrammes d'états des entités logicielles Linux

- Etats (cf. Figure 36)
  - des unités de déploiement : Installed, Removed, Resolved. Comme les mécanismes de résolution de dépendances existent sur les plateformes Debian et RedHat, l'état "Resolved" pourrait être un état pour les packages de la plateforme. De nombreux états intermédiaires existent selon les distributions mais ne sont pas représentés ici.
  - des unités d'exécution : Inactive, Active. Les scripts RC peuvent être defines alors que les packages utiles ne sont pas encore disponibles. Toutefois, il ne peuvent être activés que lorsque ces packages sont présents.
- Evénements : [Roy07] utilise les signaux POSIX associé à l'arrêt, la pause et la continuation de démons init. Pour les scripts RC, il est possible de suivre l'activité de ceux-ci grâce à la table des processus.
- Répertoires d'unités de déploiement locaux ou distants : des répertoires de packages Debian, appelés Debian Repositories, existent sur Internet.

[1] Même si le manuel Debian montre que le détail des opérations est beaucoup plus compliqué à grain fin, la finalité des ensembles d'opérations est bien l'installation, la mise à jour et la désinstallation : http://www.debian.org/doc/debian-policy/ch-maintainerscripts.html#s-mscriptsinstact

- Répertoires d'unités d'exécution locaux : la liste des processus offre le moyen de suivre l'état des processus en cours d'exécution.

3.3.4 Assemblies .NET

.NET est un environnement d'exécution innovant spécifié par Microsoft. Son originalité tient principalement dans la définition d'une machine virtuelle interprétant un code binaire commun à la compilation de plusieurs langages de programmation (e.g., C#, VB#, J#). .NET est né dans les mêmes années que la plateforme OSGi. Certaines caractéristiques rendent ces deux technologies similaires : machine virtuelle, chargement dynamique de modules logiciels appelés "assembly", résolution de dépendances par la plateforme par utilisation de métadonnées. Certains travaux [EDH06] montrent les limites d'une approche OSGi sur la plateforme .NET. Le partage et l'isolation de code sont définis au niveau des types. Les niveaux de visibilité appelés "public" et "internal" distinguent les types qui sont visibles à l'extérieur de l'assembly des types qui demeurent privés au code interne de l'assembly. Le code public d'une assembly est visible par les autres assembly d'un même domaine tandis que des assemblies de Domaines différents ne peuvent communiquer que par communication inter-processus (IPC). Un assembly peut être chargé à l'exécution par un ou plusieurs Domaines. En revanche, les assemblys sont chargées séparément par les AppDomain (pas de partage d'objets) et un assembly ne peut être déchargé dans un Domaine sans décharger tout le Domaine. Ces limites placent la modularité de la plateforme .NET entre la flexibilité OSGi et le mode tout-ou-rien du modèle de bac à sable de MIDP3.

Voici les caractéristiques des entités logicielles sur la plateforme .NET :

- Paquetage de déploiement : non défini. Les installeurs d'applications Windows peuvent être considérés comme le Paquetage de déploiement .NET. Ces installeurs sont téléchargeables et permettent l'installation directe de plusieurs ressources interdépendantes. Après installation, l'installeur peut être supprimé ou gardé pour une réinstallation.
- Unité de déploiement : l'assembly est un ensemble de ressources décrit par un fichier manifest. Il peut être chargé dynamiquement par un ou plusieurs Domaines d'Applications.
- Unité d'exécution : tout assembly propose une méthode générique d'invocation avec un tableau de paramètres (équivalent à la méthode Java main()).
- Dépendances : chaque assembly déclare les types qu'il exporte et les assemblies qu'il requiert.
- Métadonnées : les propriétés suivantes sont définies par chaque assembly dans son manifeste. Les propriétés suivantes se voient attachées un préfixe "Assembly" dans la spécification. On peut distinguer :
  - Des propriétés génériques optionnelles décrivant la provenance de l'assembly : Culture, Flags, Version, Company, Copyright, FileVersion, InformationalVersion, Product, Trademark, DefaultAlias, Description, Title.
  - Des propriétés spécifiques à la plateforme .NET utiles au déploiement et à l'exécution des assemblies : Name, FileList, TypeReference, ReferencedAssemblies, EntryPoint, Configuration, DelaySign, KeyFile, KeyName.
- Opérations de gestion de cycle de vie (cf. Figure 37) : tout assembly peut être téléchargé et supprimée dans le cache local appelé "Global Assembly Cache" ou dans des caches particuliers à chaque Domaine. Puis, l'assembly peut être chargé (Assemly.Load()), invoqué (Assembly.EntryPoint.invoke(Object[]

optionalParams) ou AppDomain.ExecuteAssembly(Assembly a)). La spécification ne définit pas d'opération de résolution, les mécanismes de résolution sont donc déclenchés par la plateforme elle-même.

- Etats des unités de déploiement et des unités d'exécution (cf. Figure 37) : Downloaded, Loaded, Unloaded. La détermination de l'activité d'un assembly n'est pas spécifiée. Et bien que les mécanismes de résolution existent, ils ne sont pas utilisés avant l'exécution de l'assembly.

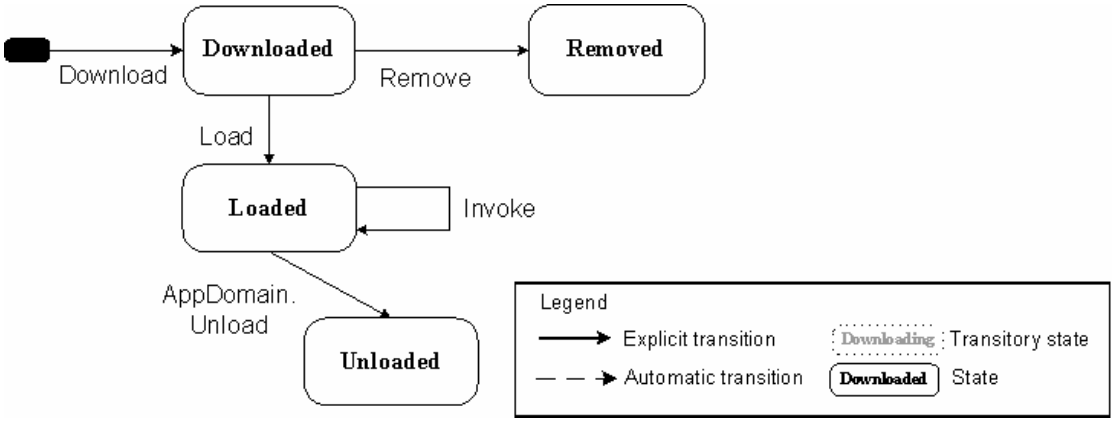


Figure 37 Diagramme d'état d'un assembly .NET

- Evénements : ils sont conséquents aux changements d'états : "AssemblyLoad", "AppDomainUnload", "AssemblyResolve" (lorsqu'un type requis n'est pas présent à l'exécution).
- Répertoires d'unités de déploiement locaux et distants : un cache global à la machine, appelé Global Assembly Cache est un répertoire où puisent les Domaines d'Applications.
- Répertoires d'unités d'exécution locaux : la plateforme .NET est livrée sur Windows avec un outil (.Net Framework Configuration) listant les assemblies installées et permettant la désinstallation de chacun avec leurs métadonnées.

3.3.5 Composants SCOMO : une tentative de généralisation

SCOMO [SCOM08] est un standard en cours de spécification par l'Open Mobile Alliance. A la différence des autres plateformes présentées, SCOMO est un protocole, qui lié à OMA DM (cf. section 3.2.2), permet de gérer le cycle de vie des entités logicielles de plateformes embarquées. Il n'est donc lié à aucun langage de programmation et suit une approche généraliste similaire à la spécification UPnP Device Management, c'est-à-dire la définition d'un protocole pouvant s'adapter au contrôle de toute technologie de plateforme logicielle.

Voici les caractéristiques des entités décrites par SCOMO :

- Paquetage de déploiement : le Delivery Package (DP). Celui-ci permet l'installation des unités de déploiement contenues. Le DP peut être retiré de la plateforme sans désinstaller les unités de déploiement.
- Unité de déploiement : le Deployment Component (DC) qui est défini en tant que représentation administrable du composant logiciel (Software Component).
- Unité d'exécution : non définie.
- Dépendances : un DP peut contenir un ou plusieurs DCs.

- Métadonnées : chacune des entités est décrite par des métadonnées qui sont répertoriées dans l'arbre de données de l'équipement. Plusieurs données sont les mêmes entre un DP et un DC :
  - Des propriétés génériques optionnelles décrivant la provenance du package : name, description, version, PkgType.
  - Des propriétés spécifiques à la gestion SCOMO et utiles au déploiement : PkgID (DP), PkgURL, ID (DC), data.

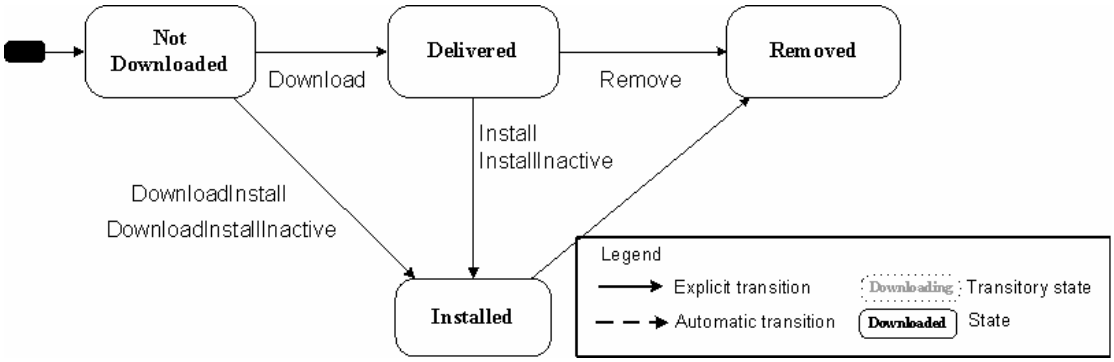


Figure 38 Diagramme d'état du Delivery Package SCOMO

- Opérations de gestion de cycle de vie :
  - des paquetages de déploiement (cf. Figure 38) : Download, DownloadInstall, DownloadInstallInactive, Install, InstallInactive, Remove. Des actions composées sont spécifiées : DownloadInstall, DownloadInstallInactive. La raison pourrait être a priori la possibilité de s'adapter au mieux aux différentes opérations spécifiées dans les différentes plateformes logicielles existantes. Elle pourrait être aussi de rendre les opérations plus rapides et assurer leur cohérence en les groupant. La citation suivante, qui indique qu'une action composée est effectuée par réalisation des deux actions correspondantes, semble en faveur de la seconde raison : "When a Composed Primitive is executed, two state transitions happen in the Device. For example if DownloadInstall is executed, a Deployment Component transits from Not Downloaded State to Delivered State after successful download procedure. It transits to Active State after successful installation procedure. If the latter processing fails it remains in previous state and the second state transition does not happen."
  - des unités de déploiement (cf. Figure 39) : Activate, Deactivate, Remove. Les unités de déploiement sont explicitement activables et dés-activables. L'état "Inactive" correspond à un état où aucune application ne peut utiliser l'unité. L'activation permet l'accessibilité des applications dépendantes à l'utilisateur. Cette activation semble correspondre à la demande de résolution des dépendances de l'unité. Les applications n'apparaissent à l'utilisateur que lorsque tous les DUs correspondants sont actifs.
- Etats :
  - des paquetages de déploiement (cf. Figure 38) : Not Downloaded, Delivered, Installed, Removed.
  - des unités de déploiement (cf. Figure 39) : Inactive, Active, Removed.
- Evénements : une alerte est définie pour chaque changement d'état.

- Répertoires de paquetages de déploiement locaux ou distants : l'inventaire des DPs dans l'état "Not Downloaded" permet de répertorier localement les unités disponibles pour le téléchargement. L'inventaire des DPs dans l'état "Delivered" est aussi maintenu et permet de lister les éléments disponibles pour l'installation.
- Répertoires d'unités de déploiement locaux : l'inventaire des DCs est aussi maintenu dans l'arbre de données de l'équipement.

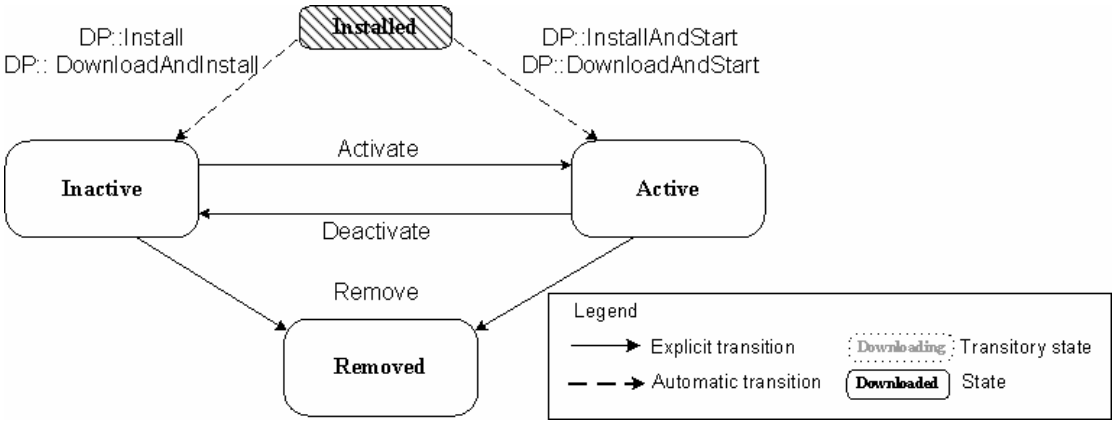


Figure 39 Diagramme d'état du Deployment Component (execution unit) SCOMO

3.3.6 Autres plateformes aux applications modulaires

D'autres plateformes d'exécution de modules logiciels existent. Les navigateurs tels que Mozilla Firefox montrent un système de plugins, les plateformes Java EE montraient jusqu'à l'adoption générale d'OSGi des systèmes modulaires variés, Qualcomm BREW est un système répandu de chargement d'applications sur les téléphones mobiles, etc.

4 SYNTHESE

Le réseau domestique est un réseau pervasif par nature. Il est ouvert à la connexion dynamique d'équipements distribués hétérogènes. Cet environnement est l'objet de nombreuses applications en gestation dans les laboratoires de recherche des fabricants et des fournisseurs de services de télécommunications. Pour ces raisons, il est le terrain choisi pour les expérimentations de cette thèse.

De nombreux protocoles de contrôle Plug-n-Play sont spécifiés pour des besoins similaires sur les réseaux locaux. Les légères différences dans leurs cibles ont menés à des différences dans leurs spécifications. UPnP semble le grand gagnant aujourd'hui sur le marché du réseau domestique, notamment dans les applications multimédia et de connectivité. Sa simplicité est sont principal atout technique. Apple Bonjour et iTunes sont des concurrents importants sur le marché du multimédia. La compatibilité avec l'infrastructure DNS est une des raisons technique de son choix, la force d'Apple est certainement plus importante. DPWS est un protocole d'avenir, la généricité des Web Services est son atout majeur, mais peut-être un point faible pour le développement sur des équipements embarqués. L'hétérogénéité est aussi très grande parmi les bus de terrain et les protocoles de la sphère personnelle.

Les protocoles de contrôle sont à même d'enrichir d'autres domaines d'applications comme la communication interpersonnelle et l'administration d'équipement. Les protocoles de communication personnelle diffèrent principalement par leur lien avec le réseau Internet. SIP montre des principes d'architecture simples et permettant une flexibilité appréciable pour

les applications de communication ambiante, applications les plus complexe de l'Informatique Pervasive.

Les protocoles d'administration d'équipements sont aussi liés à l'Internet. Visant la supervision, la configuration et la gestion du cycle de vie logiciel de milliers d'équipements, leur principale exigence technique est le passage à l'échelle. Ces protocoles spécifient généralement un ensemble minimal d'opérations pour la manipulation de données arborescentes de description d'équipements. Les protocoles TR-069 du Broadband Forum et OMA DM semble être plébiscités aujourd'hui respectivement par les marchés domestique et mobile. Si la supervision et la configuration des équipements ne présentent pas de défi majeur, la gestion du cycle de vie est affectée du problème d'hétérogénéité des plateformes d'exécution.

Enfin, surmonter l'hétérogénéité des technologies et des équipements dans ces trois domaines d'applications est un des défis présentés par plusieurs scénarios envisagés aujourd'hui. De surcroit, de nouvelles applications naissent de l'intégration de technologies distinctes afin de lier différents domaines d'application, par exemple dans des applications de communication interpersonnelle enrichie ou d'administration multi-protocolaires d'équipements. Proposer des architectures pertinentes pour des applications innovantes est l'un des axes d'investigation de ces travaux de thèse. La présentation des avancées techniques publiées par ces travaux est l'objet du Chapitre IV suivant.

Chapitre IV. PROBLEMATIQUE DU DEVELOPPEMENT D'APPLICATIONS PERVASIVES

Ce chapitre expose les défis de l'Informatique Pervasive. Il énumère ensuite la liste générale des exigences techniques construite tout au long de l'état de l'art et complète cette liste d'exigences par les contraintes associées à la programmation d'applications dans les environnements pervasifs. Enfin, les limites des réponses actuelles sont abordées et mettent en exergue les manques que ces travaux de thèse tentent de combler.

1 LES DEFIS DE L'INFORMATIQUE PERVASIVE

L'ouverture de l'environnement à des entités électroniques distinctes et variées dans un contexte évoluant rapidement est la caractéristique des environnements pervasifs. Le mémoire se concentre sur la réponse à trois défis posés par l'Informatique Pervasive dans le développement d'applications logicielles :

- Communication distante avec des entités inconnues au préalable
- Adaptation à des protocoles et des interfaces hétérogènes
- Adaptation au contexte dynamique des applications de l'Informatique Pervasive

1.1 Communication distante avec des entités inconnues

L'organisation de ces entités distinctes en un réseau cohérent pose le premier défi de la répartition des équipements. Un système réparti coordonne les fonctions de plusieurs unités de calculs disposées en réseau afin de fournir les fonctionnalités agrégées d'une même application. Des techniques intergicielles permettent de masquer l'aspect distribué de fonctions distantes dans les langages de programmation.

Masquer la communication distante à des entités programmatiques inconnues à l'avance demeure toutefois un défi de l'Informatique Pervasive. En premier lieu, il s'agit de masquer les détails de la programmation répartie qui comprend les détails protocolaires et la prise en compte d'erreurs due à la répartition. Les huit écueils de l'Informatique Distribuée (cf. Chapitre II.3.2.8) montrent que certains aspects demeureront toujours présents à la conception malgré le degré de leur transparence au cours du développement.

En second lieu, il s'agit de masquer l'aspect réparti de la découverte des entités pervasives. En effet, l'injection d'objets représentatifs d'entités distantes ne doit pas être différente de l'injection d'objets dans les applications reconfigurables localement.

1.2 Hétérogénéité

L'hétérogénéité des entités intervenant dans l'environnement est le deuxième défi auquel cette thèse apporte des réponses. Les entités d'un environnement pervasif apparaissent avec leurs protocoles de communication standards ou propriétaires, des cibles variées de domaines d'application, des langages de programmation distincts et des modèles de données sémantiquement différents (cf. Chapitre I). Face à cette variété de situations, la gestion de l'hétérogénéité pose le problème de l'intégration quantitative des éléments hétérogènes et la qualité de transparence de cette intégration pour les applications.

Des techniques de médiation technique permettent d'unifier le contrôle d'entités utilisant des éléments distincts (cf. Chapitre II.4.2). Rendre cette couche de médiation technique extensible et capable d'évoluer dynamiquement demeure toutefois un défi. Les techniques de médiation sur les mécanismes d'intégration de passerelles protocolaires et de médiateurs logiciels. Concevoir l'alliance de mécanismes de reconfiguration et de médiation passe par l'édification de modèles qui ne sont pas encore établis complètement aujourd'hui.

Par delà le développement manuel de médiateurs entre interfaces distinctes, l'automatisation de la détection de la proximité sémantique d'interfaces syntaxiquement est une autre piste de recherche dont les propositions de résolution actuelles, issues en particulier du Web Sémantique, semblent encore plus partielles.

1.3 Dynamique

La dynamique des environnements pervasifs est le troisième défi auquel sont confrontés les concepteurs d'applications. L'une des caractéristiques de l'Informatique Pervasive est l'exigence d'adaptation des applications à un contexte changeant : adapter les services offert à l'utilisateur en fonction de son activité et de sa localisation, adapter la demande de ressources selon les capacités et la disponibilité des équipements, enfin adapter la réponse du système en fonction des variables de l'espace environnant les équipements et les utilisateurs. L'adéquation entre ressources requises et fournies doit être établie et rétablie dynamiquement selon l'évolution des entités disponibles. La forte dynamique de l'environnement exacerbe la demande de flexibilité des applications et l'aspect lâche du couplage entre les entités impliquées.

Par-delà les réponses qui sont aujourd'hui offertes par de nouvelles approches logicielles comme l'orientation service, la sélection de la meilleure configuration parmi les configurations disponibles demeure un défi important pour les applications envisagées. Le système doit alors être suffisamment flexible afin de prendre en compte de nouveaux critères de sélection. L'affectation de ces critères et des éléments de contexte associés doit de surcroit être faiblement couplée avec le développement métier de l'application. Ces mécanismes de sélection impliquent des possibilités de reconfiguration dynamique des applications dont la programmation doit aussi être simple et ouverte.

De plus, cette sélection se doit d'être non ambiguë afin que l'utilisateur ne soit pas confronté à deux réponses différentes du système dans des situations identiques. L'intelligence des applications se mesure au nombre et à la qualité des réactions de l'application aux stimuli de l'environnement constitué des équipements, des utilisateurs et de paramètres externes. Les réactions peuvent amener l'utilisateur à décider de la réaction adéquate par proposition explicite de l'application ou amener l'application à implicitement réagir sur des actions indirectes de l'utilisateur et de l'environnement. Dans les deux cas,

l'utilisateur doit pouvoir comprendre aisément les causes impliquant les effets afin de pouvoir jouer lui-même explicitement avec les critères indirects. A cette fin et afin de ne pas perturber l'attendu de l'utilisateur dans ses activités, les mêmes effets doivent répondre aux mêmes causes.

2 EXIGENCES TECHNIQUES

Cette section reprend les exigences techniques énumérées dans les différentes sections de l'état de l'art concernant la gestion des aspects distribués (cf. Chapitre II.3), la gestion de l'hétérogénéité des équipements pervasifs (cf. Chapitre II.4) et la gestion de la dynamique des réseaux pervasifs (cf. Chapitre II.5).

2.1 Transparence des aspects distribués

La réalisation de scénarios de contrôle d'équipements pervasifs (cf. section Chapitre II.3.1.1) implique une réponse aux exigences techniques suivantes. Leurs exigences techniques sont reprises dans cette liste générale:

- **Transparence de la localisation pour la recherche de fonctions** : durant la phase de développement, la recherche de ressources dans l'environnement distribué doit être semblable – idéalement identique – à la recherche de ressources sur l'environnement local à l'application développée.
- **Transparence de la localisation pour l'utilisation des fonctions découvertes** : durant la phase de développement, l'appel d'objets distants doit être semblable – idéalement identique – à l'appel d'objets locaux.
- **Possibilité d'ajout de propriétés de reconfiguration, de sécurité, de fiabilité** : cette possibilité doit être prévue afin de pallier certains problèmes dus à la distribution.

Les quelques situations où la répartition s'effectue dans une phase post-développement (cf. section Chapitre II.3.1.2) impliquent une liste différente d'exigences techniques. Ces scénarios ne sont pas généraux dans le cadre de l'Informatique Pervasive et leurs exigences techniques sont donc à écarter de cette liste générale. Toutefois, la capacité de réécriture de code binaire, qui était une exigence dans la répartition d'une application dans la phase post-développement, peut être intéressante pour affirmer la transparence de la localisation pour la recherche et l'utilisation de fonctions (les deux premières exigences ci-dessus).

2.2 Adaptabilité face à l'ouverture des réseaux pervasifs

La flexibilité est l'exigence technique principale face à l'ouverture des environnements pervasifs. La flexibilité visée doit permettre aux architectures déployées d'accepter rapidement et sans heurt de nouveaux protocoles à la demande tout en gardant des applications en cours de fonctionnement. La flexibilité concerne aussi l'intégration d'équipements aux fonctions sémantiquement pertinentes découverts avec les pilotes présents. Des exigences techniques précises découlent de cette demande générale de flexibilité :

- **Couplage lâche** : les applications bâties sur l'utilisation d'équipements disponibles dans un environnement ouvert doivent être faiblement couplées avec ces équipements. De la lâcheté du couplage dépend l'évolutivité de l'application à l'exécution et au développement d'améliorations futures.
- **Interfaces uniformes** : l'hétérogénéité des protocoles doit être masquée au développeur. Celui-ci doit pouvoir aisément utiliser les équipements les plus

adaptés sans connaître profondément les interfaces réseaux de chaque technologie. Cela rejoint l'exigence technique de **séparation des préoccupations** générale : la logique de l'application doit être séparée de la gestion générique totale ou partielle de certains aspects par le système environnant – dans le cas des travaux de cette thèse, les trois aspects sont répartition, hétérogénéité, dynamique.

- **Chargement logiciel à la demande** : la gestion d'un nouveau protocole ou d'un nouvel équipement ne doit idéalement arrêter le fonctionnement d'aucune application. Cela implique des propriétés de chargement dynamique de code ou de fonctionnalités par le système. Dans un système moins performant, l'ajout de fonctionnalités au redémarrage est aussi possible et moins couteux. Et si le système ne possède pas de telles propriétés, la demande de modularité peut être comblée par l'installation de passerelles protocolaires. Toutefois, comme il est montré dans la section Chapitre II.4.2.2, les passerelles protocolaires s'avèrent couteuses ou peu pertinentes dans certaines situations.
- **Découverte protocolaire réactive** : l'entrée et le départ d'un équipement doivent être notifiés sur la plateforme de composition. Un équipement apportant des fonctions requises mais qui est accessible selon un nouveau protocole déclenche le processus d'installation du pilote pour celui-ci.
- **Négociation et adaptation de flux initiés par les applications de contrôle** : afin de faire inter-opérer les équipements du réseau domestique avec les terminaux de communication interpersonnelle, les applications doivent non seulement faire face à l'hétérogénéité des protocoles de contrôle mais aussi négocier et parfois adapter les formats et protocoles de flux multimédia selon les cas. La situation typique est celle des équipements UPnP/DLNA qui utilisent exclusivement le protocole de flux HTTP alors que les équipements SIP utilisent exclusivement RTP. UPnP/DLNA est le standard principal des équipements multimédia de la maison tandis que SIP est majoritairement accepté comme le protocole de la communication interpersonnelle sur IP.

2.3 Adaptabilité face la dynamique de l'environnement

Dans les applications attentives au contexte, les algorithmes visent le classement des fournisseurs de service selon les besoins contextuels de l'application à l'exécution. Des mécanismes de haut niveau sont requis afin de séparer la définition du comportement contextuel et la logique du cœur de l'application. La problématique est décomposable en quatre exigences techniques distinctes :

- **Qualification contextuelle des propriétés de services** : le contexte du fournisseur de service est acquis par lui-même ou de manière extérieure. Par exemple un centre multimédia pourra lister les codeurs-décodeurs (codecs) qu'il permet alors qu'un système extérieur, tel un système de gestion de localisation, seul pourra renseigner la pièce qu'il occupe. Dans les deux cas, cette information doit être ajoutée dynamiquement à la description des services fournis.
- **Qualification contextuelle des requêtes de services** : les préférences statiques et l'activité dynamique de l'utilisateur peuvent être acquises par des entrées internes au demandeur de services ou de manière extérieure. Par exemple, les paramètres entrés par l'utilisateur au travers d'interfaces graphiques peuvent être considérés comme acquis par des entrées internes si le demandeur de services délivre ces interfaces ou par des entrées externes si les interfaces graphiques appartiennent à une autre entité.

- **Classement dynamique de services** : les besoins des demandeurs de service doivent être projetés sur les différentes techniques de sélection : domaines, filtres obligatoires, filtres optionnels, algorithmes de tri afin de classer dynamiquement les services selon les besoins de l'application donnée. Un changement dans le classement propre à l'application déclenche des actions de reliaison.
- **Sujétion à des politiques de (re-)liaison** : le concepteur peut raffiner l'ensemble des actions de dé-liaison et de reliaison afin de garantir un niveau de continuité de service. Ces taches dépendent souvent spécifiquement de l'application donnée, ce qui rend difficile la tentative d'automatiser les réactions de liaison. Les actions de continuité de service sont le plus souvent l'objet d'une optimisation spécifique.

3 CONTRAINTES

Les aspects répartis, l'ouverture de l'environnement et sa dynamique ne sont pas les seules barrières à surmonter dans le développement d'applications pervasives. Ces défis s'accompagnent d'autres aspects contraignant la réponse à ces défis. Les motifs de sécurité et de préservation de l'intimité sont d'autres aspects importants dont les concepteurs doivent tenir compte. Les contraintes de l'embarqué sont les contraintes fortes qui freinent l'adoption des principes de flexibilité prônés par la littérature technique.

3.1 Sécurité et préservation de l'intimité

L'ouverture des environnements pervasifs à de nombreuses entités inconnues au préalable pose d'autres défis comme la sécurité. Si l'environnement est ouvert à toute entité inconnue, il se doit paradoxalement d'authentifier et de réglementer l'accès aux différentes parties. L'authentification et l'affectation de droits d'accès s'accompagnent du contrôle de l'intégrité des données échangées et de la vérification effective des effets escomptés.

La sécurité est aussi liée au respect de l'intimité des individus. La remontée d'information nécessaire aux applications attentives au contexte exige des protections pour les informations prises sur les utilisateurs.

3.2 Contraintes embarquées

L'aspect embarqué des équipements accroît la difficulté de la conception des applications innovantes visées. La plupart des différentes exigences techniques énumérées demandent des ressources de calcul et de mémoire et parfois des capacités spécifiques.

La répartition des applications imposent des ressources de calcul et de mémoire en raison de la sérialisation et la dé-sérialisation de la communication de l'application avec les équipements distribués. La communication transparente dans le langage objet avec des équipements aux protocoles humainement lisibles augmente les taches de calcul nécessaires.

L'adaptabilité face à l'ouverture de l'environnement exige de la plateforme d'exécution des capacités de reconfiguration applicatives. Le chargement logiciel à la demande est une exigence technique due à l'ouverture de l'environnement. Le déchargement logiciel est demandé en retour afin de libérer des ressources en cas d'installations prioritaires.

Aussi, le couplage lâche et le passage par des adaptateurs en langage objet, induits par le besoin d'adaptabilité face à l'ouverture de l'environnement, alourdissent la transmission d'appels des applications clientes à la destination des équipements. Ces mécanismes d'indirection introduisent inévitablement des pertes de performance [2008-2][2008-5].

L'adaptabilité à la dynamique du contexte de l'environnement pervasif entraine l'implantation de mécanismes de contextualisation, des algorithmes de sélection et des mécanismes de reconfiguration qui pèsent sur le fonctionnement des applications. La contextualisation demande la connexion à un gestionnaire de contexte (ou la gestion du contexte elle-même) et le stockage de descriptions riches d'informations de contexte. Les algorithmes de sélection varient du simple filtre de données à la résolution de problèmes d'optimisation de plusieurs équations. Les mécanismes de reconfiguration ont pour conséquence des séries d'appels de méthodes afin d'établir les liaisons les plus adéquates après un changement de situation. Pire, les mécanismes de continuité de service impliquent souvent une redondance du service à l'utilisateur durant un certain laps de temps [2007-3]. Tous ces mécanismes ont un impact sur les ressources de calculs et l'empreinte mémoire prises par l'application [2007-7].

La prise en compte de toute autre qualité logicielle comme la précision de la QoS dans le contexte applicatif, la sécurité ou la préservation de l'intimité (cf. section 3.1) engendre chaque fois l'ajout de code et de données dans l'application. Par évidence, ces ajouts ont un coût en terme de ressources de calcul et de mémoire.

Par conséquent, catégoriser les exigences techniques de flexibilité et mesurer leur impact sur le coût des applications est un devoir pour les chercheurs et architectes du Génie Logiciel. Face au prix du nombre des équipements à répartir dans l'environnement, les concepteurs d'applications tentent un compromis entre des plateformes peu couteuses dont les contraintes interdisent l'évolution de l'application originale et des plateformes plus riches qui s'accordent avec l'extensibilité demandée à moyen terme par l'utilisation. La conception des interfaces intergicielles doit toujours être accompagnée d'une implémentation démontrant que le surcout ajouté est négligeable face aux atouts d'un intergiciel [Kra03]. Certaines contributions de cette thèse – [2007-7][2008-2][2008-5] – sont évaluées sur des critères de performances liés au caractère embarqué des applications visées. Le Chapitre VII apporte des réponses quantitatives et qualitatives concernant l'impact des techniques et de leur implémentation proposée dans ce mémoire.

4 LIMITES DES REPONSES ACTUELLES

4.1 Le marché des réseaux locaux aujourd'hui

Sur le marché des réseaux locaux, en particulier celui du réseau domestique, les solutions actuellement installées sur les équipements sont rarement dédiées à plusieurs applications. Lorsqu'elles le sont, par exemples les applications installées sur les smartphones, chaque application est isolée et n'utilise le plus souvent qu'un seul protocole de contrôle (par exemple, UPnP dans les applications de navigation dans le contenu de serveurs domestiques et envoi sur la télévision). Pourtant, plusieurs scénarios innovants sont possibles aujourd'hui par utilisation de plusieurs protocoles différents et rapprochement de domaines d'applications distincts (cf. Chapitre II.4.1). Connecter les ilots de découverte ("Connecting Islands of Discoverability") [Kei06] est bien un des défis nouveau de l'Informatique Pervasive tiré par l'évolution du marché d'aujourd'hui.

Un des leviers pour la connexion des ilots de découverte est l'utilisation de plateforme de déploiement modulaire d'applications permettant le partage de ressources de code entre modules applicatifs substituables. Toutefois, ce levier ne semble aujourd'hui réservé qu'aux logiciels installés sur les serveurs d'exploitation des PCs, des PDAs et serveurs d'applications. Quelques exceptions peuvent être citées : la passerelle véhiculaire des voitures de marque BMW et les imprimantes Ricoh et Canon sur réseau d'entreprise qui hébergent une plateforme OSGi, le serveur de contenu domestique appelé "Home Server" de Microsoft commercialisé en 2008 qui héberge une plateforme .NET. Au contraire, la plupart des

équipements sur les réseaux locaux domestiques, mobiles, véhiculaires, SOHO[1] et industriels sont développés dans un langage natif (langage C habituellement) souvent dépourvu de structuration objet, d'approche composant et loin de l'orientation service.

De surcroit, l'ouverture du réseau domestique appelle les équipements à se mettre à jour régulièrement, en particulier les plateformes de services à adapter leurs capacités protocolaires à l'offre de services sur le réseau domestique. Il est notable aujourd'hui que seuls de rares équipements connectés peuvent être mis à jour. Et ceux-ci ne proposent généralement qu'une mise à jour globale du micrologiciel (firmware). La mise à jour partielle repose premièrement sur la modularité des couches logicielles et deuxièmement sur les capacités de chargement de code à la demande. Il apparaît que les téléphones mobiles permettent le chargement d'applications à la demande mais les technologies utilisées isolent ces applications des autres applications installées. Pour des réseaux de sécurité et d'une spécification technique peu flexible, les applications mobiles vont demeurer simples encore pour quelques années. Sur les équipements domestiques comme la passerelle Internet, de nombreux prototypes donnent l'espoir de voir se commercialiser dans les années à venir les prémices de plateformes embarquées de téléchargement d'applications modulaires : le "Home Server" de Microsoft, les prototypes OSGi (www.osgi.org/Markets/SmartHome), etc.

Enfin, les applications conscientes du contexte n'en sont encore qu'à de vagues balbutiements. La plupart des applications du réseau domestique ne sont que des actionneurs rudimentaires couplés à des capteurs simples, par exemple l'ouverture de robinets d'eau, l'ouverture de portes ou les alertes de sécurité sur détection de présence infrarouge, ou encore l'éclairage indexé sur la luminosité extérieure. Les systèmes de localisation sur réseaux locaux, les systèmes de détection de postures, de visage, d'activités ne sont encore que des prototypes. Les logiciels s'adaptant aux préférences et à l'environnement de l'utilisateur sont encore peu nombreux. Le champ des applications conscientes du contexte peine à décoller en raison du cout d'installation de capteurs connectés en réseau, de la non-maturité des protocoles et de leurs implémentations, de la complexité des modèles d'inférences pertinents, de la complexité de liaison d'un tel système à une application proche de l'utilisateur.

4.2 Limites des travaux de recherche actuels

4.2.1 Transparence des aspects distribués

L'état de l'art de la gestion des aspects distribués (cf. Chapitre II.3) montre un domaine mature qui bénéficie de travaux nombreux et depuis des décennies. Toutefois, le courtage dynamique de l'orientation service et l'avènement de plateformes d'applications modulaires remettent à jour l'affinement de patrons de conception dans ce domaine.

La transparence d'appels répartis de procédures et de méthodes est établie depuis quelques années dans des patrons de conception et appliquée dans des langages de programmation divers. Des travaux de recherche montrent même la possibilité de leur application par réingénierie de code binaire dans une phase post-développement après analyse automatique de délimitation de modules à répartir. La complexité de ces systèmes et l'aspect discutable des exemples d'application fournis expliquent que ces approches demeurent marginales.

Le courtage dynamique des applications orientées service pose à nouveau le problème de la transparence de la recherche d'objets distribués. Idéalement, la recherche

[1] Small Office Home Office: qualifie les équipements et les réseaux pour les activités de "bureau" tant en entreprise qu'à la maison.

d'objets et la liaison de ceux-ci se doivent de s'appliquer de manière unique et indiscernable sur un ensemble d'objets locaux et répartis. Cette propriété n'est toutefois pas encore répandue sur les plateformes orientées services. Elle le devient progressivement dans les architectures proposées par plusieurs projets contemporains comme le projet Extended Service Binder de cette thèse [2005-1][2006-2][2006-4][2006-5][2006-6][2008-3], le projet R-OSGi [RAR07], et les travaux de l'OSGi Enterprise Expert Group pour la version 5 non encore publiée de la spécification OSGi.

4.2.2 Adaptabilité face à l'ouverture et la dynamique des réseaux

Les applications sur le réseau domestique sont en plein essor [Aie06]. Toutefois, le contrôle d'équipement, la communication interpersonnelle, l'administration d'équipements sont des domaines où les nouvelles applications manquaient à être affirmées. De surcroit, la littérature manque encore de cadriciels cohérents et répandus.

Si les protocoles de contrôle Plug-n-Play (cf. Chapitre III.1) et les plateformes de déploiement modulaires se normalisent et se répandent depuis l'année 1999 – cette année marque la fondation d'UPnP, d'OSGi, de .NET, ZigBee, DPWS et d'autres spécifications apparaissent plus tardivement – et commencent leur commercialisation, les applications mêlant ces nouveaux protocoles dans une même application apparaissent de manière contemporaine à ces travaux de thèse. Les projets de communication interpersonnelle mêlant SIP, UPnP et d'autres protocoles sont nouveaux dans la littérature, l'essai le plus approchant de la contribution [2008-6] étant [CCV06]. La création du Comité de Travail UPnP Device Management (ex-Execution Platform) par quelques partenaires issus de la communauté OSGi et l'auteur [2007-9] montrent aussi l'aspect contemporain des applications d'administration visées. Le travail de spécification parait même en avance de phase sur l'aspect de la généralisation de modèles d'unités logicielles génériques et de leurs cycles de vie. Les seuls travaux s'en approchant semblent être ceux du groupe de travail OMA SCOMO [SCOM08].

Avec l'avènement des applications à composants orientés service, de nouvelles technologies et de nouveaux protocoles, les cadriciels commencent à apparaître avec les patrons de conception associés au début des travaux de cette thèse : Base Drivers pour le contrôle d'équipements [DFKM02], courtage de pilotes à la demande dans l'OSGi Device Access Service de mai 2000, automatisation de la composition locale de service par conteneur [CeH03], Whiteboard pattern sur la plateforme OSGi [KrH04], publish-subscribe orienté service du service OSGi Event Admin [OSGi05], utilisation des erreurs de chargement modulaire pour les erreurs de distribution [RAR07]. Les contributions de cette thèse concernant l'utilisation de la technique des base drivers pour le protocole DPWS [2007-5][2007-6][2008-2], le raffinement des pilotes pour de la médiation technique [2006-3][2007-2][2007-4] et la population du registre de services par des informations de contexte [2007-3] accompagnent cet effort de réflexion sur les nouvelles techniques de composition de services sur les plateformes de déploiement modulaire telles que la plateforme OSGi. Un cadriciel complet est décrit par les publications complémentaires [2007-3][2007-4][2008-2][2008-5].

Enfin, certaines pistes de travail demeurent encore à explorer. L'automatisation de la détection d'interfaces sémantiquement proches mais syntaxiquement distinctes et l'automatisation de leur rapprochement syntaxique sont des objectifs qui semblent inatteignables dans des applications importantes aujourd'hui. La construction automatique de pilotes raffinés sémantiques proposée dans [2007-4] ne fait qu'effleurer le champ d'investigation.

5 CE QUE PROPOSE CETTE THESE

Cette thèse apporte des contributions selon deux grandes lignes directrices où des manques ont été détectés dans l'état de l'art. Ces lignes directrices seront détaillées dans le chapitre suivant :

- La première est la proposition d'un cadriciel aux patrons de conception cohérents pour le développement d'applications réparties dans un environnement ouvert au contexte dynamique. Ce cadriciel, appelé Home SOA, reprend les patrons de conception pertinents dans la littérature et propose des avancées originales. Chacune des sections du Chapitre V décrit des techniques de conception. Ces techniques sont implémentées dans différentes applications de contrôle d'équipements hétérogènes, de communication interpersonnelle riche, et d'administration d'équipements (cf. Chapitre VI). Elles sont validées par les tests et les comparaisons du Chapitre VII.
- La seconde est une ligne de conduite face aux problèmes d'hétérogénéité de protocoles, de plateformes d'exécution et d'interfaces programmatiques (cf. Chapitre V.1). Grace à cette ligne de conduite claire, la proposition IGRS a pu être rectifiée avant sa normalisation en Comité ISO et la proposition déplacée de Samsung d'une interface de contrôle UPnP pour la plateforme OSGi est devenue un projet d'ambition plus générale au sein du Forum UPnP. Une contribution majeure de cette thèse est la réponse protocolaire poussée dans le Comité de Travail UPnP Device Management pour l'administration du cycle de vie logiciel de plateformes d'exécution hétérogènes. Un modèle général de plateforme d'exécution est proposé afin d'administrer des plateformes variées avec le même protocole. La proposition de ce modèle est accompagnée d'une analyse des plateformes existantes (cf. Chapitre III.3.3) et d'implémentations sur trois d'entre elles (cf. Chapitre VI.7).

Chapitre V. HOME SOA : TECHNIQUES DE COMPOSITION DE SERVICES PERVASIFS

Cette thèse propose une architecture pour la composition contextuelle de services dans les réseaux d'équipements pervasifs. Cette architecture, nommée Home SOA, regroupe un ensemble de patrons de conception cohérents issus des paradigmes composants et services. Cet ensemble est implémenté dans un cadriciel éponyme concret décrit dans le Chapitre VI et est validé par des tests et la comparaison aux travaux concurrents dans le Chapitre VII.

L'approche logicielle Home SOA est pertinente dans des situations précises. Ce chapitre introduit d'abord une ligne de conduite à tenir dans les situations de réseaux hétérogènes avant de décrire les techniques du Home SOA. Cette ligne de conduite catégorise les situations où apporter des situations protocolaires et les situations où l'approche logicielle Home SOA est pertinente. La contribution majeure au Comité de Travail UPnP Device Management est un exemple d'approche protocolaire dans une situation où les protocoles standards manquent.

L'approche logicielle Home SOA se base sur des travaux antécédents liant approches composants et orientation service. Elle est l'application d'une vision plateforme-centrique des réseaux pervasifs, en particulier du réseau domestique, et de patrons de conception issus des paradigmes composant et service. Cette vision ramène toutes les applications de contrôle d'équipements pervasifs à un problème conception localisée sur une plateforme unique de l'environnement. Les techniques du Home SOA répondent à trois des grands défis de l'Informatique Pervasive : la distribution, l'hétérogénéité et la dynamique des équipements dans les réseaux locaux.

La section 1 présente la vision plateforme-centrique de la composition d'équipements pervasifs dans les réseaux locaux. La section 2 définit le concept de plateforme de services et les patrons de conception associés. La section 3 décrit alors le patron de conception "registre de services mandataires" simplifiant la gestion des aspects distribués. La section 4 explique le raffinement du patron "Adapteur" employé à masquer l'hétérogénéité des entités pervasives. La section 5 achève la présentation de ses techniques avec les patrons aidant à l'automatisation de la sélection de services. Les chapitres 6 et 7 apportent des solutions protocolaires à des situations où l'approche logicielle Home SOA est impuissante.

1 UNE ARCHITECTURE PLATEFORME-CENTRIQUE

Tous les protocoles et interfaces programmatiques qui coexistent sur un même réseau peuvent inter-opérer grâce à des techniques de médiation. Les techniques appliquées dans cette thèse obéissent à cette réponse graduelle :

- Séparer les ressources empruntées par les différents protocoles s'ils ne peuvent coexister sur ces mêmes ressources,
- Appliquer des techniques de passerelles si les protocoles sont utilisés sur des réseaux aux topologies différentes (terrain, LAN, WAN) et appliquer des techniques d'adaptation logicielles pour lier des protocoles distincts sur ces passerelles,
- Appliquer une approche protocolaire pour lier des plateformes hétérogènes sans protocole commun,
- Appliquer des techniques d'adaptation logicielles pour simplifier la gestion de la distribution, de la dynamique et de l'hétérogénéité sur une plateforme intelligente au cœur de réseaux pervasifs à la topologie homogène. Cette plateforme intelligente héberge les applications innovantes du réseau visé et porte l'attention principale des techniques du Home SOA.

1.1 Coexistence et interopérabilité des protocoles

L'incompatibilité de deux protocoles peut être de deux ordres distincts :

- Non-coexistence : les protocoles qui ne peuvent pas coexister sur les mêmes couches physiques sans perturbations excluent les situations de coexistence.
- Non-interopérabilité : des protocoles peuvent coexister sans pouvoir inter-opérer. Ceux-ci sont l'objet des techniques de passerelles et d'adaptation protocolaires et logicielles (cf. Chapitre II.4.2).

Les scénarios mêlant plusieurs ensembles protocolaires distincts reposent sur l'hypothèse que ces ensembles puissent coexister sur un même réseau. Cette hypothèse n'est pas naturelle a priori. Elle s'avère fausse dans certaines situations où des ressources physiques ne peuvent être partagées par des ensembles distincts. Les technologies sur courant porteur HomePlug et DS2 sont un exemple : ces technologies sont incompatibles et empruntent le même lien physique sans pouvoir coexister, le réseau électrique domestique. Les utilisateurs de courants porteurs doivent donc effectuer un choix entre l'une ou l'autre des deux normes. De même les standards protocolaires radios tentent de se partager des bandes de fréquences étroites, la ressource étant rare.

De nombreux protocoles s'affranchissent de l'association à un lien physique par le recours à une abstraction topologique. Le réseau IP (couche de niveau 3 dans le modèle de référence OSI, couche Internet de la pile TCP/IP) est un exemple qui délivre cette abstraction à certains protocoles Plug-n-Play du réseau domestique (cf. Chapitre III.1.1). Même au-dessus d'une telle abstraction, certaines ressources demeurent à partager afin d'éviter toute interférence nuisible. C'est le cas des adresses multicast et des numéros de port des équipements IP. Deux protocoles peuvent interférer par emprunt d'une même adresse multicast sans pouvoir inter-opérer. C'était le cas du protocole de découverte d'IGRS et celui d'UPnP au moment de la proposition d'IGRS en tant que norme ISO en 2006 (cf. Chapitre VI.6). La contribution en Comité ISO [2006-7] a principalement proposé le changement de l'adresse multicast empruntée par les équipements IGRS.

Enfin, tous les protocoles qui coexistent sur un même réseau peuvent inter-opérer grâce à des techniques de médiation.

1.2 Une vision plateforme centrique

Le problème de médiation d'entités hétérogènes peut toujours être ramené à la gestion de plusieurs protocoles dans un réseau à la topologie homogène. De surcroit, comme la section Chapitre II.3.2 le montre, les techniques de communication répartie permettent de penser la composition de services distribuées comme une composition locale de services sur une seule et même plateforme. Dans le cadre d'une topologie homogène, la gestion de l'hétérogénéité des protocoles est efficacement traitée par des techniques de médiation programmatiques locales (cf. Chapitre II.4.2). C'est pourquoi une vision plateforme-centrique émerge dans plusieurs travaux de la littérature et dans cette thèse [2007-4].

Dans cette vision, les réseaux pervasifs comme le réseau domestique sont constitués de multiples équipements. Ces équipements sont considérés en tant que fournisseurs et consommateurs de services. Ces réseaux montrent une asymétrie entre la multitude des équipements contraints et les quelques riches plateformes où l'intégration prend place. Les plateformes du réseau domestique peuvent être hébergées par la passerelle Internet, la Set-Top-Box, le centre multimédia, ou la passerelle domotique du tableau électrique. Les autres équipements sont des éléments pervasifs intégrés dans l'environnement. Cet ensemble inclut les capteurs, les périphériques d'entrée et de sortie de commandes, les écrans, les haut-parleurs, les lumières, le système de chauffage, de ventilation et d'air conditionné. La Figure 40 illustre cette vision par le dessin de deux plateformes composant des équipements divers. Au passage, si la vision décrite est plateforme-centrique, elle permet toutefois la communication entre plateformes riches.

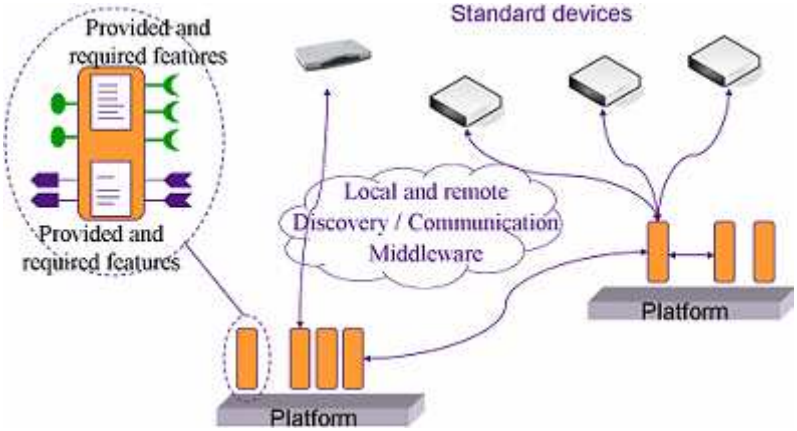


Figure 40 Une vision plateforme-centrique des réseaux pervasifs

2 PLATEFORME DE SERVICES ET PATRONS ASSOCIES

Trois entités logicielles sont considérées dans l'architecture du Home SOA :

- **le Service** est un ensemble d'opérations. Les services sont déclarés fournis et requis par les composants.
- **le Composant** est une unité logique et cohérente de code. Les composants sont auto-décrits. Leur besoin et leur offre de services sont écrits dans des fichiers spécifiés par un modèle à composants.
- **L'Application** est définie par un ensemble de composants qui interagissent afin de fournir un service intéressant pour l'utilisateur.

Les paradigmes composant et service sont complémentaires, et sont ici mariés dans la notion de "composant à services" (cf. section 2.1). Cette vision partage la vision du projet Gravity [CeH04], un des germes principaux de cette thèse. Les services considérés ici sont

des services procurés par des composants locaux ou sont issus de la réification d'entités distribuées sur le réseau. Dans le Home SOA, des pilotes orientés services sont responsables de la réification des services fournis par les différents équipements de l'environnement (cf. section 3). Tous ces services participent à un système de découverte dynamique local à une plateforme appelée "plateforme de services" (cf. section 2.2). Au centre de cette architecture, le registre de service local est la pièce maitresse des techniques et patrons de conception simplifiant la programmation sur cette plateforme : composition automatique de composants à services (cf. section 2.3), Whiteboard pattern (cf. section 2.4), publish-subscribe orienté service (cf. section 2.4), pilotes raffinés pour l'application de technique de médiation (cf. section 4), contextualisation de la composition de services (cf. section 5).

2.1 Composants à services

Une approche composant est mariée au paradigme d'orientation service afin d'offrir un environnement de développement et de déploiement à la fois structurant et flexible [2008-2][2008-5].

2.1.1 L'orientation service dans les réseaux pervasifs

La programmation orientée service consiste à décomposer les applications en fonctionnalités fournies et requises par des entités logicielles cohérentes et souvent distribuées. Chaque fonctionnalité est un ensemble d'opérations invocables qui est appelé "service". La description (abstraite) de cet ensemble d'opérations partagé entre clients et serveurs est appelée "interface de service". Les entités logicielles appelées "fournisseurs" d'un service implémentent l'interface du service et peuvent être utilisée par un client. Les clients de ce service sont appelés "demandeurs" de ce service. Le schéma d'interaction associé au paradigme d'orientation service est présenté dans la section Chapitre II.1.3 et illustré par la Figure 1.

Les caractéristiques de l'orientation service (cf. Chapitre II.1.3) s'appliquent aux réseaux d'équipements pervasifs :

- **Abstraction** : les fonctionnalités offertes par les équipements d'un réseau pervasif sont considérées en tant que services. Chaque équipement domestique peut alors être représenté par une Boite Noire.
- **Dynamique** : le courtage protocolaire et la liaison tardive des services aux applications permettent une composition réactive aux événements du réseau pervasif visé.
- **Administration partagée** : le cycle de vie des différents équipements et autres entités pervasives obéissent à différentes règles (notamment les mesures de sauvegarde d'énergie), différents administrateurs (fournisseurs de service) et différents utilisateurs.

L'Orientation Service apporte l'abstraction nécessaire à la modélisation des fonctionnalités offertes par les entités hétérogènes et la flexibilité indispensable à la résilience des applications face à la dynamique des réseaux pervasifs comme le réseau domestique.

2.1.2 Les composants dans les applications orientées service

L'orientation composant apporte un mode de programmation complémentaire à l'orientation service [CeH03]. Les deux modèles promeuvent une séparation claire entre conception d'interfaces et développement d'applications. La séparation entre une interface et les implémentations de celle-ci est le premier principe sur le chemin de l'évolutivité du code. Les interfaces fournies par les composants peuvent être simplement considérés comme

services fournis dans une Architecture Orientée Service. L'encapsulation des fournisseurs et demandeurs de services dans des unités logiques cohérentes est une approche qui favorise la réutilisabilité du code. Cette approche de composants orientés service est prise par OSGi, non seulement dans sa spécification principale [OSGi05] mais particulièrement dans le chapitre Declarative Services (cf. Chapitre VI.1.2.1), et par la spécification SCA [Cur07] (Service Component Architecture), un ensemble de spécifications issu de la collaboration industrielle nommée Open SOA (www.osoa.org).

L'orientation service modélise les applications dans une composition à haut niveau alors que la programmation orientée composant permet de structurer le code applicatif dans un modèle de développement homogène et simplificateur. Les avantages saillants des approches à composant sont :

- **Structuration** : chaque partie du code implémenté est décomposé en un ensemble d'unités de code logiques et cohérentes. La structuration en composants apporte généralement une encapsulation à plus gros grain que celle de l'approche objet, une approche complémentaire.
- **Séparation de la logique métier des aspects non-fonctionnels** : ces derniers – ici la gestion de la distribution, la gestion de la dynamique et la gestion de l'hétérogénéité – sont pris en compte par des conteneurs de manière transparente pour le développeur du "métier". Un conteneur est une partie applicative commune d'un modèle à composants qui est responsable d'un aspect non-fonctionnel pour tous les composants.
- **Inversion de contrôle** : les opérations sur le cycle de vie d'un composant sont invoquées à l'extérieur de celui-ci. Le contrôle du cycle de vie attaché à un aspect non-fonctionnel est la responsabilité d'un conteneur. Par exemple, le cycle de vie des composants du modèle OSGi Declarative Services est géré par le Service Component Runtime (cf. Chapitre VI.1.2.1).

Les approches composant apportent la modularité applicative à toute entité de l'application implémentée et simplifie la gestion d'aspects non-fonctionnels communs aux composants. La composition de l'application peut alors être délivrée à un administrateur (ou une application externe). Celui-ci devient responsable de la composition des composants pertinents et de leur configuration. L'adaptabilité et l'automatisation de la composition rendues possibles par une approche composant sont une des bases de la réponse aux besoins de flexibilité et substituabilité demandées respectivement par les contraintes d'hétérogénéité (cf. Chapitre II.4.3.1) et de sélection dynamique de services (cf. Chapitre II.5.4).

2.1.3 Composants à service

L'approche de composants à service de cette thèse suit la vision de [CCCS07] où les approches composant sont utilisées dans un mode d'intégration pré-facto où tous les composants doivent obéir à un modèle prédéfini. L'orientation service est, elle, utilisée dans une phase d'intégration post-facto où certaines parties de l'application doivent être intégrées sans introspection possible et avec le respect de leur comportements respectifs. Les composants et les services sont définis au niveau programmatique et au niveau réseau. Les services programmatiques peuvent être rendus accessibles au niveau réseau.

Un modèle à composant est premièrement utilisé pour modéliser les composants fonctionnels de l'application. Les besoins non-fonctionnels de composition externe et de reconfiguration dynamique sont gérés par un conteneur externe. Celui-ci automatise les liaisons et dé-liaisons de services selon la description disponible de chaque composant. Les services requis sont symbolisés par les connexions en demi-lune et les services fournis par les connexions en rond (cf. Figure 40). Les critères de sélection de services sont partie intégrante de cette description (cf. section 5.3).

La séparation des interfaces et des implémentations se retrouve dans la structuration en unités de déploiements distinctes : les unités de déploiement contenant les interfaces de services seront déployées dans des unités distinctes de celles des composants fournissant des implémentations. Le modèle d'unités de déploiement définis par les plateformes d'applications modulaires peut être considéré comme un autre modèle à composant. Les unités de déploiement hébergent les composants fonctionnels. Les ressources requises sont symbolisées par des queues de flèches tandis que les ressources fournies sont symbolisées par des têtes de flèches (cf. Figure 40). Le gestionnaire des dépendances de ressources entre ces unités est le conteneur associé à ce modèle (cf. Chapitre III.3.3). La structuration de ces composants augmente le couplage lâche des clients et serveurs.

La transparence des **aspects distribués** profite de cette séparation et de la définition de service au niveau programmatique. Les services locaux et distants se retrouveront accessibles selon les mêmes interfaces de service programmatiques. Un service pourra être implémenté par un mandataire ou par une implémentation locale complète (cf. section 3).

L'**hétérogénéité** des entités pervasives profite aussi de cette séparation. Tous les équipements d'un même protocole sont réifiés sous la même interface. L'hétérogénéité des protocoles est réifiée dans l'hétérogénéité des interfaces de services d'un premier niveau. La transparence est apportée ensuite par la médiation de ces interfaces vers une interface unifiée pour chaque domaine d'application. L'hétérogénéité des protocoles se retrouve figée dans les implémentations multiples d'une interface unique de 2ème niveau pour une application donnée tandis que la diversité des domaines d'applications se retrouve alors dans la diversité de ces interfaces de 2ème niveau (cf. section 4).

La **dynamique** des entités pervasives est représentée par la dynamique de l'enregistrement des services au niveau de la plateforme locale. La vision plateforme-centrique de composition de service s'allie à la définition d'une plateforme de services réifiant toutes les entités extérieures en tant que services locaux. La reconfiguration au niveau composant sera augmentée par les possibilités de courtage dynamique de services (cf. section 2.2) et de contextualisation des services (cf. section 5).

2.2 Plateforme de services

Le concept de plateforme de services repose sur le patron de conception "Service Locator" [Fow04] allié à un système de notification dans une programmation orientée objet [2008-5]. Ce concept, ou ce patron de conception (dérivé), reproduit les acteurs et les mécanismes du schéma principal de l'Architecture Orientée Service au niveau du langage de programmation :

- Le **registre de services** (ServiceRegistry cf. Figure 41) est une table énumérant les "références de services" disponibles et les "objets de services" associés. Un service d'événements notifie les souscripteurs intéressés des événements de service : enregistrement, mise à jour, suppression.
- Les **fournisseurs de services** (ServiceProvider cf. Figure 41) sont des objets qui enregistrent ou retirent des références de services auprès du registre de services. Les références sont associées à un objet de service indiqué par le fournisseur. L'objet de service implémente la (ou les) interface(s) de services sous lesquelles il est enregistré. Afin que seul le fournisseur d'un service donné puisse retirer son service, un objet représentant son enregistrement est donné exclusivement par le registre au fournisseur à l'appel de la méthode d'enregistrement. Seul cet objet permet la suppression du service dans le registre (méthode " unregister()").
- Les **demandeurs de services** (ServiceRequester cf. Figure 41) sont des objets qui requièrent des services de manière active ou passive auprès du registre de

services. Les requêtes actives et la souscription d'événements sont des appels de méthodes sur une interface programmatique définie par la plateforme. Ces méthodes permettent le passage de filtre de services en arguments. La notification suit le patron de conception "Observateur". Après l'obtention d'une référence de service, ils peuvent ensuite appeler les méthodes du service sur l'objet de service obtenu. Ces appels de méthodes s'effectuent directement dans le langage de programmation.

Chaque représentation objet enregistrée dans le registre est appelée service pour les mêmes raisons, d'abstraction et de couplage lâche, recherchées par les architectures orientées services (SOA).

Le couplage est lâche car :

- Demandeurs et fournisseurs peuvent être développés indépendamment par la seule connaissance de ces interfaces. Ils ne se connaissement pas a priori.
- Les demandeurs se lient aux fournisseurs au cours de l'exécution au moment où le demandeur en éprouve le besoin et que les fournisseurs adéquats sont présents. Le registre de service permet le courtage des fournisseurs selon des filtres de service, soit des ensembles de propriétés requises par le demandeur.
- les cycles de vie des entités mises en relation sont indépendants : l'équipement contrôlé peut être allumé, éteint, modifié indépendamment de l'utilisation de l'application sur la plateforme distante.

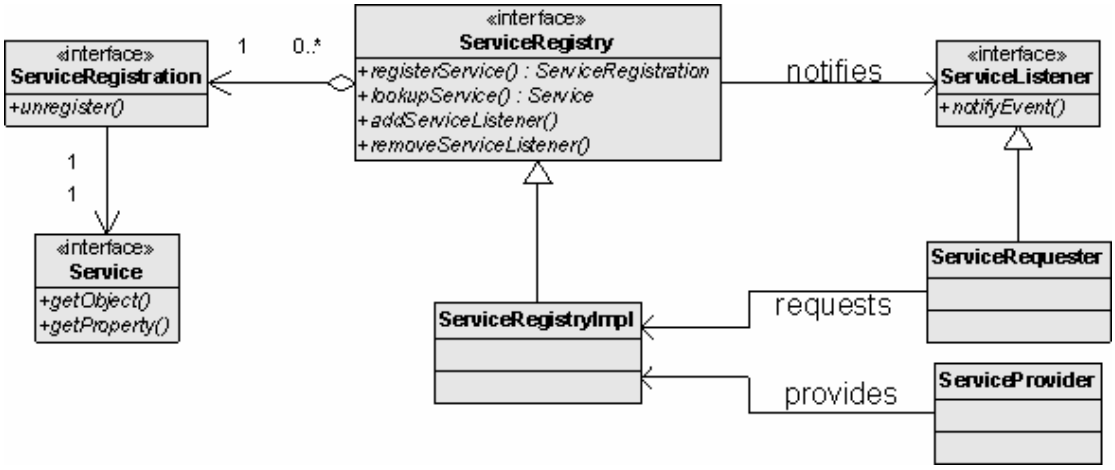


Figure 41 Patron de conception "Plateforme de service"

Le registre de service est le réceptacle dynamique de toutes les entités pervasives de l'environnement. Son rôle est de centraliser la représentation de toutes les entités pervasives pour tous les objets de la plateforme. Chaque objet est représenté avec la description la plus riche possible afin de permettre la composition la plus adéquate à tout instant. Cette connaissance associée à l'auto-description des composants permet d'automatiser la composition de services des applications (cf. section 2.3).

Le registre de service est la pièce maitresse de l'architecture logicielle promue dans ces travaux. Il permet le découplage :

- **des clients de services et des fournisseurs locaux et distants**. Les objets enregistrés obéissent à une interface (ensemble contractuel de méthodes) préalablement définie. C'est la seule connaissance partagée entre le composant demandeur de service et le composant fournisseur de service. Dans le cas distribué, le fournisseur de services est un pilote protocolaire responsable de

l'enregistrement des objets mandataires.La découverte des services distants est indiscernable de la découverte de services locaux (cf. section 3).

- **des clients de services et des médiateurs**. Les clients de services ne connaissent précisément que l'interface de service qu'ils requièrent et ne connaissent aucun détail sur les implémentations de cette interface avant exécution (cf. section 4).
- **des observateurs et des ressources observées** dans les patrons de conceptions "Tableau Blanc" (Whiteboard pattern) et "publish-subscribe orienté services" (cf. section 2.4).
- **des composants à services et des sources de contexte**. La contextualisation des services est effectuée par les sources de contexte directement sur le registre de services au travers d'interfaces de configuration définies de manière générique sur la plateforme (cf. section 5).

2.3 Composition automatique de services

Le Home SOA vise à permettre la composition de services sans connaissance précise préalable de ceux-ci dans des environnements ouverts. Cette composition nécessite l'accessibilité des descriptions riches des différentes entités pervasives au moment où elles apparaissent dans l'environnement. Les informations du contexte sont premièrement obtenues auprès des équipements eux-mêmes, les protocoles de contrôle d'équipements présentant généralement une couche protocolaire de description (cf. Chapitre III.1). Ces informations peuvent être augmentées par les informations issues d'un serveur de contexte (cf. section 5). Afin de simplifier la composition de services au sein des applications clientes, les équipements ou, du moins, leur représentation sur la plateforme doivent s'auto-décrire.

Les informations décrites par les composants sont de nature variée. Elles peuvent décrire des dépendances de ressources, des informations générales quand à la provenance du code du composant, la description de son contenu, des informations de QoS associé au fonctionnement, etc. Les unités de déploiement définies par des plateformes diverses, tels que Linux Debian, .NET, OSGi, MIDP, peuvent être considérés comme des composants (cf. Chapitre III.3.3) dont l'installation, le fonctionnement et la désinstallation sont conditionnées par la satisfaction de leurs dépendances vers d'autres unités de déploiement.

Dans le cadre du Home SOA, les composants fonctionnels sont les composants fournissant et requérant des services à l'exécution. Leur fonctionnement dépend par conséquent de la satisfaction de leurs dépendances vers des services fournis par d'autres composants. Ces dépendances et la liste des services fournis vont être décrites dans la description du composant. Les dépendances de services peuvent être caractérisées plus avant par un caractère obligatoire, un caractère multiple, et être l'objet d'un filtre de service [CeH03]. Dans un développement orienté objet, le composant sera caractérisé par un objet implémentant les services fournis, des méthodes de liaisons et des méthodes d'activation et de désactivation (cf. une implémentation possible dans la section Chapitre VI.1.2).

La description des dépendances de services par les composants permet l'automatisation de liaisons des composants aux services requis. Cette automatisation est déléguée à un gestionnaire par composant qui confronte la liste des services décrits comme requis par la description du composant et la liste des services présents dans le registre de services (cf. section 2.2). Pour maintenir cette connaissance à tout instant, le gestionnaire se met à l'écoute des événements concernant les services requis. Un certain nombre de règles sont suivies :

- Si les besoins obligatoires de services viennent à être satisfaits par des services disponibles, le composant est activé après avoir été lié à ceux-ci et après avoir

enregistré les services décrits comme fournis. L'objet de service (cf. section 2.2) est l'objet caractérisant le composant. Par la suite, les autres services optionnellement requis sont liés dès qu'ils sont présents.

- Si un service requis vient à disparaître alors que le composant est actif, le service est délié. Si le service délié était obligatoire et qu'aucun service présent ne peut le remplacer, le composant est désactivé. Dans ce cas, les services fournis sont retirés du registre de services.

Une composition spontanée et paire-à-paire s'effectue sur la plateforme de service grâce à ce modèle d'auto-description et d'automatisation des liaisons au niveau de chaque composant [2006-4]. De cette composition paire-à-paire émerge des applications complètes. Ces applications ont un comportement plus ou moins dégradé suivant le nombre et la qualité des composants activés, le nombre et la qualité des services liés.

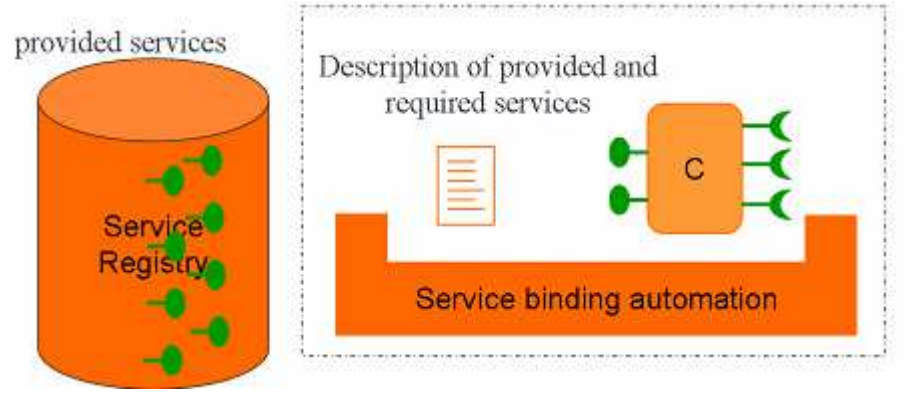


Figure 42 Automatisation de la composition de services

Afin de simplifier le développement, le gestionnaire des liaisons de services peut être écrit de manière générale par un conteneur dans le modèle de composants à service [CeH03]. Un conteneur est en général une partie de code implémenté par un spécialiste d'un aspect non-fonctionnel afin de gérer cet aspect pour tous les composants du modèle. Il doit avoir la capacité d'introspecter le fichier de description adéquat du composant lors de l'apparition du composant sur le cadriciel sous-jacent. Dans le cas du modèle de composants à service proposé, le conteneur doit pouvoir introspecter la description de tout composant présent sur la plateforme sous-jacente. A tout composant obéissant au modèle, le conteneur instancie et affecte un gestionnaire configuré de manière spécifique à sa description. Dans la Figure 42, le conteneur appelé "service binding automation" gère le cycle de vie du composant "C" aux dépendances de services décrites dans sa description, de façon réactive à la présence des services dans le registre ("Service Registry").

2.4 Communication par événements

2.4.1 Patron de conception "Observateur"

Le patron de conception "Observateur" (cf. Figure 43), très répandu dans la programmation orientée objet, voit ses limites dans une programmation orientée service où les différents objets observateurs apparaissent et disparaissent dynamiquement. Le patron de conception apporte une réponse simple à un problème courant :

- **Problème** : le problème peut être de deux types différents :
  - Plusieurs objets doivent être notifiés des changements d'une même donnée au cours du temps.

- Une liaison entre deux objets nécessite que les appels soient initiés parfois par l'un et parfois par l'autre des objets. Si les deux objets définissent un lien d'association vers l'autre, les deux objets sont alors fortement couplés.

- **Solution** (cf. Figure 43) : dans les deux cas, la solution impose une asymétrie à la relation. Elle distingue en effet un sujet observé et des objets "observateurs". Le sujet implémente une interface où des méthodes d'ajout et de suppression d'objets observateurs sont disponibles (interface "Subject"). L'observateur implémente une interface définissant une simple méthode de notification. Les objets observateurs s'abonnent et se désabonnent à la notification des changements en s'ajoutant et se retirant de la liste d'observateurs auprès du sujet. Le sujet notifie la liste d'observateur courante lors de chaque événement pertinent. La solution répond au premier besoin d'une liaison "un vers plusieurs" et au second besoin en découplant le serveur de l'implémentation des clients potentiels. En effet, l'implémentation de l'observateur peut encore invoquer directement le sujet (liaison "observes") tandis que l'implémentation du sujet ne connaît a priori que l'interface "observateur".

Dans une application importante, le patron "observateur" montre premièrement le désavantage de nécessiter l'allocation et la gestion d'une liste d'éléments multiples (généralement la classe Vector en Java) à chaque utilisation. Comme le remarquent les auteurs connus de la communauté OSGi [KrH04], la simple machine virtuelle Java arme plus d'une centaine de serveurs d'événements de ce type au démarrage. La redondance est gourmande en ressources mémoire. Pour des applications embarquées, elle semble déplacée.

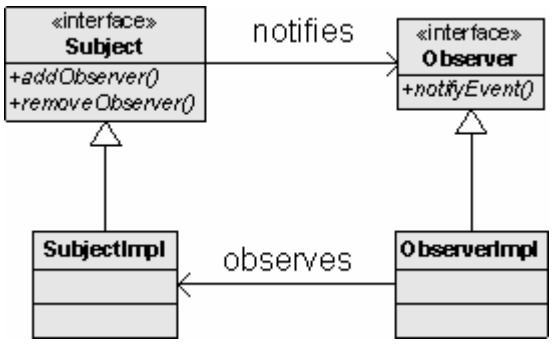


Figure 43 Patron de conception "Observateur"

Deuxièmement, sur une plateforme de services, qui est souhaitée être toujours disponible et où les objets apparaissent et disparaissent au gré des changements de configuration et des activités de l'utilisateur, le faible couplage de l'"observateur" s'avère encore trop important. En effet, l'observateur doit s'abonner et se désabonner à l'apparition et la disparition du sujet sur la plateforme tandis que le sujet rencontre des situations où les observateurs de la liste n'existent plus sur la plateforme, la méthode de suppression n'étant négligemment pas appelée par certains clients à leur départ. Ces problèmes sont dits "de cycle de vie" [KrH04].

2.4.2 Patron de conception "Tableau Blanc"

Le patron de conception "Tableau Blanc" (Whiteboard pattern [KrH04]) répond à ces problèmes par une solution reprenant le patron de conception "plateforme de services" (cf. section 2.2) :

- **Problème** : les défauts du patron "observateur" mentionnés ci-dessus, c'est-à-dire l'allocation multiple de listes d'objets au même objectif, la redondance du code de gestion de ces listes, les problèmes dus aux cycles de vie distincts des sources d'événements et de leurs observateurs.

- **Solution** (cf. Figure 44) : utiliser le registre de services en tant que liste d'observateurs. La solution inverse les rôles a priori naturels des sources d'événements et des observateurs : ces derniers deviennent des services qui s'enregistrent et se retirent du registre. Le sujet n'implémente plus de méthodes de gestion de liste et devient client des observateurs. Lors d'un événement, le sujet notifie tous les observateurs du type adéquat présents dans la liste tenue par le registre. La gestion des abonnements disparaît du code des observés et est reléguée à un gestionnaire unique sur la plateforme. Aussi, les observateurs ne se soucient plus de la présence des sources d'événements. De surcroit, une seule allocation de liste est effectuée.

Plus loin, le "Tableau Blanc" garde encore un couplage important entre le sujet et la liste de services observateurs associés. En effet, le sujet doit toujours effectuer une requête de découverte auprès du registre de services avec le nom de l'interface particulière des observateurs recherchés et notifier les objets adéquats un par un. De plus, si plusieurs sujets souhaitent des propriétés de délivrance particulière, comme la garantie de délivrance malgré les déconnexions, ils doivent chacun maintenir la liste de leurs abonnés et gérer ces propriétés.

Aussi, le "Tableau Blanc" impose la définition d'une interface particulière pour chaque type d'événement, ce qui peut induire une certaines lourdeurs pour la supervision de nombreuses données de grain fin.

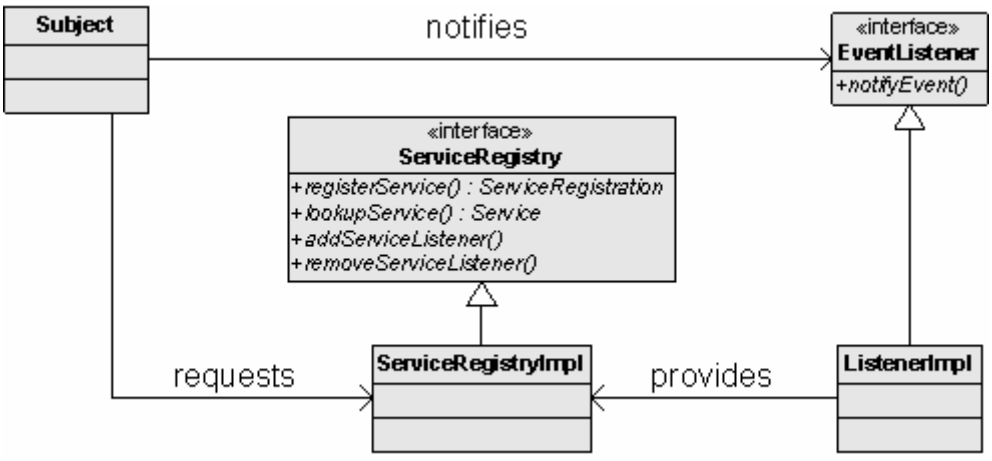


Figure 44 Patron de conception "Tableau Blanc"

2.4.3 Patron de conception "Publish-Subscribe orienté service"

Le patron de conception "publish-subscribe" peut alors être dérivé dans un patron nommé ici "publish subscribe orienté service". Il apporte une solution aux défauts mentionnés ci-dessus en apportant un couplage encore plus important :

- **Problème** : les deux derniers défauts du patron "Tableau Blanc" mentionnés ci-dessus, c'est-à-dire la gestion de la notification des observateurs par le sujet et la définition d'une interface de service particulière pour chaque type d'événement.
- **Solution** (cf. Figure 45) : suivre le patron de conception publish-subscribe et enregistrer un sujet général (ou observateur général) dans le registre de service. Chaque sujet ne requiert et ne notifie que le sujet général. Celui-ci a la responsabilité de requérir et de notifier les observateurs adéquats. Une seule interface de service est implémentée par les observateurs (interface "Subscriber"). La distinction des types de données échangés vient raffiner le système de notification. En effet, les sujets ("Publisher") notifient le sujet

général en indiquant le type de données notifiées dans un argument et les observateurs s'enregistrent sous le nom de l'interface unique avec une propriété indiquant le type de données auxquels ils sont intéressés.

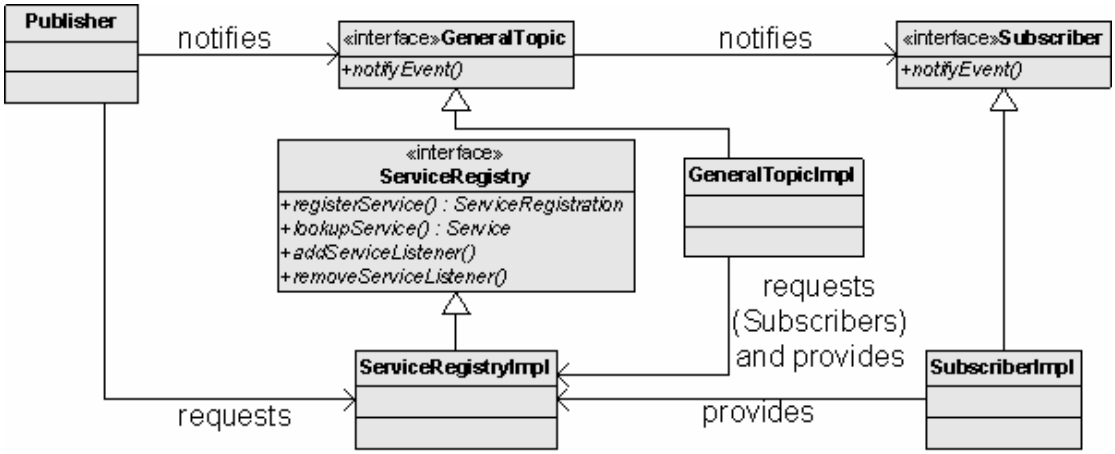


Figure 45 Patron de conception "Publish-subscribe orienté service"

2.4.4 La pertinence du découplage selon les situations

Il est notable toutefois que le surcroit de découplage apporté par ces patrons porte des défauts vis-à-vis du patron "observateur" :

- **Défaut du "Tableau Blanc"** : l'observateur ne possède pas de référence directe à la donnée observée, il ne peut pas surveiller une source identifiée (liaison "observes" de la Figure 43) et n'écouter que les événements de cette implémentation. Le défaut est surmontable si la source s'enregistre en tant que service et se déclare avec un identifiant. Mais cela complexifie la relation. Cette légère complexité semble cependant négligeable vis-à-vis des bénéfices du patron "Tableau Blanc".
- **Défaut du "Publish-Subscribe orienté service"** : cette solution découple plus encore les sujets et les observateurs que le "Tableau Blanc". Plus aucun appel direct n'intervient entre eux. Les notifications des sujets sont à la destination du seul sujet général. Le sujet ne peut donc pas obtenir de retour à sa notification de la part des observateurs. Le mode d'interaction "sollicit-response" (cf. Chapitre II.2.4) n'est pas permis par ce patron, ce qui a un impact dans l'implémentation de protocole comme DPWS (cf. Chapitre VI.1.3.3). Seul le premier des deux problèmes adressés par le patron "Observateur" n'est résolu par le patron "Publish-subscribe orienté service".

En conclusion, chacun des patrons exposés possède des atouts et des défauts pertinents selon les situations. L'Observateur est adéquat pour le découplage de deux objets aux cycles de vie identiques. Le Tableau Blanc est adéquat pour une relation "un vers plusieurs" où il est important de conserver le mode "sollicit-response" entre le serveur d'événement (par exemple un équipement) et le client observateur. Le Publish-Subscribe orienté service est adéquat pour la gestion de flux de données à grain fin de multiples serveurs vers de multiples clients aux besoins diverses. Pour ces raisons, le Tableau Blanc est le plus adéquat dans la conception des pilotes orientés services de la section suivante.

3 DISTRIBUTION ET PILOTES ORIENTES SERVICES

Le concept de pilote orienté service offre la transparence de la localisation des entités pour la recherche et l'utilisation des services disponibles, soit la résolution des deux

premières exigences techniques mentionnées dans la problématique (cf. Chapitre IV.2). La recherche de services locaux et distribués est indiscernable de la recherche de services distribués. Les pilotes orientés services cachent aussi les détails protocolaires de la communication distante lorsqu'une communication distribuée est établie entre un objet de la plateforme et une entité distante. La dernière exigence technique de possibilité d'ajout des propriétés de reconfiguration, de sécurité, de fiabilité est permise et simplifiée par la séparation des préoccupations offerte par la séparation de l'application et des pilotes orientés service. Cet ajout peut a priori être effectué directement au niveau de la logique du pilote orienté service.

Le principe majeur des pilotes orientés services est la population dynamique du registre de services avec les objets mandataires représentant les équipements distants. Un objet mandataire est instancié pour chaque entité distante. Le pilote est responsable du maintien dynamique dans le registre de services de la liste des entités distantes disponibles selon ce protocole. Les événements locaux de services associés permettent aux applications de réagir dynamiquement au départ et à l'arrivée des équipements. La disponibilité dynamique des entités pervasives de l'environnement est réifiée sur la plateforme de services par un pilote orienté service pour chaque protocole.

Ces objets sont enregistrés avec la même interface que les objets locaux prêts à être exportés selon le même protocole sur le réseau. Cela permet aux objets clients de découvrir et de contrôler de façon identique tant les services locaux que les services distants. Dans les deux cas, les objets de services sont accédés par appels programmatiques directs dans le langage de programmation de la plateforme. Dans le cas où les services locaux sont des mandataires, les appels cachent des appels protocolaires distants. Les pilotes orientés services sont par conséquent responsables de la sérialisation et dé-sérialisation des appels programmatique. Les détails protocolaires sont masqués par l'utilisation des services mandataires locaux présentés par le pilote.

Un exemple dans la littérature est l'UPnP Base Driver décrit dans [DFKM02] et normalisé dans le chapitre OSGi intitulé "UPnP Device Service Specification" [OSGi05]. Cette thèse réexamine les scénarios d'usage et exigences techniques associés à ces pilotes [2007-5], spécifie les interfaces externes d'un tel pilote pour le protocole DPWS et l'implémente dans une contribution open source du projet Amigo[1] [2008-2].

La transparence de la recherche et de l'utilisation des services distants est apportée par un patron de conception dérivé du patron "export-bind" dans la section 3.1. L'usage de mandataires afin de représenter de manière identique objets locaux et distants est expliqué dans la section 3.2. Enfin, la section 3.3 expose la pertinence du patron de conception "Tableau Blanc" pour les modes de communication événementielle des équipements.

3.1 Patron de conception "export-bind orienté service"

Le patron "export-bind" (cf. Chapitre II.3.2.4) permet la transparence de l'obtention d'une référence d'objet distant par la requête d'un nom auprès d'un objet – Binding Factory – lié à un serveur de nommage. Le patron de conception "export-bind orienté service" dérive de celui-ci en apportant le couplage lâche du registre de services dans le modèle d'interaction. Ce dérivé est présenté dans les articles [2006-2][2006-4][2006-5]. Il est aussi commenté dans la littérature [Kra08]. Il répond de façon similaire à un problème élargi :

[1] France Telecom DPWS Base Driver specification, Java API, implementation and examples: http://gforge.inria.fr/frs/?group_id=160&release_id=1804

- **Problème** : le client et le serveur d'une interface programmatique ne sont pas conçus pour fonctionner de manière distante avant l'exécution. Le client connaît toutefois un nom d'interface et des propriétés associées à l'objet distant.
- **Solution** (cf. Figure 46) : la solution repose sur les notions d'annuaire de services – Service Directory – d'usine à exporter – Export Factory – et d'usine à liaisons – Binding Factory. L'annuaire de services est un annuaire de références définies par un ensemble libre de propriétés. Ces propriétés contiennent généralement le nom d'une interface de service et plusieurs attributs qualifiant le l'origine et le fonctionnement de l'entité référencée. Le client demandeur de service et le serveur fournisseur de service partagent l'annuaire et tout ou partie de l'ensemble défini par l'interface de service et la catégorisation des propriétés. Le serveur, dès qu'il est prêt à servir son interface de manière distribuée, appelle l'usine à exporter afin que celle-ci construise la référence qualifiée (étape 1 : "create"). Il enregistre alors cette référence auprès de l'annuaire (étape 2 : "export"). Le demandeur recherche le service selon un filtre évaluant différentes propriétés auprès de l'annuaire qui lui retourne la référence associée (étape 3 : "look up"). Cette référence est conçue pour être interprétable par l'usine à liaison. L'usine à liaison est un composant accessible dans l'environnement d'exécution du demandeur. Le demandeur passe la référence à l'usine à liaison qui crée ou obtient un objet proxy encapsulant le stub de l'objet distant (étape 4 : "bind"). Il peut alors appeler le serveur distant au travers des appels du mandataire (étape 5 : "access").

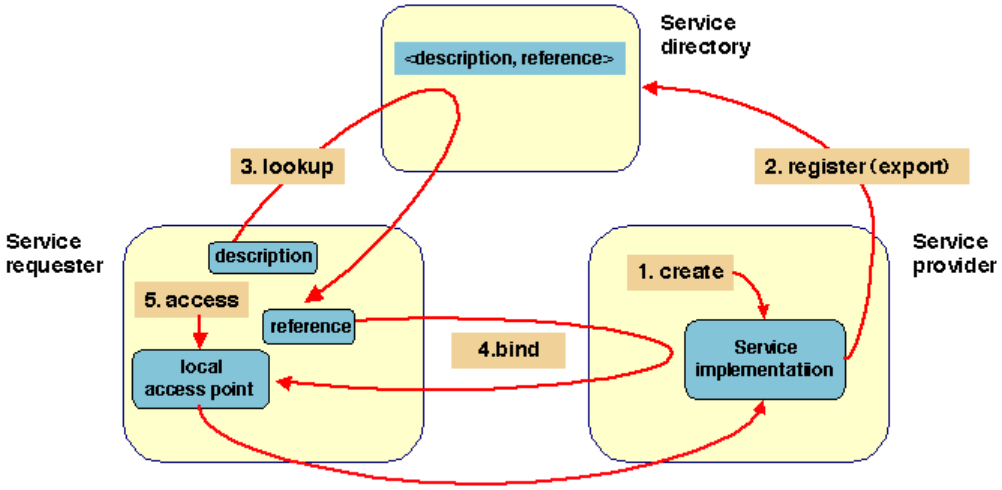


Figure 46 Patron de conception export-bind et découverte de service [Kra08]

Ce patron est utile pour la mise en relation de demandeurs et fournisseurs de services distants. Les opérations auprès du registre de services et la communication aux objets récupérés demeurent distantes. Cette limite est adressé par le patron de conception "Registre de services mandataires" décrit dans la section suivante.

3.2 Registre de services mandataires

Le patron de conception "Registre de services mandataires" (ou "Service Proxy Registry") utilise le patron "plateforme de services" (cf. section 2.2). Il est implicitement marié au patron "export-bind orienté services" de la section précédente :

- **Problème** : les demandeurs de services possède un registre de service local pour la recherche de services locaux et un annuaire de service distants pour la

recherche de services équivalents mais distants. La recherche et de la communication aux entités locales et distantes sont pourtant souhaitées transparentes.

- **Solution** (cf. Figure 47) : la solution consiste en la population spontanée du registre de service local de services découverts sur le réseau et représentés par des objets mandataires grâce à l'utilisation du patron "export-bind orienté service". L'entité responsable de la population du registre est appelé "pilote orienté service" (ou "Service Oriented Driver"). Il peut être assimilé au "Discovery Base Driver" défini dans [DFKM02] et [OSGi05]. Cette entité découvre de façon active ou passive les services distants (étape "remotely discovers") et les enregistre dans le registre de services local (étape "remotely binds") après instanciation d'un objet mandataire par l'usine à liaison ("Binding Factory"). Dès qu'un service distant se déconnecte du réseau, le pilote retire le mandataire local du registre. Grâce au patron "plateforme de service", le demandeur de service local découvre de manière active ou passive la représentation local du service qu'il accède alors directement dans le langage de programmation de la plateforme (étape "locally discovers and binds"). Il est aussi notifié du départ du service local qui signale l'indisponibilité du service distant.

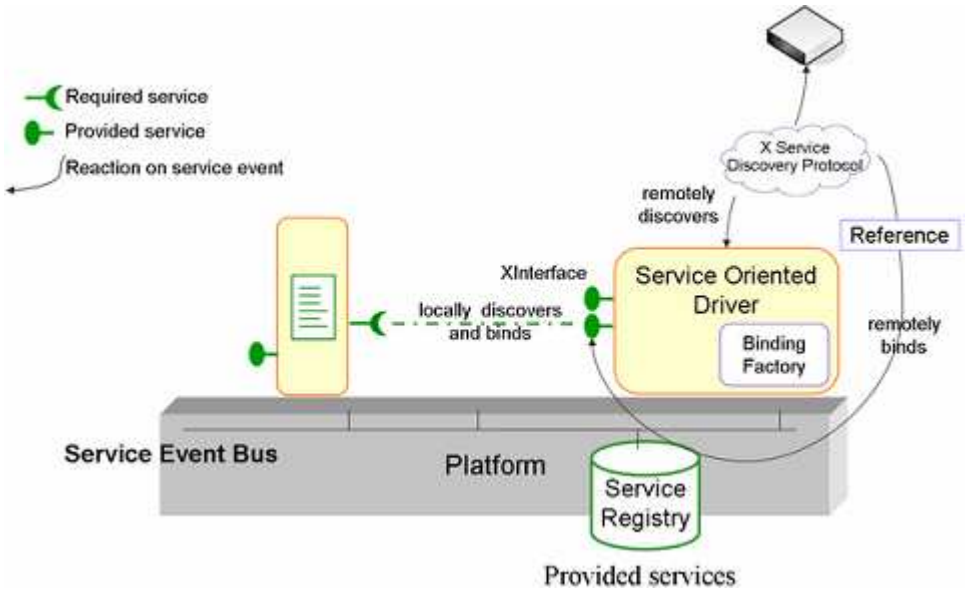


Figure 47 Pilote orienté service pour l'import

Le pilote orienté service peut aussi être utilisé pour un export transparent de service sur le réseau. Il est alors marié implicitement à la partie export du patron "export-bind orienté service" (cf. Figure 48). Le fournisseur de service enregistre et délivre localement un service selon une interface définie requise par le pilote orienté service (étape "locally registers and delivers"). Ce dernier le découvre localement de façon active ou passive, crée un squelette et une référence d'export qu'il enregistre auprès de l'annuaire distant (étape "remotely registers"). Une plateforme distante peut alors découvrir, lier le fournisseur de services et appeler ses méthodes à distance ("remotely discovers and accesses").

Le pilote orienté service définit une interface programmatique pour la représentation d'un service selon l'ensemble protocolaire associé, ici nommé X. Cette interface est la transposition dans un modèle objet de la représentation des équipements dans la couche de description défini par le protocole X. Afin qu'un client local puisse accéder de manière transparente à des services locaux et distants, ces services doivent implémenter cette même interface ("XInterface"). Dans les deux cas, la communication entre l'objet client et l'objet de

service s'effectue par appel direct dans le langage de programmation de la plateforme. Si l'objet de service est un objet mandataire, alors ce dernier retransmet les appels dans le protocole de contrôle X au service distant correspondant (cf. Chapitre II.3.2.3).

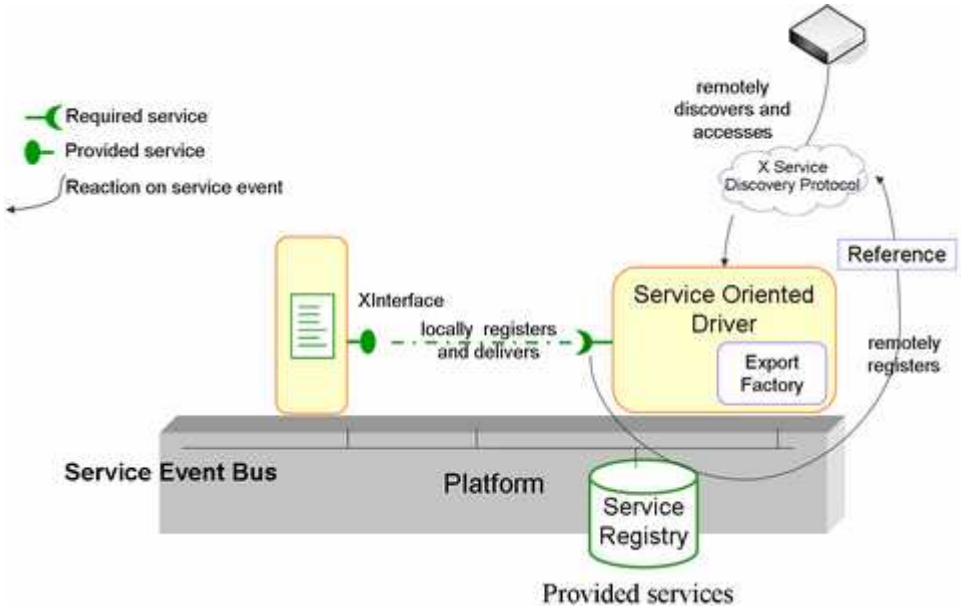


Figure 48 Pilote orienté service pour l'export

3.3 Communication locale et distante par événements

Dans la plupart des intergiciels distribués orientés services, une couche d'événement est spécifiée (cf. Chapitre II.2.6). La notification d'événement inverse le rôle des clients et des serveurs dans leur mode d'interaction : le serveur a l'initiative des appels vers le client. Ce renversement nécessite que le client se déclare auprès du serveur au préalable. Cette déclaration est effectuée dans un appel appelé souscription. Les protocoles du réseau domestique définissent la notification exclusivement selon le patron de conception réseau "Observateur". Le patron "publish-subscribe" n'est pas commun.

Ce mode de souscription et de notification distantes se doit d'être représenté de manière transparente dans le langage de programmation. A cette fin, les patrons de conception objet de communication par événements (cf. section 2.4) peuvent tous a priori être utilisés entre les objets clients et serveurs. Comme ces entités possèdent généralement des cycles de vie différents et que la notification d'un serveur s'adresse potentiellement à plusieurs clients, le patron Observateur est à écarter pour les défauts mentionnés dans la section 2.4. Ensuite, comme les événements ne concernent pas des flux de données à grain fin – UPnP définit même une souscription exclusivement à l'ensemble des événements d'un service Chapitre III.1.1.1 – et que certains protocoles comme DPWS définissent un mode d'interaction "sollicit-response", le Tableau Blanc semble un meilleur candidat que le Publish-Subscribe orienté service.

Le pilote orienté service gérera par conséquent les événements de la façon suivante :

- **A l'import** : les clients intéressés par un événement particulier dans le protocole X enregistrent un objet à l'interface XEventListener avec des propriétés décrivant l'événement attendu (identifiant d'équipement, identifiant de service, type d'équipement, type de service, nom de l'événement, etc.). Le pilote découvre activement ou passivement ce service enregistré et souscrit à ce type d'événement pour le compte de ce client auprès des services adéquats. Le pilote renouvelle éventuellement la souscription tant que le service XEventListerner

est présent dans le registre. Si de nouveaux serveurs pertinents apparaissent, le pilote souscrit auprès d'eux. Chaque fois qu'un serveur envoie un message de notification, le pilote le reçoit et invoque la méthode de notification du service XEventListener avec les paramètres dé-sérialisés. Le retrait du service XEventListener du registre est relayé sous forme de désabonnement auprès des serveurs distants. En parallèle, les serveurs locaux gèrent la présence du service XEventListener d'une manière similaire : tant que le service est présent dans le registre, la méthode de notification est directement appelée avec les paramètres adéquats dès que le serveur doit notifier un événement.

- **A l'export** : le pilote représente sur le réseau les services locaux implémentant l'interface XInterface. Il relaie les souscriptions provenant de clients en enregistrant un ou plusieurs services de type XEventListener dans le registre local. Comme dit précédemment, lors de chaque événement, les serveurs locaux appellent les méthodes de notifications de tous les services de type XEventListener dont les propriétés montrent un intérêt pour l'événement donné.

En conclusion, le mécanisme de "Tableau Blanc" inverse les rôles du client et du serveur dans les interactions protocolaires à l'initiative du serveur. Le client devient un service enregistré localement. Cet enregistrement signale aux serveurs locaux et au pilote que le client est intéressé par la notification d'événements au type spécifié dans les propriétés du service. Le pilote souscrit aux services distants correspondant et devient alors client pour le service enregistré : il appelle les méthodes du service chaque fois qu'il reçoit des messages de notification en provenance des serveurs distants.

4 HETEROGENEITE ET PILOTES RAFFINES

Les pilotes orientés services réalisent la vision que toutes les entités pervasives peuvent être composées localement dans le langage programmation sur une seule et même plateforme. Grâce à cette technique, le défi posé par la gestion de l'hétérogénéité des protocoles est ramené à un problème de composition locale d'interfaces programmatiques distinctes, soit le problème de fragmentation d'interfaces (cf. Chapitre II.4.2 et [Vay01]).

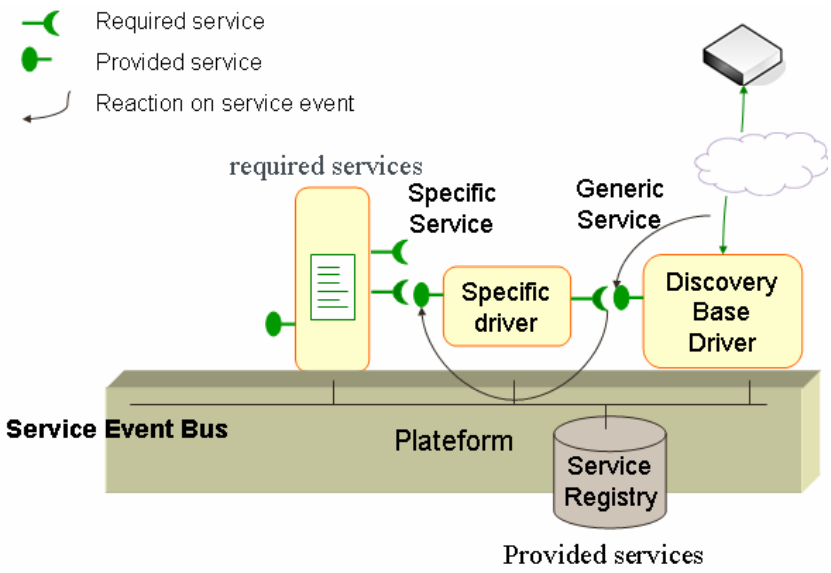


Figure 49 Chaine de pilotes : "Discovery Base Driver" et "Specific Driver"

Des techniques de médiation technique sont alors employées afin de représenter de manière uniforme des interfaces sémantiquement proches mais syntaxiquement différentes. Les "pilotes orientés services raffinés", appelés simplement "pilotes raffinés" par la suite,

sont la réponse technique du Home SOA à ce besoin de rapprochement d'interfaces [2008-5][2007-4][2007-2]. Alors que le pilote orienté service est associé à une interface unique attachée à un protocole donné, le pilote raffiné est associé à une interface unique attachée à un domaine d'application donné. Les pilotes raffinés créent un objet médiateur ("adapteur") implémentant cette interface unique pour chaque objet d'une interface reconnue sémantiquement proche. Cette deuxième couche de pilotes est responsable du gommage de l'hétérogénéité des interfaces dans un même domaine d'application.

Chaque pilote raffiné réagit dynamiquement à la disponibilité des services d'interfaces particulières dans le registre de services. Dès qu'un service de ce type est enregistré, mis à jour, retiré dans le registre de services, le pilote enregistre, met à jour, retire l'objet médiateur approprié. Grâce à la chaine de réactions construite avec les deux niveaux de pilotes (cf. Figure 49), le registre est dynamiquement peuplé d'objets exempts de détails protocolaires et adaptés à la sémantique de domaines d'applications spécifiques.

Tandis que la section 4.1 introduit les détails de la conception de pilotes raffinés, la section 4.2 décrit les interfaces uniformes choisies pour les situations de contrôle multi-protocolaire générique d'équipements, de contrôle spécifique d'équipements multimédias, de passerelle topologique. La section 4.3 montre la possibilité de génération statique de pilotes raffinés à partir de description protocolaire de service. La section 4.4 présente une piste de travail pour la génération automatique d'adapteurs sémantiques.

Le concept de pilote raffiné répond aux exigences de couplage lâche et d'interfaces uniformes décrit dans la section Chapitre IV.2.2. Le choix d'une technologie modulaire telle qu'OSGi ou .NET de la plateforme d'exécution répondra à l'exigence de chargement logiciel à la demande (cf. Chapitre VI.1.1). L'association du téléchargement modulaire à la demande et la modularisation des pilotes répond au besoin de déploiement protocolaire réactif. Le dernier besoin de négociation et d'adaptation de flux initiés par les applications de contrôle est résolu par la généricité des interfaces pivots utilisées dans le cadre des applications multimédias permettant la négociation des flux et par l'utilisation de capacités de transodage dans le réseau (cf. Chapitre VI.5).

4.1 Pilotes orientés service raffinés

Le pilote raffiné crée et maintient dynamiquement une liste d'objets adapteurs. Chaque objet adapteur est créé pour représenter dans une interface unique un objet à l'interface sémantiquement proche. Conformément au patron de conception "Adapteur" (cf. Figure 50), les appels du client sur l'objet adapteur ("Adapter") cachent l'appel correspondant au serveur adapté ("Adaptee"). L'adapteur classique répond au problème d'incompatibilité des interfaces d'un objet client et d'un objet serveur dans un environnement statique en leur permettant d'interagir entre eux sans modification des objets initiaux.

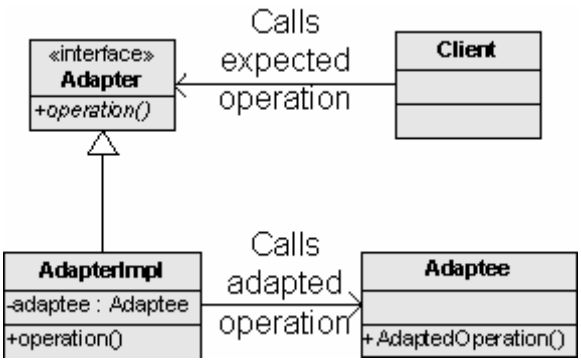


Figure 50 Patron de conception Adapteur

L'interface dans laquelle représente le pilote raffiné les objets divers est généralement une interface requise par une application. Les pilotes raffinés transforment donc des objets fournis divers en objets satisfaisant des dépendances de services. Le pilote raffiné peut être érigé en patron de conception :

- **Problème** : un client demande auprès d'un registre de service local les services suivant une interface spécifique à un domaine d'application. Certains services apparaissent dans le registre avec une interface sémantiquement proche mais syntaxiquement différente. Le client ne peut pas interagir avec ces services en raison de l'incompatibilité des interfaces.

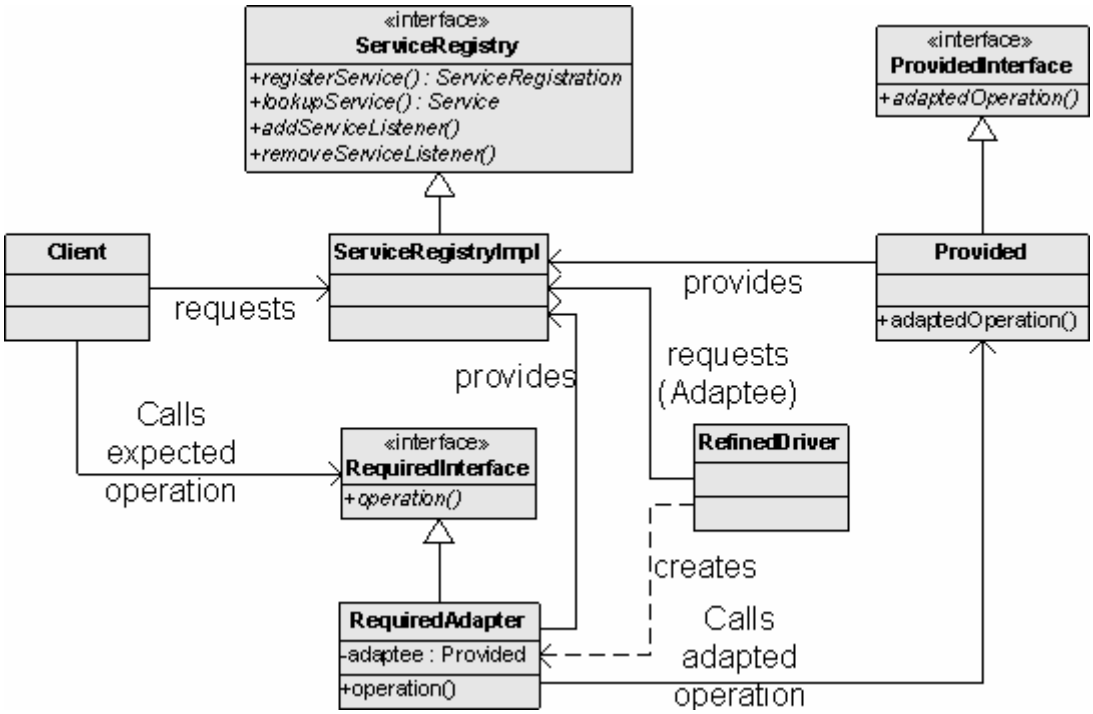


Figure 51 Patron de conception Pilote Raffiné

- **Solution** (cf. Figure 51) : la solution consiste en la population dynamique du registre de service local d'un objet adapteur implémentant l'interface requise pour chaque service fourni. L'entité responsable de la population du registre est appelé pilote raffiné (ou "Refined Driver"). Le nom est repris du chapitre "Device Access" de la spécification OSGi [OSGi05] mais le concept est ici original. Cette entité découvre de façon active ou passive les services fournis d'un type à adapter, crée un objet adapteur le référençant et enregistre celui-ci dans le registre de services. Dès que le service fourni est retiré, le pilote retire l'objet adapteur du registre. Grâce à l'utilisation du patron Plateforme de Service, le demandeur de service découvre de manière active ou passive la représentation adaptée du service qu'il accède alors directement selon l'interface compatible. Il est aussi notifié du départ du service adapté qui signale l'indisponibilité du service originalement fourni.

Le pilote orienté service agit comme une usine (patron de conception "Factory") créant automatiquement un adapteur pour chaque service découvert. La méthode de création est appelée de manière réactive à la découverte d'un service à adapter. Le pilote peut être conçu dans un mode dynamique de création et de destruction à chaque association avec un nouveau service ou dans un mode de gestion d'un pool limité d'adapteurs avec retraitement de chaque objet lors du départ d'un service pour l'utilisation d'un autre service présent. La limite

du pool peut être consignée en fonction de contraintes générales de mémoire ou en fonction du nombre de demandeurs de service à pourvoir.

4.2 Exemples d'interfaces programmatiques pivot

La médiation technique est appliquée dans les exemples de scénarios décrits dans la section Chapitre II.4.1. Le point de contrôle générique sur le réseau domestique de la section 4.2.1 est un scénario choisi dans les contributions [2008-5][2008-4]. Ce procédé est aussi employé dans l'exemple de passerelle topologique en section 4.2.2 [2008-5]. Le pivot pris pour les équipements multimédias des applications de Home cinéma intelligent [2008-5] et de communication interpersonnelle enrichie [2008-6] est décrit dans la section 4.2.3.

4.2.1 Point de contrôle générique sur le réseau domestique

Un premier scénario est la médiation générique de tout équipement du réseau domestique dans une interface de contrôle unique. La section Chapitre II.4.1.1.3 décrit ce besoin dans un scénario de "self-care". L'indépendance de l'application par rapport aux différents protocoles présents repose sur un pivot programmatique unique dans lequel tous les équipements du réseau domestique sont représentés. L'interface la plus appropriée est celle qui est suffisamment générique pour la représentation d'ensemble d'actions et d'événements aux types divers dans le cadre d'applications du réseau domestique (cf. Figure 52). Après analyse des différents protocoles de cet environnement (cf. Chapitre III.1), DPWS est l'ensemble protocolaire au protocole de description le plus générique : WSDL (Web Service Description Language). Pour cette raison la représentation objet associée à WSDL est l'ensemble d'interfaces pivot de choix. Le standard JWSDL, représentation objet de WSDL dans le langage Java est choisi dans l'expérimentation [2008-4] (cf. Chapitre VI.4.1).

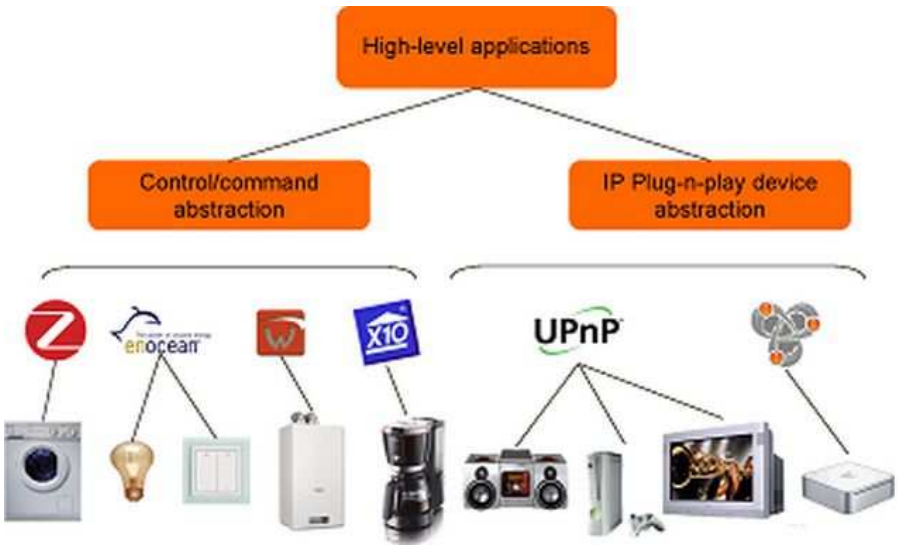


Figure 52 Médiation technique à partir d'équipements aux protocoles et aux domaines d'applications différents

4.2.2 Contrôle de réseaux capillaires et pont protocolaire simple

Dans les situations de contrôle de réseaux capillaires aux bus de terrain divers (cf. Chapitre III.1.2), le protocole pivot choisi peut être plus simple. En effet, les bus de terrain sont généralement les protocoles de contrôle d'équipements très contraints où la pile TCP-IP est considérée couteuse. UPnP et son protocole de description ad hoc (UPnP Template Language) sont les technologies de choix. Le projet européen Teaha a pris cette décision par exemple [DSTG07]. [2008-5] montre un exemple d'application avec ce pivot dans

l'implémentation du scénario de partage de services entre plateformes intelligentes de la section Chapitre II.4.1.1.2.

La Figure 53 montre une implémentation possible avec les techniques du Home SOA. Le pilote orienté service nommé UPnP Base Driver est utilisé par la passerelle domotique ("Automation Box") et la plateforme d'applications générique "Home SOA Box" afin de communiquer en masquant les aspects distribués aux développeurs d'applications. Un pilote raffiné (nommé ZigBee-UPnP) effectue la conversion d'objets représentant les équipements ZigBee en objets représentant des équipements UPnP sur la passerelle domotique. Il effectue la conversion inverse sur la plateforme "Home SOA Box". Cette architecture permet à la plateforme Home SOA de Maxandre de piloter des équipements ZigBee au travers du protocole UPnP malgré l'incapacité de la plateforme à communiquer dans les fréquences radios des équipements ZigBee.

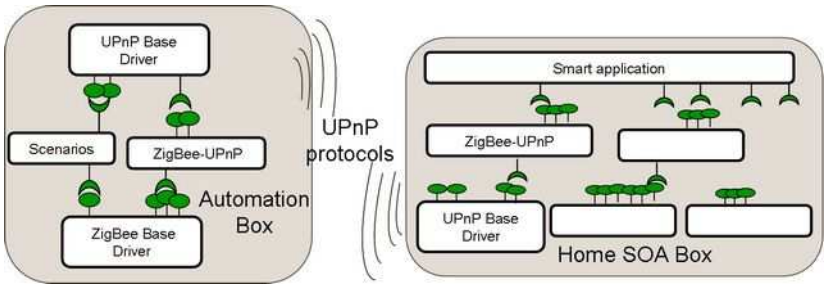


Figure 53 Pont protocolaire entre deux plateformes intelligentes

4.2.3 Interface générique pour le contrôle multimédia

Dans les applications impliquant le contrôle d'équipements multimédias, la médiation technique est utile afin de représenter tous les équipements multimédias contrôlables dans des interfaces uniformes. Ces équipements sont aujourd'hui les serveurs de contenus UPnP et iTunes et les serveurs de restitutions UPnP. Les applications de Home Cinéma intelligent [2008-5] et de communication personnelle enrichie [2008-6] utilisent des interfaces représentant génériquement des sources (interface "MediaSource") et des puits (interface "MediaSink") multimédias. Ce raffinement cachent le détail des protocoles de descriptions UPnP et iTunes et embrassent la sémantique du domaine multimédia.

L'interface MediaSource offre des méthodes de navigation dans un répertoire de contenus multimédias, de négociation des paramètres de flux et de pilotage les flux poussés vers les puits multimédia. L'interface MediaSink expose des méthodes de contrôle du rendu audio-visuel, de négociation de paramètres de flux et de pilotage des flux tirés depuis les sources multimédia. Le contrôle du rendu audio-visuel est lié au contrôle des paramètres tels que la luminosité, la couleur, le niveau sonore, la balance des fréquences sonores. Les flux peuvent être fournis selon des protocoles de flux divers, principalement RTP et HTTP et selon des formats multimédias variés tels que mpeg2, divx. Ce modèle générique de sources et de puits multimédia est proche de celui de la spécification UPnP AV [UAV02], très répandue aujourd'hui.

4.3 Génération statique de pilotes raffinés

Les pilotes orientés services sont des ponts de bas niveau permettant de réifier sur une plateforme logicielle tout équipement compatible avec un protocole de découverte. Les applications clientes découvrent, introspectent, contrôlent et supervisent les équipements par des appels objets. Les développeurs de ces applications n'ont pas conscience des détails protocolaires répartis de découverte et de communication.

Cependant, les protocoles de contrôle tels qu'UPnP, DPWS, EIB, X10, etc. sont génériques et permettent la spécification de services dans différents domaines d'application. Cette généralité se retrouve au niveau du lot d'interfaces (API) implémenté par les objets mandataires présentés par le pilote orientés service. La compréhension de cette API générale est équivalente à celle du protocole de description de l'ensemble protocolaire donné. Le développeur doit par exemple connaître les détails de la description des équipements spécifiques visés afin d'obtenir les méthodes spécifiques par l'API générique d'introspection objet. La Figure 54 montre un extrait de la description d'un serveur de contenu UPnP (Media Server). Le lecteur y lit le nom de l'action "Play" et la liste d'arguments associée. Le chapitre "UPnP Device Service" de la spécification OSGi [OSGi05] définit une telle API objet générale. La Figure 55 montre le code à développer afin d'appeler l'action Play dans cette API.

```
<action>
  <name>Play</name>
  <argumentList>
    <argument>
      <name>InstanceID</name>
      <direction>in</direction>
      <relatedStateVariable>A_ARG_TYPE_InstanceID</relatedStateVariable>
    </argument>
    <argument>
      <name>Speed</name>
      <direction>in</direction>
      <relatedStateVariable>TransportPlaySpeed</relatedStateVariable>
    </argument>
  </argumentList>
</action>
```

Figure 54 Action Play dans le protocole de description d'UPnP

```
Dictionary dictionary = new Hastable();
Dictionary.put("InstanceID", "0");
Dictionary.put("Speed", "1");
upnpDevice.getDescription(null)
       .getService("AVTransportService")
       .getAction("play").invoke(dictionary);
```

Figure 55 Appel de l'action Play dans l'API Java "UPnP Device Service"

La correspondance bijective entre la description des équipements, services, opérations et arguments des protocoles de contrôle domestique et l'API objet généraliste généralement disponible permet la génération du code d'appel des opérations distantes à partir de fichiers de description protocolaire. L'exemple de l'API Java spécifiée par l'Alliance OSGi pour UPnP montre la simplicité laborieuse du développement manuel de ce code. Des exemples similaires peuvent être offerts à partir d'une description de service UPnP et du code Java pour le contrôle de ce service dans l'API du kit de développement d'Intel[1], ou encore à partir d'une description de service en WSDL et l'implémentation de l'appel d'opérations dans l'API procurée par Apache Axis[2]. La génération de code source de stubs à partir de descriptions de service UPnP et Web Services est d'ailleurs offerte respectivement par les outils d'Intels et les outils d'Apache Axis.

[1] Outils Intel pour le développement d'applications UPnP : http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/overview/index.htm

[2] http://ws.apache.org/axis/

La transparence des détails protocolaires de la description peut donc être fournie à un plus haut niveau pour les développeurs d'applications pervasives. Avec les techniques du Home SOA, l'outil correspondant est un générateur d'API spécifique et du pilote raffiné associé. [2007-4]. Ce pilote crée et enregistre des objets adapteurs implémentant les interfaces générées et référençant des objets à l'interface du pilote orienté service associé au protocole général sous-jacent. Le code à développer pour l'invocation de l'action Play du service AVTransportService du Media Renderer UPnP devient alors celui de la Figure 56.

```
mediaRenderer.getAVTransportService().play(0,1);
```

Figure 56 Appel de l'action Play dans l'API générée par l'UPnP Device Generator

Les scénarios et les besoins d'une projection entre description de service UPnP et API Java spécifique ont été soumis dans un document technique accepté par l'Alliance OSGi en tant que "RFP 72 Extended Mapping for UPnP Discovery Transparency" [2006-3]. L'outil a pu être utilisé dans les preuves de concept (et probablement les futures implémentations de référence) du protocole UPnP Device Management en cours d'élaboration[1], pour le contrôle d'équipements multimédias UPnP [2007-4] et pour le contrôle de caméras UPnP dans le prototype de communication personnelle enrichie [2008-6]. L'UPnP Device Generator est décrit avec détails dans la section Chapitre VI.4.3.

4.4 Médiation sémantique à la volée

La fragmentation d'interfaces est une limite connue des Architecture Orientée Service. Le couplage lâche du paradigme SOA repose sur le partage de mêmes interfaces entre fournisseurs et demandeurs de service. Ces interfaces sont définies syntaxiquement. Cette syntaxe est le minimum imposé par le SOA. Comme des objectifs sémantiquement uniques peuvent être définies par des interfaces syntaxiques différentes par des acteurs différents – même au-dessus d'un même protocole de description – la lâcheté du couplage n'est pas parfaite (cf. Chapitre II.4.2.4).

Les méthodes de médiation de service par pilotes raffinés permettent de représenter dans une même interface des objets à aux interfaces sémantiquement proches mais syntaxiquement différentes. Cela permet au client de cette interface unique d'interagir avec des serveurs à l'interface initialement incompatible.

L'élaboration de pilotes raffinés pour cet objectif de rapprochement d'interfaces sémantiquement proche est jusqu'ici présentée comme manuelle. La génération de code source de la section 4.3 ne vise qu'à simplifier le développement de l'appel d'opérations spécifiques à un équipement sans prise en compte d'interfaces programmatiques définies au préalable par des applications existantes. Les mécanismes de chainage d'adapteurs proposés par [Vay01] et [PoF03] reposent aussi sur des adapteurs développés manuellement.

Ces travaux de thèse proposent une piste de travail pour la génération automatique d'adapteurs sémantique à partir de la connaissance de deux interfaces sémantiquement proches, l'une requise et l'autre fournie sur la plateforme de service. Les techniques de Home SOA pourraient être enrichies par la mise au point d'un tel système dans des cas complexes.

L'adaptation automatique est effectuée en deux phases successives :

[1] Présentation par Maxime Vincent et l'auteur durant la réunion de l'OSGi User Group France le 29 janvier 2008 : http://france.osgiusers.org/wiki/uploads/Meeting/upnpexe200801.pdf

- **Détection de services semblables** : cette phase consiste à détecter des méthodes qui correspondent sémantiquement dans deux services différents, l'un requis, l'autre fourni sur la plateforme de services. Le nom des services, le nom, le nombre et le type des arguments d'entrée et de sortie de certaines méthodes sont des critères de raisonnement. Le raisonnement sur les entrées et sorties de méthodes peut emprunter des algorithmes existants [Tha94]. Le raisonnement sémantique sur les noms de services et de méthodes est délégué à un système de raisonnement basé sur des ontologies. A partir de deux interfaces suffisamment proches, un plan d'adaptation de méthodes est calculé.

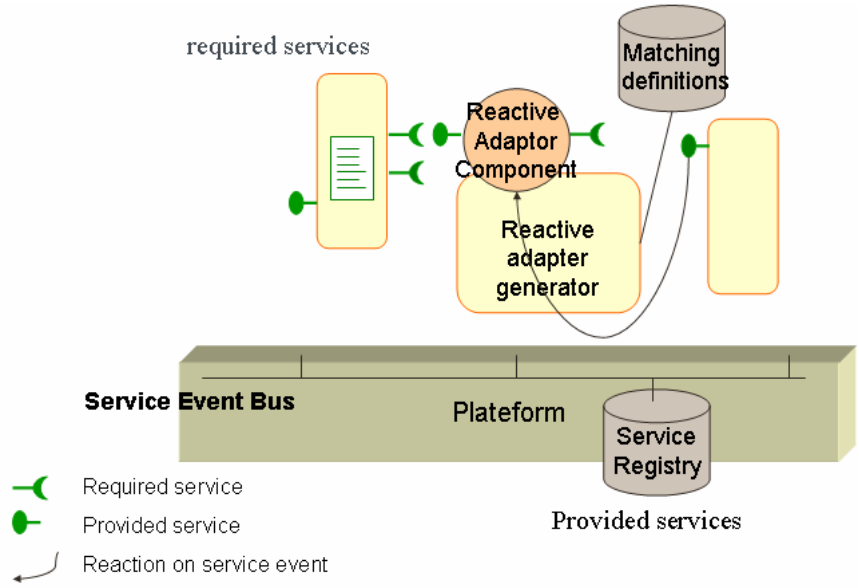


Figure 57 Génération d'adaptateurs sémantiques à la volée

- **Génération dynamique de pilotes raffinés** : étant donné un plan d'adaptation de méthodes indiquant les méthodes et arguments correspondant sémantiquement, il est aisé de construire dynamiquement un adapteur et même un pilote raffiné pour l'interaction entre le client et le serveurs aux interfaces incompatibles. Cette génération peut utiliser l'API d'introspection de langages de programmation tels que Java (Java Reflection API) [KeB03] ou la génération dynamique de code binaire (cf. Chapitre II.3.2.7). Le premier mode est plus aisé mais le second donne des adapteurs plus efficaces (rapidité de temps de traduction d'appel).

La Figure 57 illustre le système d'adaptation sémantique à la volée. Un répertoire de définitions de correspondance de services et de méthodes est référencé par le composant de génération dynamique d'adapteurs ("Reactive adapter generator"). Ce dernier parcourt la liste des services fournis maintenue par le registre de services, parcourt la description des services requis des composants les déclarant par auto-description (cf. section 2.3) et tente de détecter les services aux noms et méthodes semblables. Dès qu'un service fourni et un service requis sont détectés comme semblables, il génère un pilote raffiné ("Reactive Adaptor component") pour l'adaptation spécifique de ces services. Celui-ci requiert le service du type fourni et fournit un service du type requis pour chaque service du type fourni présent sur la plateforme.

5 CONTEXTUALISATION DE LA SELECTION DE SERVICES

La conscience du contexte est un aspect primordial pour les applications dans les environnements pervasifs. Les solutions proposées pour gérer les aspects contextuels (cf.

Chapitre II.5.3) montrent généralement un lien étroit des applications avec un serveur de contexte qui prêtent aux utilisateurs des informations de localisation, de posture, de préférences, voire d'activité et d'humeur, tandis qu'ils qualifient les entités pervasives avec des informations de localisation, de qualité de service, de niveau de sécurité. Les applications s'informent raisonnent sur les données de ce gestionnaire du contexte afin de définir la configuration la plus adéquate.

La configuration d'une application orientée service est définie par le choix des instances de services parmi les instances présentes dans l'environnement et par la qualité des liaisons établies avec ceux-ci. La configuration change par rupture et créations de liaisons. Les changements de configuration posent le problème de continuité de service. Ce thème est abordé dans la section 5.3.

La section courante présente les techniques du Home SOA qui répondent aux exigences techniques énumérées en section Chapitre IV.2.3 :

- La contextualisation du registre de services répond au besoin de qualification contextuelle des propriétés de services en découplant sources de contexte et fournisseurs de services [2007-3] (cf. section 5.1).
- La contextualisation du registre de services répond de même au besoin de qualification contextuelle des requêtes de services. Le même patron de conception découple sources de contexte et applications clientes ou demandeurs de service [2007-3] (cf. section 5.2).
- Des outils de sélection de services sont intégrés au modèle de d'auto-description des dépendances de services afin de permettre le classement dynamique de services dans l'automatisation de la composition de services [2007-3][2007-7][2006-2] (cf. section 5.3)
- Des pistes de travail sont indiquées pour la sujétion de la reconfiguration des applications à des politiques de (re-)liaison de services [2007-3][2007-7][2006-2] (cf. section 5.4).

5.1 Contextualisation du registre de services

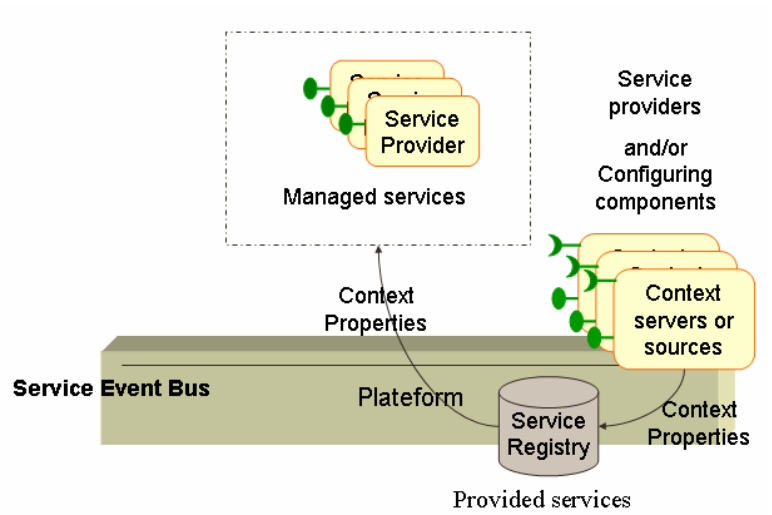


Figure 58 Contextualisation des fournisseurs de services

Les informations de contexte du fournisseur de service est acquis par lui-même ou de manière extérieure. Par exemple un centre multimédia pourra lister les codeurs-décodeurs (codecs) qu'il permet alors qu'un système extérieur, tel un système de gestion de localisation,

seul pourra renseigner la pièce qu'il occupe. Dans les deux cas, cette information doit être ajoutée dynamiquement à la description des services fournis.

Le registre de service est ici encore la pièce maitresse du découplage de la fourniture du service et l'attachement des informations de contexte. En effet, le fournisseur de service est simplement tenu de fournir un service générique supplémentaire à l'interface nommée ManagedService. Le Registre de Service contextualisé peut être érigé en patron de conception :

- **Problème** : un fournisseur de service est souhaité contextualisé par une application mais ce fournisseur est développé sans connaissance des informations de contexte pertinentes qui pourront être ajoutées à l'exécution.
- **Solution** (cf. Figure 58) : la solution consiste en l'enregistrement par le fournisseur de service d'une interface générique appelée "ManagedService". Toute source de contexte extérieure pourra ajouter, modifier ou supprimer des propriétés de contexte grâce à une méthode appelée updated(). Chaque fois que cette méthode est appelée avec un ensemble de propriétés, le fournisseur de services les applique en modifiant les propriétés du service enregistré.

Dans un modèle orienté service, il est naturel de concevoir aussi les sources de contexte comme des fournisseurs de service. Des composants gestionnaires de contexte sont responsables de requérir les informations de contexte et de souscrire à leurs changements auprès des sources de contexte de plus bas niveau. Une hiérarchie de gestionnaire de contexte peu se construire de manière paire à paire de la même façon que les applications (cf. section 2.3). Les gestionnaires de contexte de plus haut niveau se verront conférés les droits de peupler les services de type ManagedService avec des informations agrégées de contexte.

5.2 Contextualisation des requêtes de service

Afin que la sélection de services soit riche et non ambiguë, des techniques de sélection de services sont ajoutées au modèle de dépendances de services. Le modèle décrit un filtre de service obligatoire, des filtres de services secondaires et une fonction de classement de services dans un ordre total dans un langage de script (cf. Chapitre II.5.3).

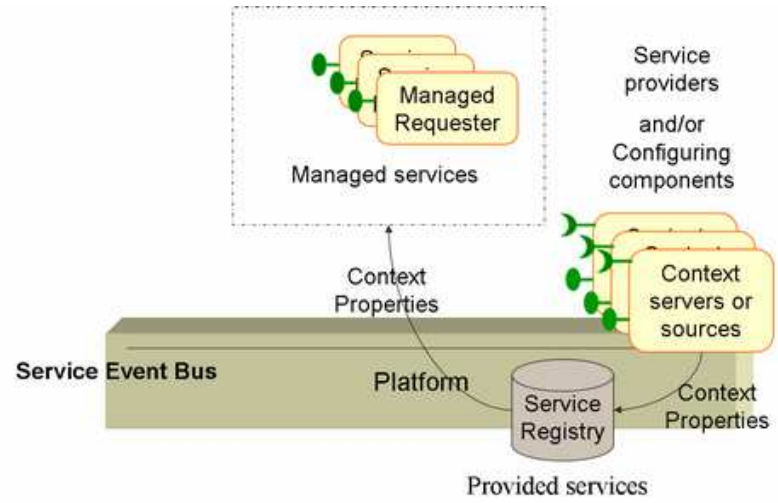


Figure 59 Contextualisation des requêtes déclarées de services

L'objectif de la section courante est d'ajouter des informations de contexte dans la précision des techniques de sélection de service des clients. Le patron de conception "Registre de Services contextualisés" est dérivé pour la contextualisation des requêtes de services. Le demandeur de services devient un fournisseur d'un service de type

"ManagedService" (cf. Figure 59) avec trois propriétés spécifiques : "filter", "optional-filters", "utility-function". Cette interface permet aux travers de sa méthode de mettre à jour le filtre obligatoire, les filtres secondaires et l'algorithme de classement des services dans un ordre total. Le demandeur de services est responsable de la répercussion des changements dans les propriétés enregistrées du service ManagedService. Le gestionnaire de dépendances du modèle de composition automatique doit prendre en compte les changements acceptés afin de modifier les mécanismes de liaison de services en conséquence.

L'algorithme interne de classement dynamique des services filtrés proposé dans les contributions [2007-7][2007-3][2006-2] voit sa limite dans le couplage des propriétés à celle d'un service de gestion de contexte. En effet, si la fonction d'utilité est définie à l'intérieur du composant client, les propriétés impliquées dans le classement doivent être connues durant la phase de développement. Si la fonction d'utilité est définie à l'extérieur, un mécanisme d'ajout de script extérieur permettra de mettre à jour l'algorithme et les informations de contexte impliquées au cours de l'exécution.

5.3 Classement dynamique de service à la composition

Les besoins des demandeurs de service sont projetés sur différentes techniques de sélection : un filtre obligatoire, des filtres optionnels pouvant être nombreux et classés du plus important au moins important, un algorithme de tri dans un ordre total (cf. section Chapitre II.5.3). L'algorithme de tri est défini dans un langage de script distinct du langage de programmation. Un changement dans le classement des services présent déclenche des actions de reliaison.

Dans le modèle proposé, un demandeur de service possède un gestionnaire de dépendances pour chaque dépendance de services déclarée (cf. section 2.3). Le gestionnaire de service maintient à jour la liste des services satisfaisant le filtre obligatoire, son partitionnement en plusieurs listes selon les filtres secondaires et l'ordonnancement selon un ordre total de la liste des services satisfaisant tous les filtres. Il est aussi responsable de la sélection du ou des services pertinents et de leur liaison par appel des méthodes déclarées dans le fichier de description.

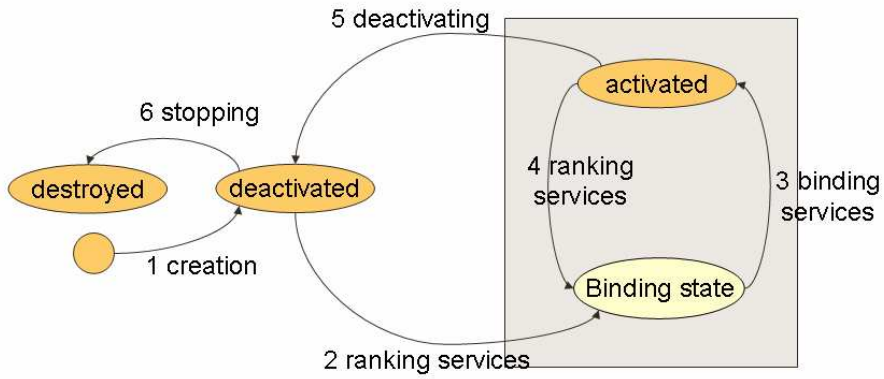
A chaque départ, à chaque arrivée, à chaque changement de propriété d'un service fourni et à chaque changement de propriété du service associé à la requête de service (cf. section 5.2) un événement de service est généré par le registre de service.

- Pour le départ d'un service, le gestionnaire supprime simplement le service de la liste maintenue.
- Pour l'arrivée d'un service et pour un changement de propriété d'un service, le gestionnaire évalue les filtres et si le service les satisfait réévalue la place du service dans la liste.
- Pour un changement des filtres, le gestionnaire réévalue la satisfaction des services de la liste, et reclasse les services les satisfaisants si certains de ces services n'étaient pas déjà classés.
- Pour un changement de l'algorithme de classement, le gestionnaire de services reclasse les services satisfaisant les filtres.

Dans tous les cas, le gestionnaire délie le ou les services n'étant plus classés en tête et lie les services les remplaçant. Le gestionnaire doit maintenir un nombre de liaisons inférieur ou égal au maximum indiquée dans la description de la dépendance. Les filtres optionnels et l'algorithme de tri n'ont un sens que dans le cas d'une cardinalité limitée de liaisons (c'est-à dire une cardinalité 0..x ou 1..x où x appartient à $]0,\infty[$). Le coût de reliaison doit faire partie

de l'algorithme de classement afin de garantir que l'avantage de la reconfiguration est supérieur à ce coût.

Le cycle de vie proposé dans la Figure 60 pour les composants à service comprend la phase de classement dynamique des services disponibles. Premièrement, le gestionnaire instancie l'objet représentant le composant et celui-ci est placé dans l'état "deactivated" (étape "1 creation"). Les services sont alors filtrés et si les liaisons obligatoires peuvent être satisfaites, les services sont classés et le composant entre dans l'état "binding" (étape "2 ranking services"). Les meilleures instances de services sont liées au composant par appel des méthodes de liaison décrites par le composant dans les limites indiquées de cardinalité (étape "3 binding services"). Si une méthode d'activation est indiquée, elle est appelée avant que le composant ne soit considéré dans l'état "activated". Si un événement de départ, d'arrivée, de changement de propriété affecte la liste de services, le filtre ou l'algorithme de classement : si la politique du composant est indiquée "dynamique" et les liaisons obligatoires peuvent encore être satisfaites, les services sont alors à nouveau classées et le composant entre à nouveau dans l'état "binding" (étape "4 ranking services"); si la politique et "dynamique" et les liaisons obligatoires ne peuvent pas être satisfaites ou si la politique est indiquée "statique" et si une à plusieurs services liés disparaissent, le composant vient alors dans l'état "deactivated" (étape "5 deactivating"). Si et seulement si la politique est "statique", le composant entrant dans l'état deactivating est abandonné et entre dans l'état "destroyed" (étape "6 stopping") pendant qu'un autre composant est instancié et entre dans l'état "deactivated".



1. Creation
2. Ranking available filtered services with optional filters and ranking method
3. (Un,Re-) binding best services
4. Service ranking on following events: Bound service leaving or new service registered or requirements changed
5. Deactivation on following events: Mandatory service missing leading or stopping.
6. Stopping

Figure 60 Cycle de vie des et classement de services contextuel

5.4 Transfert d'état automatique

Il est souhaitable de raffiner l'ensemble des actions de dé-liaison et de reliaison afin de garantir un niveau de continuité de service. Ces taches dépendent souvent spécifiquement de l'application donnée, ce qui rend difficile la tentative d'automatiser les réactions de liaison. Les actions de continuité de service sont le plus souvent l'objet d'une optimisation spécifique.

Le transfert d'état est un problème dur et le Home SOA ne propose que quelques règles de conduite pour la garantie de continuité de service. Premièrement, si la continuité de service ne peut être assurée dans des conditions acceptables au cours de reliaisons de service, il est possible d'empêcher les opérations de reliaison si les services liés demeurent présents en spécifiant la politique de composant comme "statique" (cf. section 5.3). Deuxièmement, le coût de la reliaison peut être intégré dans l'algorithme de tri, un avantage plus ou moins important peut affecter le service lié dans une dépendance critique. Enfin, dans quelques situations, la politique de reliaison peut être affinée.

Les articles [2007-3][2006-2] montrent une politique de reliaison automatisée dans une situation particulière. La situation est celle d'une dépendance de service singulière vers un service tel qu'un serveur de restitution UPnP (par exemple, une télévision). Une politique spéciale est définie dans le modèle de gestion automatique de composants. Cette politique peut être nommée "superposition". Dans ce cas, dès qu'un fournisseur de service vient à être mieux classé que le fournisseur lié, il est lié avant que le service préalablement lié soit délié après une durée configurable. Cela permet au nouveau service d'être configuré avec les propriétés caractérisant l'état du service courant. Une superposition des services en cours dure le temps configuré. Dans le scénario de "Suivez-Moi" audio-visuel, la superposition est appréciée par l'utilisateur qui ne perd pas une seconde du flux audio-vidéo en cours.

6 UPnP Device Management : une reponse protocolaire

Toutes les situations de réseaux hétérogènes ne peuvent être résolues par l'approche logicielle Home SOA. Il demeure des situations où des protocoles manquent pour le contrôle d'applications d'un domaine particulier. Le domaine de la gestion de modules logiciels sur les plateformes d'exécution d'un réseau local présente ce symptôme. Dans cette situation, la ligne de conduite de cette thèse (cf. section 1) propose la conception d'un nouveau protocole pivot qui permette à une application d'administrer de manière uniforme les différentes plateformes d'exécution existantes.

Cette section présente la proposition technique de nouvelles interfaces de services dans le Forum UPnP [2007-9]. Début 2007, le besoin d'administration d'équipements domestiques est grand alors que les protocoles manquent et que les plateformes de déploiement modulaires d'applications se répandent. Ces interfaces sont conçues au-dessus d'un protocole répandu sur le LAN pour implémenter :

- des outils d'auto-entretien (self-care) pour le réseau domestique
- un pont avec les protocoles d'administration distante tels TR-069 (cf. Figure 61)

Figure 61 Administration de bout en bout et UPnP Device Management

L'objectif du Comité de Travail UPnP Device Management est de définir une interface générale d'administration pour les équipements de réseaux locaux, tels que le réseau domestique en particulier. Plusieurs travaux techniques en découlent :

- Spécifier premièrement une interface de configuration et un modèle de données hiérarchique et, deuxièmement, spécifier une interface générale de diagnostique et de supervision de performances. Ce premier travail technique est déjà effectué par d'autres protocoles d'administration existants (cf. Chapitre III.3.2). La seule difficulté réside en l'application de cette interface sur les protocoles UPnP.
- Spécifier une interface de gestion de cycle de vie logiciel qui s'adapte à toutes les plateformes d'exécution logicielles. Ce deuxième travail est plus ambitieux car les technologies sont nombreuses et variées et peu de travaux adressent cette complexité. SCOMO [SCOM08] tente cette généralisation aujourd'hui. Des rapprochements entre plateformes existent : OSGi et .NET [EDH06], OSGi et Linux [Roy07]. Le travail effectué dans le Forum UPnP innove en visant la gestion protocolaire agnostique de plateformes modulaires et en apportant une solution portant des caractéristiques en avance sur celles de SCOMO.

La description ci-dessous correspond à la proposition technique de France Telecom envoyée par l'auteur avec la collaboration de Levent Gürgen et Maxime Vincent le 21 juillet 2008[1]. Cette proposition d'interface UPnP nommée SoftwareManagement Service découle des scénarios, exigences techniques et discussions au sein du Comité auxquels l'équipe a fortement contribué. L'analyse approfondie des différentes plateformes et la proposition quasi-complète de spécification technique sont toutefois délivrées à un moment précoce dans la phase de spécification du Comité. Par conséquent, la proposition subira certainement des modifications avant d'être acceptée comme base de spécification. Les discussions et les modifications auront lieu jusqu'au début de l'année 2009.

6.1 Entités logicielles et diagrammes d'états

Afin d'exprimer la généralité des concepts de gestion de cycle de vie logiciel, les trois entités définies dans la section Chapitre III.3.1.5 sont reprises dans la proposition :

- Paquetage de déploiement (DP pour Delivery Package) : une unité binaire empaquetant une à plusieurs unités de déploiement permettant généralement l'installation cohérente d'une application complète. Un paquetage peut être téléchargé et retiré de la plateforme. Une opération d'installation permet d'installer le contenu d'un DP téléchargé. Le DP peut être supprimé sans désinstaller les DUs associés.
- Unité de déploiement (DU pour Deployment Unit) : une unité binaire qui peut être déployée dans l'environnement – installée, désinstallée, mise à jour. Elles apparaissent après installation d'un DP ou par installation directe à partir d'une adresse locale ou extérieure à la plateforme (cf. Figure 62).
- Unité d'exécution (EU pour Execution Unit) : une unité logique qui peut être démarrée ou arrêtée. Les unités d'exécution apparaissent avec l'entrée de DUs dans l'état Resolved. La Figure 63 l'illustre avec la transition implicite entre l'état "Installed" du DU vers l'état "Inactive" de l'EU. A l'opposé, lorsqu'un DU est désinstallé, tous les EUs contenus sont stoppés et supprimés. Cette transition

[1]Email publié le 21 juillet 2008 sur la liste du Comité

implicite est visible dans la figure entre l'état "Inactive" de l'EU vers l'état "Uninstalled" du DU.

Les caractéristiques précises de ce modèle d'entités sont énumérées ci-dessous selon les critères définis dans la section Chapitre III.3.1.5 :

- Dépendances : un DP peut contenir un ou plusieurs DUs, un DU peut contenir un ou plusieurs EUs. Les dépendances sont à indiquer dans les paramètres de chaque entité définie dans le modèle de description logicielle de la plateforme (cf. section 6.6). Les dépendances entre unités d'un même type peuvent aussi être répertoriées dans la description des unités mais le format est laissé à définir par chaque technologie implémentant l'interface UPnP.
- Métadonnées : l'extensibilité du format hiérarchique de données permet la libre définition des données décrivant les entités logicielles par les technologies implémentant cette interface (cf. section 6.6).

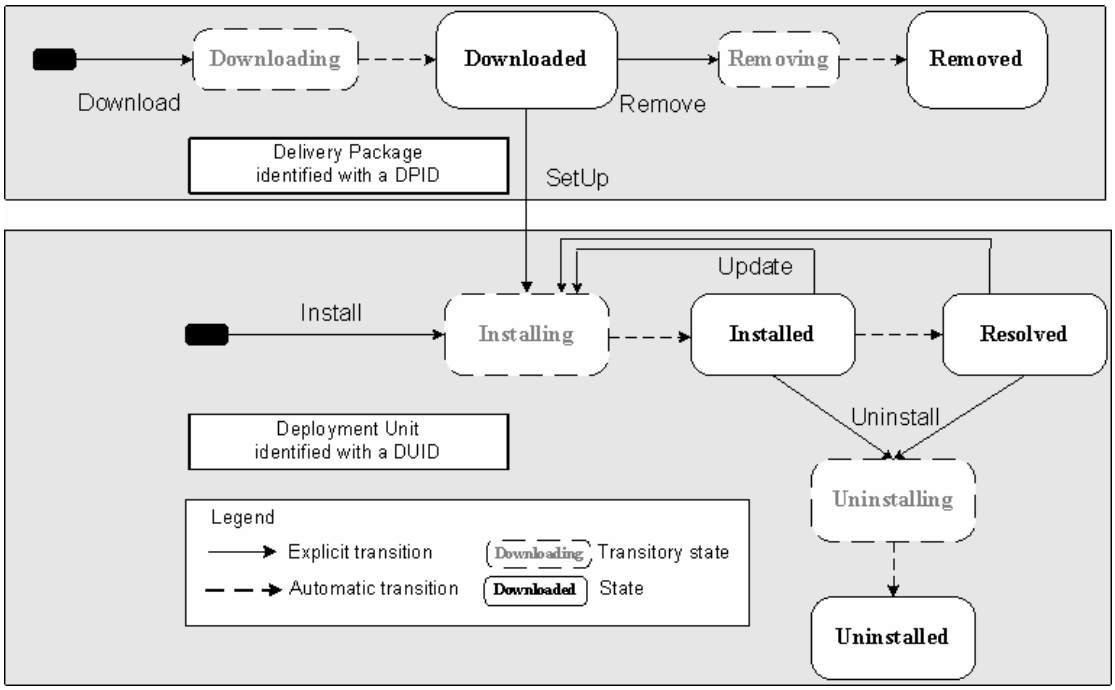


Figure 62 Diagrammes d'état des paquetages et des unités de déploiement

- Opérations de gestion de cycle de vie (cf. Figure 62 et Figure 63). Le détail des actions UPnP définies est indiqué dans la section Chapitre VI.7.
  - des paquetages de déploiement : Download(), Remove(), SetUp().
  - des unités de déploiement : Install(), Uninstall(), Update().
  - des unités d'exécution : Start(), Stop().
- Etats (cf. Figure 62 et Figure 63)
  - des paquetages de déploiement : 2 états stables : Downloaded, Removed; 2 états transitoires : Downloading, Removing.
  - des unités de déploiement : 3 états stables : Installed, Resolved, Uninstalled; 2 états transitoires : Installing, Uninstalling. Les plateformes qui permettent le diagnostique de résolution de dépendances indiquent la satisfaction des dépendances de chaque unité par l'état Resolved.

- des unités d'exécution : 2 états stables : Inactive, Active; 2 états transitoires : Starting, Stopping.

- Evénements : ils sont conséquents aux changements d'états. Le résultat des opérations de gestion logicielle sont notifiés à tous les clients observateurs de l'état du service défini par cette interface UPnP. La notification dans le modèle UPnP est détaillée en section Chapitre VI.7.

- Répertoires locaux et distants d'entités logicielles : toutes les entités sont répertoriées et décrites dans le modèle de données hiérarchique défini (cf. section 6.6). En revanche, les répertoires distants, dont sont issues les entités, ne sont pas représentés dans l'interface proposée.

Les actions Download() et Install() sont les seules actions qui prennent en argument d'entrée une adresse (de type URI) localisant une unité à l'extérieur de la plateforme. L'adresse indique l'emplacement de l'entité logicielle à télécharger ou à installer. Le type de l'entité dépend de la plateforme d'exécution.

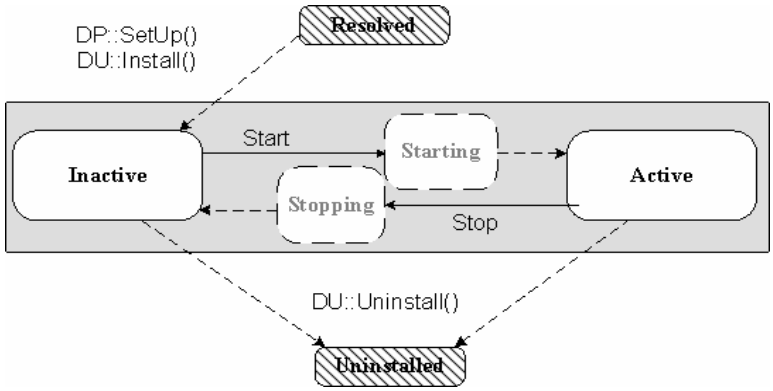


Figure 63 Diagramme d'état des unités d'exécution

L'interface protocolaire est définie pour être implémentée de manière distincte par chaque plateforme logicielle. Bien qu'il soit possible que la même implémentation puisse permettre le déploiement de différents types d'unités sur des plateformes aux technologies différentes, il est recommandé d'attacher une implémentation différente de l'interface pour chaque plateforme installée sur un équipement. Par exemple, une passerelle domestique acceptant la mise à jour de son micrologiciel (firmware) et le déploiement de bundles sur une plateforme OSGi pourra fournir deux services UPnP du type SoftwareManagement. Chacun sera qualifié par une propriété indiquant la technologie de la plateforme. L'implémentation pour la technologie OSGi acceptera le bundle en tant qu'unité de déploiement tandis que l'autre acceptera le firmware adéquat. Le premier implémentera toutes les actions alors que la deuxième pourra n'implémenter que les actions Download(), Remove(), Update().

6.2 Opérations asynchrones et gestion des événements

Les opérations de gestion logicielle sont effectuées sur des durées variant d'une fraction de seconde à plusieurs dizaines de secondes. Des minutes peuvent parfois être consacrées par exemple dans des téléchargements avec une bande passante faible ou des démarrages de serveurs d'application sur des passerelles contraintes. En raison de la longueur de ces durées, des opérations doit être prioritairement gérées de manière asynchrone. Dans la proposition, elles le sont de manière exclusive pour une raison de simplicité.

L'asynchronisme dans un protocole de contrôle client-serveur tel qu'UPnP signifie que l'opération est initiée par une action (ou méthode) au retour quasi-immédiat et s'achève sur l'envoi du résultat par notification. Une distinction est donc spécifiée ici entre action et

opération, action étant le terme normalisé par UPnP. Le retour quasi-immédiat de l'action indique au client que l'action l'opération est débutée, l'événement notifié indique aux clients intéressés le résultat final des modifications initiées.

6.3 Résolution des dépendances sur la plateforme

Plusieurs technologies – par exemple, Linux Debian, .NET, OSGi – modélisent les dépendances entre unités de déploiement et sont capables d'analyser la résolution de celles-ci pour chaque unité. Ce diagnostique de résolution est laissé ici à la plateforme elle-même. L'action passant un DU de l'état Installed à l'état Resolved est spécifiée implicite dans la Figure 62. Tant que la plateforme ne déclare pas un DU dans l'état Resolved, il ne peut pas être utilisé dans l'exécution d'applications. Les EUs apparaissent par conséquent lorsque le DU vient dans l'état Resolved[1].

6.4 Gestion partagée des dépendances

Les entités logicielles impliquées dans une opération peuvent être nombreuses. En effet, deux types de dépendances sont définis entre les entités :

- Les dépendances de type inclusif : un DP peut contenir plusieurs DUs, un DU peut contenir plusieurs EUs.
- Les dépendances de type interactif, c'est-à-dire les dépendances entre entités de même niveau : la partie applicative installée par un DU peut dépendre d'un autre DU. Par conséquent, les DPs peuvent aussi définir des dépendances entre eux. L'exécution d'EUs peut aussi dépendre de l'exécution d'autres unités. Une opération sur une entité peut entrainer les mêmes instructions sur les entités dont elle dépend si les dépendances sont gérées.

La gestion des dépendances inclusives est nécessairement gérée par la plateforme elle-même. En revanche, la gestion des dépendances interactives est plus complexe et dépend de l'effectivité de la description des différentes entités et des capacités de la plateforme sous-jacente. Il est souhaitable que la plateforme puisse gérer elle-même ses dépendances. Mais certaines situations montrent que la gestion des dépendances est préférable sur le client gestionnaire. Ces situations peuvent être le manque de ressources de la plateforme controllée ou la mauvaise description des dépendances entre entités que seul le client gestionnaire sait corriger par l'installation d'unités individuelles. Dans la section Chapitre VI.7, un argument booléen d'entrée dans chaque action permet au point de contrôle d'indiquer s'il est souhaité que la plateforme controllée gère elle-même ces dernières dépendances.

6.5 Gestion des dépendances et aspects transactionnels

Plusieurs entités dépendantes peuvent être impliquées dans une opération de gestion logicielle. En supplément des dépendances naturelles entre un DP et ses DUs contenus, entre un DU et ses EUs, il peut exister des dépendances entre entités de même type. Plusieurs entités peuvent donc être impliquées dans chaque opération de gestion logicielle :

- Les opérations de téléchargement et de retrait d'un DP peuvent impliquer le téléchargement et le retrait de DPs dont il dépend et qui ne sont pas déjà respectivement présents ou utilisés sur la plateforme. L'installation de DP (SetUp) entraine aussi l'installation des DUs contenus.

[1] L'état Resolved et ses effets diffèrent toutefois dans la contribution du 21 juillet 2008.

- Les opérations d'installation, de mise à jour et de désinstallation d'un DU peuvent impliquer les DUs dont il dépend. Elles entrainent aussi l'apparition, l'éventuel arrêt et la disparition des EUs contenus par les DUs impliqués. L'outil Aptitude des distributions Linux Debian est un exemple d'outil qui gère les dépendances entre packages durant ces opérations.
- Les opérations de démarrage et d'arrêt d'un EU peuvent impliquer le démarrage et l'arrêt des EUs dont il dépend et qui ne sont respectivement pas démarrés ou utilisés sur la plateforme. Sur Windows XP par exemple, le démarrage d'un service Windows entraine le démarrage des services dont il dépend.

Chaque opération est gérée en tant que transaction atomique. L'atomicité est visible dans la spécification de l'événement notifié à la fin de l'opération. Premièrement, un seul événement est notifié afin de caractériser l'unicité de l'opération malgré la multiplicité des instructions impliquée par la gestion de dépendances. Deuxièmement, le résultat est géré un succès total ou un échec complet. Soit chaque entité impliquée est dans l'état visé pour celle-ci (commit), soit toutes les entités sont retournées dans l'état initial à l'opération (rollback).

Afin que les clients (éventuellement nombreux) connaissent les unités impliquées dans une opération, chaque entité impliquée entre dans l'état transitoire correspondant entre état initial et état visé. Idéalement, au retour de l'action initiale, toutes les entités impliquées sont dans un état transitoire. Cet état agit tel un verrou sur les ressources impliquées. Toute requête sur une entité dans un état transitoire reçoit une réponse d'erreur. Les états transitoires sont indiqués dans les Figure 62 et Figure 63 : Downloading, Removing, Installing, Uninstalling, Starting, Stopping.

6.6 Représentation du modèle d'entités logicielles

Le modèle de description des plateformes logicielles spécifique à chaque plateforme et les technologies existantes le définissent extensible de surcroit. C'est pourquoi la proposition discute la possibilité d'utiliser le modèle de données hiérarchique utilisé par le service de configuration nommé ConfigurationManagement en cours de spécification. Dans ce modèle, un tableau d'objets sera affecté à la description des entités de chaque type. Les objets seront identifiés par l'identifiant généré par la plateforme pour chaque entité. Les paramètres de ces objets seront la liste des paires attributs-valeurs associée par chaque technologie à leurs entités.

7 SYNTHESE

Ces travaux de thèse établissent une ligne de conduite dans les situations de gestion de l'hétérogénéité des technologies sur les réseaux locaux. Cette ligne de conduite a conduit à une action de normalisation vis-à-vis de la coexistence des technologies UPnP et IGRS sur les réseaux locaux. Elle a mené principalement à la proposition d'un cadriciel nommé Home SOA, un ensemble de techniques au-dessus de plateformes modulaires comme les plateformes OSGi et .NET. Ces travaux au-dessus de plateformes modulaires et de protocoles Plug-n-Play ont entrainé enfin la création du Comité de Travail UPnP Device Management et la participation à l'élaboration de scénarios, d'exigences techniques et d'une proposition technique pour un protocole générique de gestion du cycle de vie de modules logiciels.

Les techniques du Home SOA permettent de ramener la conception de toute application pervasive à une conception localisée sur une seule plateforme d'exécution où la gestion de la distribution, de l'hétérogénéité et de la dynamique sont simplifiées. Les applications visées sont les applications composées de multiples équipements distribués

hétérogènes évoluant dans un contexte d'utilisation dynamique. Des patrons de conception implémentés dans le cadriciel éponyme simplifient, voire rendent transparent la gestion de ces aspects complexes.

Au-dessus d'une plateforme de services, un premier niveau de pilotes orientés services réifient les entités pervasives de l'environnement sous forme de services locaux. Les détails protocolaires concernant la découverte, le contrôle et la notification d'événements des équipements sont invisibles pour le développeur d'application.

Chaque pilote orienté service est spécifique d'un protocole. Un deuxième niveau de pilotes raffinés est responsable de la réification des objets mandataires du premier niveau en services implémentant des interfaces associées à un domaine d'application. La chaine de réification de ces pilotes est dynamique, la présence des objets réifiés dans le registre de services correspond à la présence des entités pervasives dans l'environnement.

Grâce à ces pilotes, les aspects distribués et l'hétérogénéité du réseau disparaissent tandis que la dynamique du réseau est reflétée sur la plateforme par la population dynamique du registre de service par des objets mandataires et adapteurs représentatifs. La dynamique est exacerbée par la nécessaire prise en compte du contexte de l'utilisateur, des équipements et du reste de l'environnement. Encore une fois, le registre des services est la clef de techniques de contextualisation dans un modèle où sources de contexte et composants applicatifs sont faiblement couplés. Enfin, un modèle à composants orienté services allié à un conteneur d'automatisation de composition et des techniques riches de sélection de services simplifie alors la gestion de la dynamique.

Le registre de services est le réceptacle de toutes les entités pervasives présentes sur le réseau. Le développeur construit idéalement ses applications en composant statiquement des services apparemment locaux aux interfaces à la sémantique naturelle.

La nouveauté des interfaces protocolaires proposées par le Comité de Travail UPnP Device Management réside en la description d'un modèle générique de modules logiciels. La proposition de France Télécom écrite par l'auteur avec le soutien de Levent Gürgen et de Maxime Vincent analyse différentes plateformes de services (cf. Chapitre III.3.2) et retient trois types d'entités : le paquetage de déploiement (Delivery Package ou DP), l'unité de déploiement (Deployment Unit ou DU), l'unité d'exécution (Execution Unit ou EU). Les opérations de gestion logicielle distinguent les opérations de déploiement sur les unités binaires que sont les deux premières entités, et les opérations d'exécution sur les unités d'exécution, unités logiques pouvant être démarrées et arrêtées. La gestion des dépendances peut être effectuée par le client comme par le serveur. La résolution des dépendances entre unités de déploiement est reflétée par un état dans le diagramme d'états de ces unités. La description des unités demeurent extensible grâce à l'association d'un modèle hiérarchique de description des entités logicielles. Enfin, l'atomicité et l'isolation des opérations de gestion sont appliquées avec les techniques de spécification UPnP et l'usage d'états transitoires dans les diagrammes d'états.

Chapitre VI. EXPERIMENTATIONS

La flexibilité des techniques du Home SOA est complémentaire de l'utilisation d'une plateforme de déploiement modulaire d'applications telles qu'OSGi ou .NET. La technologie OSGi a été choisie dans la plupart des expérimentations de cette thèse. Les atouts généraux de la plateforme et la pertinence de certains chapitres additionnels de la spécification OSGi sont décrits dans la section 1. Cette dernière présente aussi la façon dont ces travaux de thèse ont étendus certains mécanismes afin de les utiliser dans le cadre d'applications pervasives. Les détails d'implémentation de la proposition particulière de la répartition de services sur plusieurs plateformes sont alors indiqués en section 2. La section 3 présente l'implémentation du DPWS Base Driver. La section 4 détaille les différentes implémentations de pilotes raffinés. La section 5 offre un descriptif de l'implémentation de l'application de communication personnelle enrichie.

Alors que le cadriciel Home SOA répond aux besoins du contrôle d'équipements distribués et hétérogènes dans un environnement dynamique, les sections 6 et 7 détaillent les réponses de ces travaux de thèse dans des situations différentes : la première est la réponse aux problèmes de coexistence de deux protocoles – UPnP et IGRS – sur un même réseau local. La deuxième est le travail de spécification d'un protocole d'administration d'équipements aux modèles de gestion de cycle de vie hétérogènes.

1 COMPOSITION DE SERVICES SUR LA PLATEFORME OSGI

1.1 Le choix de la technologie OSGi

La technologie OSGi [OSGi05] a été choisie pour l'expérimentation de l'approche Home SOA en raison de ses avantages techniques :

- Modularité et chargement de classes dynamique répondent aux exigences de couplage lâche et de chargement logiciel à la demande exprimées face au défi d'hétérogénéité (Chapitre II.4.3.1) des environnements pervasifs. La modularité du déploiement OSGi offre une délimitation naturelle des parties applicatives pertinentes pour la répartition post-développement (cf. Chapitre II.3.3).
- Le modèle de programmation orienté service parfait la réponse à l'exigence de couplage lâche et offre une base de travail simple pour bâtir une solution de sélection de services dynamique (cf. Chapitre II.5.4)..
- La simplicité du langage de programmation associé, Java, répond au besoin de prototypage rapide et d'évolutivité face à l'ouverture des réseaux pervasifs.

Aussi, le choix se justifie industriellement sur les équipements de réseaux locaux par plusieurs raisons anticipant un déploiement à un terme plus ou moins long selon l'équipement et le réseau visé.

1.1.1 Avantages techniques

Son avantage technique principal est son modèle de développement et de déploiement modulaire par le partage et séparation de code dans des modules applicatifs distincts (bundles). Il permet la structuration des applications en modules applicatifs aux dépendances à grain fin clairement définies par des notions du langage de programmation sous-jacent : le package et la classe Java. Cette correspondance avec des notions de langage permet d'implémenter des garde-fous naturels pour le partage et l'isolation. Les limites de partage sont vérifiées par des politiques données aux chargeurs de classes (classloaders). Chaque bundle possède son propre chargeur de classes. Celui-ci charge les classes déclarées privées et exportées, et référence le chargeur de classes élu par la plateforme sous-jacente pour les packages déclarés importés. Cette flexibilité de partage et d'isolation de parties de code entre entités logicielles semblent uniques parmi les plateformes logicielles existantes. La section Chapitre III.3.3 présente comment les modèles .NET, Linux et MIDP3 n'offrent pas de telles propriétés.

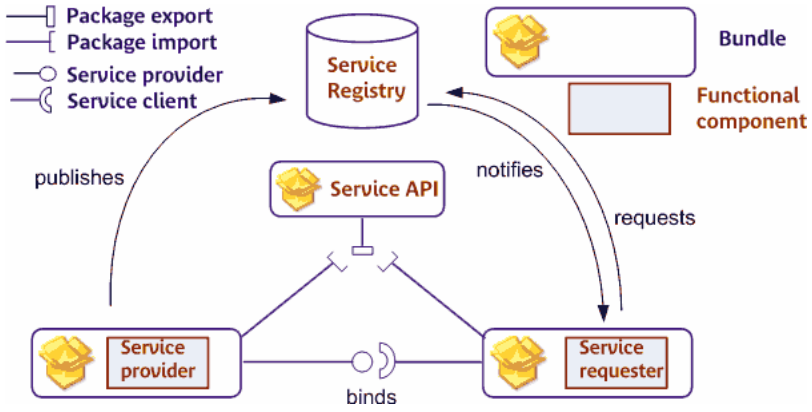


Figure 64 Patron de conception SOA sur la plateforme OSGi

Son second atout, qui peut paraître indépendant à première vue, est le modèle de collaboration à base de service entre les modules applicatifs. La technologie OSGi donne les fondements d'une programmation orientée services. Le standard OSGi applique techniquement le concept de plateforme de services décrit en Chapitre V.2.2. Elle offre le registre de services auprès duquel tout objet peut requérir, enregistrer et retirer des services. Un système de souscription et de notification d'événements lui est associé afin que tout objet puisse être notifié des départs, arrivées et mises à jour des services dans le registre. Home SOA exploite ces fondements en recommandant le suivi de patrons de conception avancés. Quelques autres cadriciels comme Avalon de la communauté Apache (avalon.apache.org) étaient basés sur le patron "Dynamic Service Locator" au moment où l'Alliance OSGi délivrait sa première version de spécification. Mais ces autres tentatives n'étaient pas associées à des mécanismes de déploiement modulaires. La programmation orientée service d'OSGi est en effet liée au partage et l'isolation de code entre bundles puisque ces mécanismes modulaires permettent

- d'une part, aux clients et aux serveurs de partager les mêmes interfaces sans que les clients et serveurs ne partagent d'autres parties de code (cf. Figure 64)
- d'autre part, aux clients de partager la même instance de service sans qu'aucun d'entre eux ne puissent introspecter le code de chacun des autres.

Le chargement de classe Java permet de plus le déploiement de code 'à chaud', utile pour l'installation, la désinstallation, la mise à jour des bundles sans arrêt de fonctionnement des applications de la plateforme (sauf, évidemment, celles qui dépendent de manière obligatoire du module donné). Il est notable que .NET et Java MIDP(3) ne permettent pas cette flexibilité sans pertes importantes sur les propriétés de partage et d'isolation.

1.1.2 Avantages industriels

La technologie OSGi possède des attraits industriels notables. Les serveurs d'applications les plus importants l'ont déjà adopté et d'autres segments montrent quelques succès. Les réseaux mobiles, domestiques et véhiculaires montrent beaucoup de prototypes et de promesses à plus ou moins long terme. Voici les avantages industriels de la technologie :

- **son état de standard** : OSGi est une technologie poussée par un large consortium presque sans concurrent au-dessus de la technologie Java. Au dessus d'autres langages de programmation, d'autres technologies existent mais n'offrent pas tous avantages techniques d'OSGi. Ceux qui s'en approchent sont de surcroit faibles (Microsoft .NET) ou fragmentés (distributions Linux).
- **l'adoption du standard OSGi** : la technologie est présente sur des serveurs d'application (IBM Websphere Suite, Jonas 5, Redhat JBoss, Apache Geronimo…), sur le marché SOHO (Imprimantes Canon et Ricoh depuis 2003) et dans l'automobile (BMW depuis 2004), mais demeure balbutiante sur le marché du mobile (applications Nokia à venir) et sur le marché domestique. La technologie OSGi suscite aujourd'hui un intérêt parmi les acteurs de ce dernier marché : NTT, Deutsch Telekom, Telefònica, Samsung, Prosyst, Makewave ont fondé en 2007 le Residential Expert Group avec l'objectif de spécifier une architecture logicielle de référence pour l'administration du réseau domestique.
- **l'ouverture du standard OSGi** : l'utilisation du standard est libre. Les spécifications ont été rendues disponibles sous 4 versions successives de 1999 à octobre 2005. Une 5ème version est attendue pour l'année 2009. Les plus grands acteurs de la spécification OSGi ont fait une promesse de non-assertion. La propriété intellectuelle est donc a priori délivrée gratuitement. Trois implémentations répandues de la plateforme sont développées et maintenues par des communautés open source importantes : eclipse equinox est soutenue par IBM, Knopflerfish est soutenue par Makewave, et felix est un projet très actif de la grande communauté Apache.
- **le nombre de briques (bundles) OSGi existantes** : les développements OSGi profitent d'un formidable réservoir de bundles disponibles dû à l'abondance des applications Java et à la maturité du standard tiré par d'importants acteurs (IBM, Nokia, Siemens, etc.) depuis 1999. Le cœur d'OSGi étant une couche intergicielle à l'API simple et d'implémentation légère (par exemple, l'implémentation open source Apache felix représente 300 KB) permettant la modularité au-dessus de Java, toute application Java peut être l'objet d'une modularisation au-dessus de la plateforme OSGi. L'évolution de la spécification OSGi a de plus construit un catalogue de nombreuses spécifications de services applicatifs pouvant fonctionner au-dessus de la plateforme.

1.1.3 Les freins à son adoption

Les freins à son adoption pour sur les réseaux locaux sont aussi d'ordre technique et ont un impact industriel : son principal défaut est d'être associé à la technologie Java, en particulier Java CDC. En effet, cette technologie est coûteuse sur les équipements embarqués. Ce coût provient du besoin en mémoire et en puissance de calcul de la machine virtuelle Java. Ce besoin est plus grand que celui de programme en langage C pour les applications

embarquées. De surcroit, il est plus grand que celui de programmes développés dans la technologie Java CLDC qui vise des environnements embarqués. La principale différence entre les deux plateformes Java concerne les chargeurs de classes qui ne sont disponibles que sur CDC et qui sont le fondement de la modularité d'OSGi. Ce coût en ressources de la plateforme Java sous-jacente semble plus grand que l'avantage d'évolutivité apporté par OSGi dans la plupart des applications embarquées sur les équipements des réseaux locaux.

L'impact industriel est visible : Java n'a de succès véritable que sur les serveurs d'applications, sur les navigateurs Internet et sur les équipements importants des réseaux d'entreprises (imprimantes). Son adoption dans les véhicules est marginale; seul BMW équipe ses voitures avec la technologie Java/OSGi. L'adoption de Java CDC par une plateforme avancée dans le réseau domestique est récente : les lecteurs DVD Blu-ray Sony. La situation du réseau mobile est particulière : Java est déjà adopté sous sa forme CLDC. CDC apporte une véritable rupture technique, le marché ne l'acceptera pas à court terme.

1.2 Automatisation des liaisons de services

Les travaux sur la dynamique du réseau domestique tentent de raffiner les approches du laboratoire Adèle, Service Binder [CeH03] et aujourd'hui iPojo [EHL07] par addition de techniques de sélection dynamique de services. Après un travail de catégorisation du contexte [2006-1], un premier prototype [2007-3] est construit au-dessus du cadriciel OSGi Declarative Services [OSGi05] qui est la normalisation de l'approche appelée Service Binder. Les aspects contextuels sont mieux séparés par l'usage de techniques à POJO (Plain Old Java Object) dans un deuxième prototype [2007-7].

1.2.1 OSGi Declarative Services

Le chapitre "Declarative Services" de la spécification OSGi spécifie un modèle de composants à service et un conteneur automatisant la composition de services de manière paire-à-paire au niveau de chaque composant tel qu'ils sont décrits dans la section Chapitre V.2.3. Cette spécification est la normalisation des concepts développés dans le projet Service Binder [CeH03]. Le conteneur appelé SCR (Service Component Runtime) affecte un gestionnaire de composition à chaque composant décrit dans un fichier de description du bundle (3 composants C et fichier metadata.xml sont indiqués dans le bundle de la Figure 65). Ce gestionnaire utilise les méthodes de requête, d'écoute d'événements et d'enregistrement de service du registre OSGi afin de satisfaire les dépendances et l'offre de services décrites dans les composants. Chaque dépendance de service est décrite par :

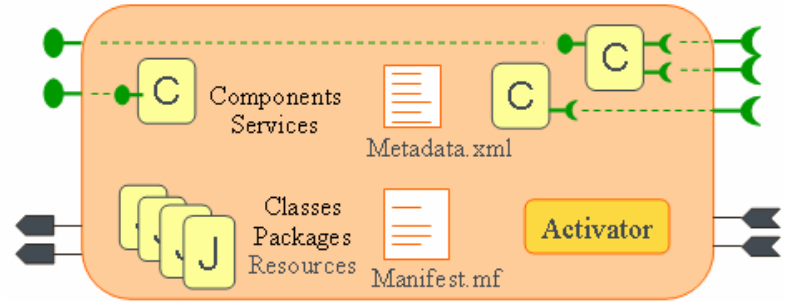


Figure 65 Bundle OSGi et composants à service

- Le nom d'une interface Java, dite de service
- Le filtre de service, filtre LDAP d'assertions booléennes attribut-opération-valeur
- Le nom des méthodes de liaison et de dé-liaison à appeler sur l'objet-composant

- La cardinalité de la liaison
  - 0..1 – Optionelle et singulière.
  - 1..1 – Obligatoire et singulière.
  - 0..n – Optionelle et multiple.
  - 1..n – Obligatoire et multiple.
- La politique statique ou dynamique de la liaison

Le cycle de vie est simplement défini par le diagramme d'état de la Figure 66. Le gestionnaire instancie chaque objet indiqué comme point d'entrée du composant dans le fichier de description. Pour chacun de ces objets, il attache un gestionnaire de dépendances pour chaque dépendance. Ces gestionnaires lie et délie les services recherchés. Dès que les dépendances obligatoires sont satisfaites, le gestionnaire de composant appelle la méthode d'activation. Si la politique est dynamique, les services peuvent être lié et délié dans l'état validate. Avec cette politique, le composant n'est invalidé que si une liaison obligatoire vient à n'être satisfaite par aucun service. Si la politique est statique, tout départ d'un service lié invalide l'instance qui est alors abandonnée ("destroyed") pour l'instanciation d'un autre objet.

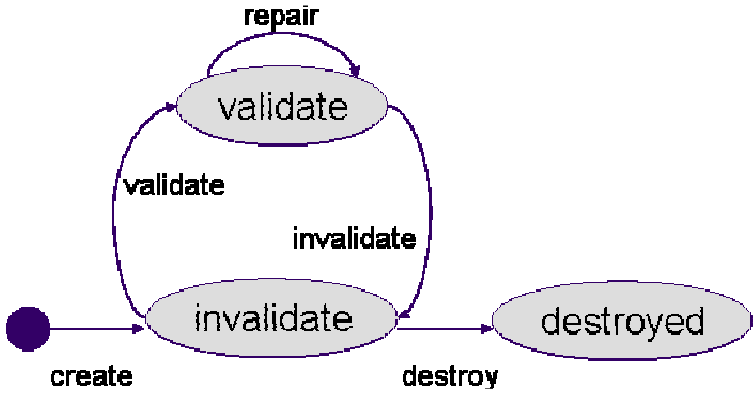


Figure 66 Cycle de vie d'un composant selon "Declarative Services"

1.2.2 Solutions à base de "POJOs" : Spring DM et iPOJO

POJO est un terme signifiant "Plain Old Java Object" ou "le bon viel objet Java". Ce concept a été introduit par Martin Fowler, Rebecca Parsons et Josh MacKenzie en l'an 2000 pour parler d'objets simples en dehors de tout modèle à composants complexe. L'utilisation de POJOs met en exergue le retour de cadriciels Java qui n'impliquent pas la soumission des développements à un modèle complexe. Les deux principaux cadriciels qui fournissent cette approche est les cadriciels pour serveurs d'application que sont EJB3 et Spring. Après analyse d'annotations laissées par le développeur dans le code, ils génèrent du code binaire (bytecode) pour la projection d'un modèle à composants dans le code au déploiement.

iPojo [EHL07] est une technique à POJO proposée dans la suite du projet Service Binder de l'équipe Adèle dont sont issus ces travaux de thèse et développée au sein du projet Apache felix. iPojo génère les gestionnaires du cycle de vie des composants à service déclarés dans une nouvelle syntaxe XML ou dans les annotations du code des bundles OSGi. Plus loin, iPojo propose la gestion de tout aspect fonctionnel par un conteneur au modèle quasiment transparent pour le développeur. Le développeur doit toutefois indiquer dans les techniques à POJO dans quels champs d'objets injecter les dépendances.

1.2.3 Pertinence et extensions pour le Home SOA

Le modèle d'automatisation de la composition de service est une technique du Home SOA (cf. Chapitre V.2.3) qui est réalisée par le modèle OSGi Declarative Services.

Description des dépendances et cycle de vie dont le contrôle est délégué à un conteneur. Le conteneur prend la forme d'un bundle nommé SCR.

Ce modèle de description de dépendances, de cycle de vie et le conteneur associé sont étendus dans les travaux de cette thèse afin d'enrichir les moyens de sélection de services et de supprimer l'ambigüité de celles-ci [2007-3][2006-4][2006-2]. En effet, le modèle OSGi Declarative Services montre les limitations suivantes :

- La déclaration des propriétés et du filtre de services est statique.
- Le filtre de services est une expression LDAP n'acceptant que types et opérateurs simples.
- Aucun algorithme de classement de services d'ordre total ne peut être ajouté.

Par conséquent, 4 améliorations ont été pensées afin de réutiliser SCR dans le cadriciel Home SOA (cf. déclaration de la Figure 67) :

```
<!-- A component requiring a service -->
<component name="service.requester">
       <implementation class="pack.ControlPoint"/>
       <reference name="PLAYER"
              interface="api.Player"
              target="(location=$location{MaxAndre})"
              optionalfilters="(hifi=true)"
              cardinality="1..1" policy="dynamic"
              bind="bindPlayer" unbind="unbindPlayer"
              sort-method="sortPlayers"/>
</component>
```

Figure 67 Déclaration de dépendances de service

1ère amélioration : des filtres optionnels sont ajoutés au premier filtre de service. Ils permettent une sélection plus fine des services lorsque plusieurs fournisseurs passent le filtre obligatoire. Les filtres optionnels peuvent être nombreux et organisés du plus important au moins important. De plus, des chiffres de cardinalité entre 1 et l'infini sont acceptés alors que le modèle OSGi Declarative Services ne les accepte pas.

2ème amélioration : les propriétés et les filtres de service deviennent modifiables à l'exécution. Cette amélioration utilise les techniques de contextualisation du registre de services du Home SOA (cf. section 1.6).

3ème amélioration : afin de montrer l'intérêt du demandeur de service pour la valeur de propriétés spécifiques dans l'environnement, il est permis d'écrire des modèles de valeurs interprétables par un gestionnaire de contexte particulier. Par exemple, une application cherchant les services disponibles dans la pièce où se trouve l'utilisateur Maxandre pourrait être exprimée par le filtre suivant : location=$location-room{Maxandre}. Les sources de contexte sachant interpréter ce modèle et ayant accès à l'information recherchée pourrait alors écrire la valeur à la place de l'expression modélisée. Chen et al. [ChK03] introduisent cette syntaxe pour les requêtes et la publication de valeurs contextuelles.

4ème amélioration : il est ajouté la déclaration du nom d'une méthode publique ("sort-method") à appeler afin de classer dynamiquement les services présents. Cela permet au développeur de programmer une méthode de tri complexe dans le langage de programmation des composants. Exposer tous les termes de l'algorithme dans des entités accessibles à l'extérieur du composant n'est pas aisé dans les applications usuelles et la sémantique de l'algorithme de yri peut être complexe. Pour ces raisons, l'algorithme est ici spécifié de manière interne au composant, ce qui apporte une contradiction à la lâcheté du couplage recherché. C'est une limite du système proposé. L'ajout de script à évaluer par un conteneur extérieur [DaL06] est une possibilité qui n'a pas été implémentée.

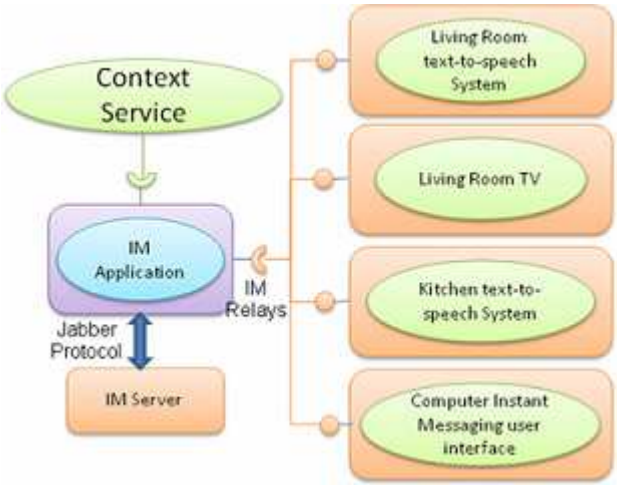


Figure 68 Recherche du meilleur service de restitution de messagerie ambiante

Les articles [2007-3][2006-4][2006-2] prennent l'exemple de la recherche des meilleurs haut-parleurs dans le voisinage de l'utilisateur. Dans l'article [2007-7], le meilleur service de restitution d'une session de messagerie ambiante est recherché (Figure 68).

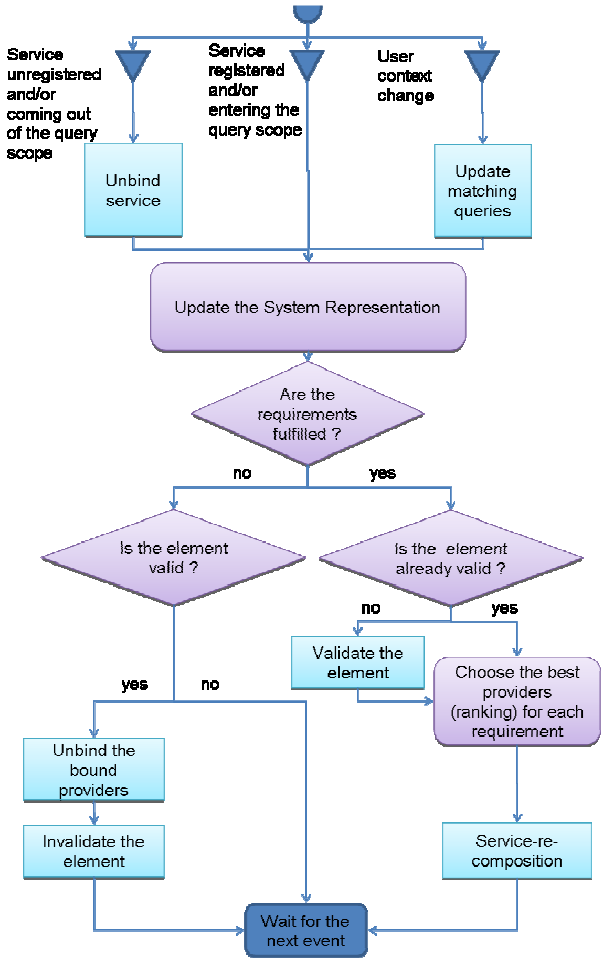


Figure 69 Boucle de rétroaction pour la reconfiguration de chaque composant

Ce dernier article présentait un modèle affiné par l'utilisation de techniques à POJO. La déclaration de la sélection du meilleur service pour chaque dépendance est alors déléguée à un gestionnaire particulier ("handler" dans la nomenclature d'iPojo) nommé Cadep pour "Contextd-Aware DEPendency". La Figure 70 montre la déclaration d'un filtre acceptant un

modèle de valeur ("filter") et d'une méthode de classement interne ("ranking-method") associé à une dépendance de services simplement déclarée dans le code par un champ ("field") dont le type est pris comme interface de service pour la dépendance.

```
<ipojo
xmlns:cadep="cadep.CadepHandler" >
 <component className="aim.App">
  <cadep:dependency field="relay"
    filter="(location=${Context:maxandre.location})"
    ranking-method="rankRelays"/>
 </component>
 <instance component="aim.App" name="IMApplication"/>
</ipojo>
```

Figure 70 Dépendance de service contextualisée par un gestionnaire iPOJO

Le gestionnaire de dépendances de service et le gestionnaire de classement contextual se partagent le travail de la gestion du cycle de vie du composant. Le détail de la boucle de rétroaction effectuée par ces gestionnaires est décrit par la Figure 69. Les événements d'enregistrement, de mise à jour et de dés-enregistrement de fournisseurs de service et les changements de propriétés du filtre de service déclenchent la mise à jour du système de représentation des services. Le classement des services et la reconfiguration du composant sont effectués dans le cas où les dépendances obligatoires peuvent être satisfaites.

1.3 Gestion des événements sur une plateforme OSGi

1.3.1 Les utilisations du Whiteboard Pattern

Le patron de conception "Tableau Blanc" est utilisé notamment par la spécification UPnP Device Service de la spécification OSGi [OSGi05]. Chaque application OSGi contrôlant des équipements UPnP implémente et enregistre un service de type UPnPEventListener avec des propriétés indiquant l'intérêt de ce client pour un ensemble d'événements particulier. Les 4 clefs spécifiées de propriétés potentielles correspondent aux critères de souscription UPnP :

- UPnPDevice.TYPE : type d'équipement aux services visés par la souscription.
- UPnPDevice.ID : identifiant d'équipement aux services visés par la souscription.
- UPnPService.TYPE : le type des services auxquels souscrire.
- UPnPService.ID : l'identifiant du service auquel souscrire.

L'enregistrement d'un service de type UPnPEventListener indique à l'UPnP Base Driver, implémentation du chapitre UPnP Device Service, que la souscription est demandée aux services correspondant aux critères ci-dessus. L'UPnP Base Driver souscrit à chaque service correspondant et retransmet tous les messages de notification dé-sérialisés à la méthode notifyUPnPEvent() de l'objet de service UPnPEventListener.

1.3.2 Event Admin : solution publish-subscribe

Les patrons de conception "Oserver" et "Tableau Blanc" étaient utilisés dans les spécifications OSGi jusqu'à la 4ème version où la spécification Event Admin suivant le paradigme publish-subscribe fit sont apparition. Différents chapitres furent alors modifiés pour être liés à l'Event Admin Service qui n'est autre qu'une implémentation du patron "publish-subscribe orienté service" (cf. Chapitre V.2.4.3). L'UPnP Device Service Specification s'est notamment vue spécifié ce nouveau détail de conception pour la possibilité d'un autre mode de communication par événements. La discussion des trois patrons en

Chapitre V.2.4 et section suivante montrent que des hypothèses au problème manquent à l'appel de cette solution.

1.3.3 Pertinence et extensions pour le Home SOA

Le patron de conception "Tableau Blanc" est pertinent pour la communication par événements entre le pilote orienté service et les points de contrôle implémentés parmi les applications hébergées par la plateforme de services. Le patron "publish-subscribe orienté service" ne l'est pas pour plusieurs raisons :

- Le pilote orienté service a besoin de connaître les objets intéressés afin de souscrire aux services distribués intéressants et seulement à ceux-ci. Le patron "publish-subscribe" et l'objet GeneralTopic entraine la souscription du pilote à tous les services de l'environnement, ce qui surcharge les traitements du pilote et de tous les services UPnP présents non seulement à la souscription mais lors des notifications. Ces dernières chargent de surcroit la bande passante.
- Le cas d'UPnP ajoute une hypothèse en défaveur du découplage apporté par le concept publish-subscribe : la souscription à un service UPnP génère un message de notification au seul nouvel abonné. Ce message initial permet à l'abonné de connaître l'état du service sans attendre le prochain événement. Seuls les patrons Observateur et Tableau Blanc permettent au pilote de détecter un nouvel abonné sur la plateforme et de notifier ce seul abonné.
- Les événements d'ensemble protocolaires Plug-n-Play généraux nécessitent une souscription à gros grain satisfaite par la seule interface XEventListener pour tout protocole X. La multiplication des interfaces n'est pas une hypothèse valide.

Le mécanisme suivant le Tableau Blanc qui est spécifié dans le chapitre UPnP Device Access est repris dans la conception du DPWS Base Driver de ces travaux de thèse [2008-2][2007-5]. La spécification délivrée dans le dépôt open source du projet Amigo[1] montre une hypothèse supplémentaire que ne peut supporter le service Event Admin :

- DPWS spécifie un mode de communication sollicit-response (cf. Chapitre II.2.4). Ce mode force le pilote et toute implémentation d'équipement à connaître ses clients afin que chacun d'entre eux puisse retourner une réponse au serveur.

Le cadriciel Home SOA apporte une extension (toutefois mineure) au mécanisme pour UPnP dans le cadre de DPWS : comme DPWS permet la souscription à des événements particulier sur chaque service, une propriété supplémentaire est nécessaire dans l'enregistrement des services de type DPWSEventListener. Cette propriété permet de préciser l'événement auquel souscrire auprès du service cible.

1.4 Pilotes protocolaires à la demande

1.4.1 OSGi Device Access

Le chapitre Device Access de la spécification OSGi [OSGi05] définit les concepts de courtage et d'installation de pilotes à la demande. Des services peuvent être enregistrés avec l'interface "Device" par les applications dans le registre de services afin d'indiquer un besoin de pilotes. Les propriétés enregistrées avec le service donnent des informations utiles à un gestionnaire d'équipements (Device Manager) responsable de l'interrogation de plusieurs

[1] France Telecom DPWS Base Driver specification, Java API, implementation and examples: http://gforge.inria.fr/frs/?group_id=160&release_id=1804

services de répertoires d'équipements (Device Locators) et d'un sélecteur (Device Selector) du meilleur pilote parmi les pilotes disponibles de ses répertoires. Le gestionnaire d'équipements réagit à l'enregistrement, la modification et la suppression des services, de leurs pilotes et des répertoires de pilotes dans le registre de services afin de télécharger et d'installer les pilotes les plus adéquats dans toutes les situations.

1.4.2 Pertinence et extensions pour le Home SOA

Ce procédé, allié à la capacité d'installation de modules à la demande d'OSGi, répond aux exigences techniques de chargement logiciel à la demande et de découverte protocolaire réactive définies dans la section Chapitre IV.2.2. Il peut être complété par la spécification de modules de découverte de protocoles utilisés sur le réseau afin d'initier le téléchargement des pilotes associés. [GBS03] spécifie de tels modules dans un modèle à composants (sans plateforme de services). Chaque module est alors responsable de l'écoute de messages réseau spécifiques – "ssdp:alive" et "ssdp:byebye" dans le cas d'UPnP et d'IGRS, "Hello" et "Bye" dans le cas de DPWS, etc. – et enregistre un service "Device" avec des propriétés indiquant l'ensemble protocolaire détecté. Cet enregistrement déclenche le procédé défini par la spécification OSGi Device Access d'installation du pilote adéquat.

1.5 Base Drivers et le contrôle d'équipements distribués

1.5.1 UPnP Base Driver (et Jini Base Driver)

La spécification OSGi définit des "Base Drivers" responsables de la représentation dynamique des équipements par des services dans le registre de services OSGi. Les détails protocolaires de l'invocation d'opérations distantes, de souscription et de notification sont respectivement masqués par les méthodes de l'interface de ces services, l'enregistrement d'écouteurs (EventListener) et les méthodes de notification de l'interface écouteur (cf. Chapitre V.3). Le chapitre "UPnP Device Service" définit les interfaces de représentation des équipements UPnP sur la plateforme OSGi. Le chapitre "Jini Driver Service" existait dans la version 3 de la spécification OSGi et est supprimé depuis la version 4 en raison de la non-adoption générale de Jini.

1.5.2 Pertinence et extensions pour le Home SOA

Les "Base Drivers" d'OSGi répondent partiellement aux exigences techniques de transparence de la localisation pour la recherche et l'utilisation de fonctions disponibles dans l'environnement dans le cadre spécifique du contrôle d'équipements standards. Ce cadre implique la définition d'un ensemble d'interfaces programmatiques en adéquation avec la couche de description de l'ensemble protocolaire standard visé. Le développeur de clients et de serveurs de ce protocole doit prendre en compte cet ensemble d'interfaces spécifiques. Le cadriciel Home SOA reprend le concept de Base Drivers [2008-5][2007-4]. Ces travaux de thèse l'appliquent dans la conception d'un pilote plus complexe – le DPWS Base Driver [2008-2] – et montrent que les interfaces de programmation peuvent être simplifiées dans le cadre des protocoles permettant la description de services [2006-3].

Le DPWS Base Driver [2008-2] est un pilote orienté service développé pour l'implémentation d'équipements DPWS et pour le contrôle de ceux-ci. De la même façon que l'UPnP Base Driver d'OSGi, les services DPWS locaux et distants sont enregistrés avec le même ensemble d'interfaces. Toutefois, a contrario de l'exemple OSGi, l'interface de description des services est séparée de l'interface d'invocation afin d'alléger le nombre de requêtes réseau, l'empreinte mémoire et la prise de ressources de calcul. Une interface de service supplémentaire est par ailleurs proposée pour simplifier le travail d'implémentation de l'interface de service des développeurs d'équipements. Ces deux particularités sont l'objet de modules complémentaire au module "Base Driver" principal (cf. section 3).

Les ensembles protocolaires qui définissent une syntaxe de description de service tels qu'UPnP permettent la génération de code spécifique pour l'implémentation et le contrôle d'équipements vérifiant une description donnée (cf. Chapitre V.4.3 et [2006-3]). L'UPnP Device Generator (cf. section 4.3) est un générateur de pilotes raffinés pour UPnP. Les pilotes générés présentent des interfaces moins génériques que celles de l'UPnP Base Driver. Cet outil augmente la transparence du développement de clients et de serveurs tout en imposant encore des interfaces en lien avec un standard.

1.6 Contextualisation des services

1.6.1 OSGi Configuration Admin Service

Le chapitre "Configuration Admin Service" définit une interface de service pour la configuration d'objets identifiés comme "gérés" ("managed"). Ces objets doivent être enregistrés dans le registre de service selon une interface particulière nommée "ManagedService".

1.6.2 Pertinence et extensions pour le Home SOA

La fonctionnalité offerte par le service Configuration Admin d'OSGi peut être utilisée dans le cadre de la contextualisation de services. En effet, le service inverse les rôles de client et serveur entre les objets à contextualiser et sources de contexte et permet à ces dernières d'ajouter toute information de contexte à la liste des propriétés objets "gérés". Ce service OSGi va même plus loin que le modèle décrit parmi les techniques du Home SOA (cf. Chapitre V.5.1) puisque le service se charge de la gestion de la présence des objets gérés. Les sources de contexte adressent leurs informations de contextualisation directement à ce service qui se chargent de les transférer aux objets gérés dès qu'ils sont présents.

Ce service de configuration est donc repris pour l'objectif de contextualisation de services et des requêtes de service. La relative nouveauté dans ces travaux de thèse est son emploi dans les applications conscientes du contexte et la reprise de ce mécanisme pour la contextualisation des requêtes de services [2007-3]. La contextualisation des services influence alors directement la composition de services au niveau de chaque composant et par conséquent, la gestion du cycle de vie de ces composants [2007-7][2006-2].

2 EXTENDED SERVICE BINDER

Masquer la distribution dans un environnement changeant a été le premier but recherché dans ces travaux de thèse. Cette première étude a débouché sur un premier prototype appelé Pervasive Service Binder [2006-4][2006-2] (ou encore Extended Service Binder) mêlant des techniques de l'informatique distribuée à un modèle de programmation à service. Tandis que la modularité du logiciel dessine le contour des différentes fonctions d'ensembles protocolaires (Découverte, Description, Contrôle synchrone), la conception mixe en revanche la gestion de la dynamique avec la gestion des aspects distribués dans une brique dont la scission est difficile. Ce prototype a fait l'objet d'un brevet [2005-1] et a été intégré dans un prototype de plus grande taille par différents partenaires industriels et académiques du projet Pise [2008-3][2006-6] et du projet Amigo [2006-5].

2.1 Choix des protocoles de découverte et de communication

Le projet Amigo a choisi DPWS pour le partage de services entre technologies de plateformes hétérogènes : les plateformes OSGi et .NET [2006-5]. Le choix de DPWS – plus exactement de SOAP, WSDL, WS-Discovery et WS-Eventing – est basé sur le besoin de

généralité pour la sérialisation de méthodes et d'objets de langages de programmations à la puissance d'expressivité importante.

Le protocole SLP couplé avec la communication binaire RMI est certainement le choix le plus approprié pour le partage de services entre plateformes OSGi. SLP possède les filtres de service LDAP pour point commun avec OSGi et RMI est un protocole de communication dérivé du langage Java, langage sur lequel est basée la technologie OSGi. C'est l'ensemble protocolaire pris dans le prototype Extended Service Binder [2006-2][2006-4][2008-3], en particulier dans le cadre du projet Pise [2006-6]. Par ailleurs, R-OSGi [RAR07] a fait un choix similaire en remplaçant toutefois RMI par un protocole ad hoc basé sur le langage Java. Toutefois, SLP a le défaut de ne pas définir de mécanisme de découverte passive. [RAR07] pallie ce défaut par l'envoi de requêtes périodiques de découverte avec une fréquence suffisamment élevée. Ce remède ne permet pas de rendre toute la dynamique d'une plateforme sur les plateformes distantes. Le défaut peut être comblé plus efficacement par la modification du protocole – dans le cas de la distribution de services OSGi, l'aspect normatif du protocole n'est pas prépondérant.

2.2 Architecture de l'Extended Service Binder

L'Extended Service Binder étend les mécanismes de composition dynamique de services locaux du "Service Binder" (pré-Service Component Runtime, cf. section 1.2.1 et [CeH03]) pour la composition tant locale que distante. L'Extended Service Binder prolonge la recherche locale par une recherche de services distants auprès d'un service d'annuaire connecté à un annuaire distant (SLP Directory Agent, cf. Chapitre III.1.1.5 ou moyens multicast de WS-Discovery, cf. Chapitre III.1.1.3). Il permet aussi la liaison de services lorsque les services trouvés sont distants [2005-1].

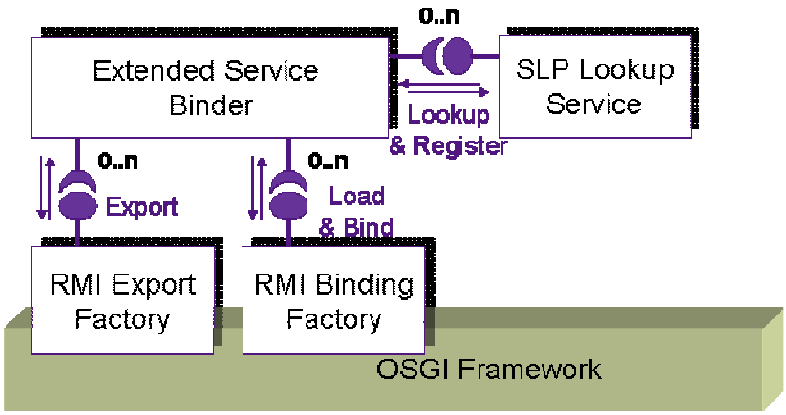


Figure 71 Extended Service Binder SLP-RMI

Le patron de conception "export-binding orienté service" est employé par le gestionnaire de chaque composant à l'import et à l'export de services (cf. Figure 71). Lors de la recherche de services, le gestionnaire prolonge la découverte locale auprès du service d'annuaire. Si des services distants conviennent, l'usine à liaison adéquate est appelée afin d'obtenir un mandataire pour le service correspondant. L'usine adéquate est celle qui répond à la description distante du service. La référence de service enregistrée comporte des propriétés désignant la technologie de communication.

Lors de l'export de services, les objets de service à exposer de manière distante sont passés à l'usine à exporter adéquate afin de créer une référence distante. Le gestionnaire de composant enregistre cette référence avec les propriétés du service auprès du service d'annuaire. Les propriétés importantes sont celles qui permettent le choix de l'usine à liaison, la création des mandataires adéquats par celle-ci et un numéro unique identifiant le service.

Plusieurs types de paires d'usine à liaison et à exporter peuvent être définis pour différents besoins. Dans le projet Amigo, les usines a liaison et à exporter se répondaient au travers de l'échange de mandataires SOAP, alors que dans d'autres projets, les mandataires utilisaient la technologie RMI. De même le service d'annuaire peut être implémenté selon des protocoles variés (WS-Discovery et SLP dans Amigo et Pise).

2.3 Création des mandataires et chargement de classes

La création de mandataires est voulue transparente pour le développeur. Celui-ci n'a qu'à préciser la nature locale ou possiblement distribuée de l'offre ou de la requête de service dans le fichier de description de composants. A cette fin, de nouvelles propriétés sont spécifiées et sont gérées par le modèle de description de dépendances étendu. De nouveaux attributs se retrouvent dans le fichier XML de méta données attaché au modèle :

- local-only : un attribut booléen attaché au noeud <requires> côté client. La valeur est déclarée vraie ou fausse si le service requis doit être découvert seulement localement ou sur tous les enregistrements disponibles de service. Si la propriété n'est pas écrite, la valeur par défaut considérée est déclarée vraie.

```
<bundle>
	<!-- A component requiring a service -->
	<component class="org.mine.abo.HelloServiceRequester">
		<requires
			service="org.mine.service.HelloService"
			filter="(language=en)"
			cardinality="1..1"
			policy="dynamic"
			local-only="false"
			bind-method="bindHelloService"
			unbind-method="unbindHelloService"
	/>
	</component>
</bundle>
```

- registry : une chaine de caractère attachée au noeud <provides> côté fournisseur. L'attribut est défini "local" ou "remote". Si la propriété n'est pas déclarée, la valeur par défaut considérée est "local". Les services sont toujours enregistrés localement afin de tirer bénéfice des mécanismes de la plateforme OSGi.

```
<bundle>
	<!-- A component providing services -->
	<component class="org.mine.abo.HelloServiceImpl">
		<provides
			service="org.mine.service.HelloService"
			registry="remote"/>
		<provides service="org.mine.service.SimpleService"/>
		<property language="en" type="string"/>
	</component>
</bundle>
```

La précision de ces propriétés entraine respectivement la création et le chargement de mandataires sur la plateforme à l'export et à l'import de services. Des techniques de génération de code binaire sont employées dans la création des mandataires. Deux procédés sont possibles :

- La création des mandataires est effectuée de manière statique à la génération des interfaces et les mandataires sont déployés dans le bundle d'interfaces. Avec RMI, la tache de compilation RMIC après compilation des interfaces permet la création de mandataires. Dans le projet Amigo, le projet ksoap

(ksoap2.sourceforge.net/) et axis (ws.apache.org/axis/) pouvaient être utilisé pour la génération de stubs au-dessus de la technologie SOAP.

- Les mandataires sont créés durant l'exécution. Java 5 génère les mandataires RMI de manière transparente. Dans Amigo, il est possible de générer les stubs à partir de la description WSDL des services grâce aux outils Axis (ws.apache.org/axis/). Si les interfaces de services sont écrites originellement dans un langage de programmation comme Java ou C#, la génération du fichier WSDL est préalablement nécessaire. Les outils axis permettent cette génération.

Le chargement des classes mandataires est délicat sur la plateforme OSGi puisque le cœur de la technologie est basé sur une architecture particulière de chargeur de classes (classloaders). En effet, un chargeur de classe est attaché à chaque bundle. Ce chargeur est responsable de charger les classes privées du bundle et celles des paquetages exportés si ceux-ci ne sont pas l'objet d'un import et qu'un autre bundle ne les exporte pas déjà. Dans le cas d'un import de service, le cadriciel OSGi se doit de procurer les classes chargées par un bundle exporteur au chargeur de classes du bundle importeur.

Dans le cas de l'import d'interfaces de services potentiellement distants, le chargeur de classes du bundle d'interfaces charge et exporte les classes d'interfaces et du proxy. Afin que l'export des classes du proxy soit transparent, elles sont générées dans le même paquetage que celui des interfaces. Le client, qui importe nécessairement le paquetage d'interface, a alors accès aux classes mandataires.

Afin que le client soit géré par l'Extended Service Binder, son activateur n'est autre qu'une classe de l'Extended Service Binder. Cet activateur, et par conséquent le gestionnaire de composant, est alors instancié par le chargeur associé au bundle client. Grace au chargeur du client que le gestionnaire de composant passe aux usines à liaison, ces dernières ont alors accès aux classes de proxy.

3 DPWS BASE DRIVER : UN PILOTE ORIENTE SERVICE

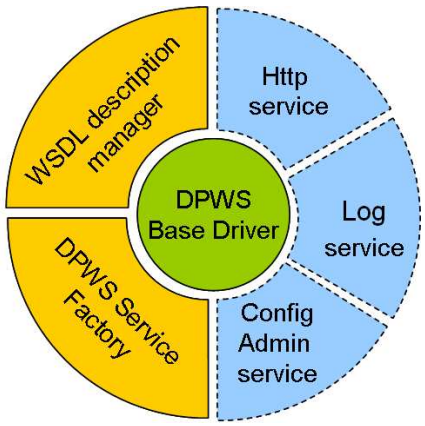


Figure 72 Vision schématique du DPWS Base Driver

Les études suivantes se sont évertuées à scinder la gestion de la distribution et l'automatisation de la composition de services. La plateforme de services est alors utilisée dans la définition et la programmation de pilotes – "drivers" – orientés services [2008-2]. Ces pilotes réagissent à l'apparition, la disparition, la modification des entités – souvent des équipements – du réseau domestique en enregistrant, supprimant, modifiant la représentation de chacune de ces entités dans le registre de la plateforme de services. Le registre de services est le réceptacle logique de toutes les entités pervasives de l'environnement domestique. Ce travail a été élaboré au sein du projet européen ITEA ANSO. Le DPWS Base Driver est un exemple de pilote orienté services [2008-2]. Ce travail a fait l'objet d'une présentation et

d’un RFP (Request for Proposal, c'est-à-dire document d’exigences techniques) acceptés par l’Alliance OSGi [2007-5][2007-6] – www.osgi.org. La spécification et l’implémentation sont de plus délivrées sur le répertoire open source du projet IST Amigo[1].

3.1 Architecture du DPWS Base Driver

L'architecture du pilote orienté service pour l'ensemble protocolaire DPWS est modulaire (cf. Figure 72 et Figure 73) afin de répondre aux différents scénarios visés tout en s'adaptant aux ressources disponibles de la plateforme [2008-2].

Le cœur de la plateforme (partie centrale) est constitué de deux bundles :

- Les interfaces de services correspondant à la description de services DPWS dans le langage objet Java.
- Le DPWS Base Driver : l'implémentation de ces interfaces pour la réification de services distants dans des services mandataires dans le registre de services et pour la détection des services implémentés sur la plateforme OSGi à exposer sur le réseau. Le DPWS Base Driver est construit à partir de la pile Java fournie en open source dans le projet ITEA SODA (www.soda-itea.org).

A l'import, le DPWS Base Driver est responsable de requérir activement les services DPWS présents sur le réseau lors de son démarrage et d'écouter les annonces de départ et d'arrivée des services sur le réseau par la suite. Grâce à ce mécanisme, le pilote est capable de refléter la présence des services DPWS par l'enregistrement, la modification et le retrait des services mandataires correspondant dans le registre de service OSGi. L'invocation de chaque objet mandataire est relayée en l'invocation des services distants. La présence d'écouteurs d'événements dans le registre de services est relayée en souscription aux services distants correspondants (cf. section 1.3).

A l'export, le DPWS Base Driver est responsable de refléter la présence de services sur la plateforme prêts à être exportés. Il utilise pour cela le protocole de découverte WS-Discovery. Il présente l'interface de contrôle adéquate face aux clients du réseau. Il relaie tous les messages d'invocation en appels de méthodes des services OSGi et les messages de souscription en enregistrant des écouteurs en tant que services OSGi (cf. section 1.3).

Trois bundles (partie droite en pointillé de la Figure 72) standards OSGi sont utilisés par le DPWS Base Driver. La réutilisation de services standards OSGi permet la mutualisation de ces bundles par différentes applications en supplément de l'utilisation par le DPWS Base Driver :

- Le service HTTP est employé par le pilote afin de présenter des servlets SOAP pour l'interface de notification des clients (import) et pour l'interface de contrôle des serveurs de la plateforme OSGi (export).
- Le service Configuration Admin est utilisé pour la configuration du pilote selon deux propriétés importantes : la possibilité de chargement tardif et la possibilité de requête réseau tardive (cf. paragraphes ci-dessous).
- Le service Log est un service de journalisation générique.

Deux bundles (partie gauche en trait plein de la Figure 72) peuvent être optionnellement installés sur la plateforme dans des scénarios précis :

[1] France Telecom DPWS Base Driver specification, Java API, implementation and examples: http://gforge.inria.fr/frs/?group_id=160&release_id=1804

- WSDL Description Manager : ce module permet la représentation objet complète des services distants après analyse de leur fichier de description. Cette description objet est superflue pour les clients qui connaissent par avance les opérations à invoquer sur les services recherchés. C'est le cas d'applications ayant des besoins de services aux types définis. En revanche, les points de contrôle génériques permettant de présenter tous les services existants et leurs opérations possibles nécessitent cette analyse exhaustive de la description. Les points de contrôle génériques sont souvent des applications d'administration ou de tests de services sur le réseau [NPD07][2008-2][2008-5].
- DPWS Service Factory : ce module facilite l'implémentation et l'enregistrement des services OSGi pour l'export dans le protocole DPWS. En effet, l'implémentation des interfaces symétriques pour clients et serveurs rend l'implémentation de ces derniers laborieuse. Le module présente des interfaces plus intuitives pour une création plus aisée de services et d'équipements DPWS. Ces implémentations sont traduites par le module en objets attendus par le DPWS Base Driver. Les métadonnées de l'équipement, la représentation WSDL et les relations entre équipement et services sont gérées par le module.

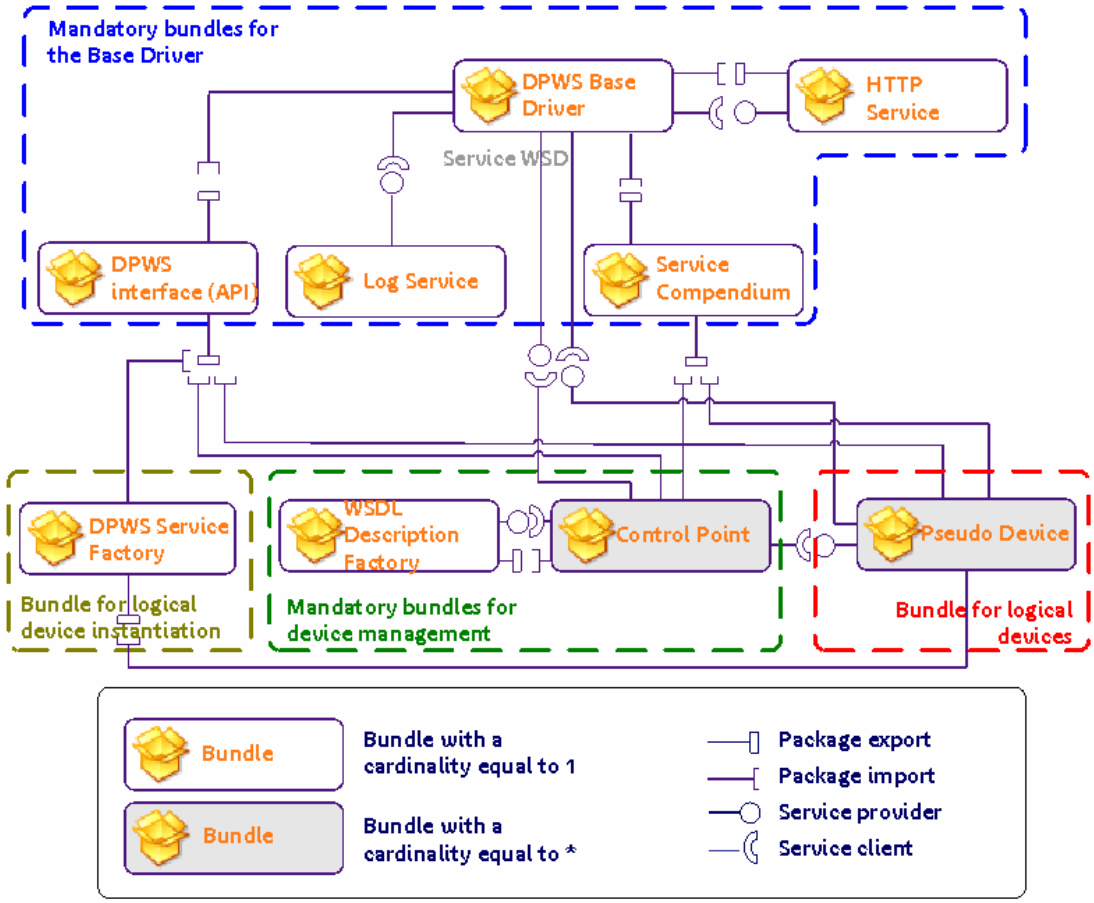


Figure 73 Architecture logicielle du DPWS Base Driver

3.2 Interface des services importés et exportés

3.2.1 Interfaces identiques pour l'import et l'export

Les applications OSGi clientes interagissent de manière transparente avec les équipements DPWS importés et les équipements DPWS développés localement sur la

plateforme. Pour cela, les deux types de service sont enregistrés selon les mêmes interfaces dans le registre de services, l'interface "org.osgi.service.dpws.DPWSDevice". Certaines propriétés attachées sont définies par le standard DPWS. Les services "équipement" sont enregistrés au moins avec les propriétés suivantes :

- DPWS.device.id, un identifiant unique sur le réseau,
- DPWS.device.scope, une propriété permettant la catégorisation des services selon des groupes logiques, par exemple selon leur domaine réseau.

Les services embarqués par ces équipements sont enregistrés sous l'interface "org.osgi.service.dpws.DPWSService". Ces objets sont enregistrés avec les 4 propriétés suivantes :

- DPWS.service.id, l'identifiant du service qui doit être unique parmi ceux des services embarqués par l'équipement.
- DPWS.service.parentid, l'identifiant du service hôte qui est égal au DPWS.device.id si le service hôte est l'équipement.
- DPWS.device.id, l'identifiant de l'équipement.
- DPWS.service.type, la liste des interfaces Web Services ("port types") implémentées par le service.

Cet enregistrement permet aux applications découvrant les services de remonter la hiérarchie de l'équipement grâce aux propriétés DPWS.service.parentid et DPWS.device.id. Afin de ne pas entrainer d'erreur lors de la recherche des services parents, ceux-ci doivent être enregistrés avant leurs services enfants. La hiérarchie des services d'un équipement peut être descendue dans le sens inverse grâce à la méthode appelée getHostedServices() disponible sur l'interface DPWSDevice.

3.2.2 Séparation des interfaces d'invocation et de description

Afin de permettre que l'analyse des descriptions WSDL des équipements et de leurs services soit optionnelle, les méthodes de contrôle sont insérées dans les interfaces principales (EndPoint et DPWSService de la Figure 74). Ces interfaces sont instanciées par le DPWS Base Driver en priorité et sont enregistrées dans le registre de services. La méthode de contrôle, EndPoint.invoke(), se trouve à la base de la représentation du service.

Au contraire, les interfaces de description ne comportent pas de méthodes de contrôle et ne s'obtiennent qu'à partir d'un getter de l'interface DPWSService : DPWSService.getMetadata(). Grâce à cette association simple, le DPWS Base Driver peut conditionner l'instanciation de la description complète :

- L'instanciation est conditionnée à la présence du DPWS Base Driver. Si le DPWS Base Driver est absent, une description vide est retournée.
- L'instanciation peut être conditionnée à l'appel de la méthode d'obtention des métadonnées, DPWSService.getMetadata(). Ce mécanisme est appelé chargement tardif (cf. section 3.3.2). Il peut être couplé à un mécanisme de requêtes réseau tardives (cf. section 3.3.1). Dans ce dernier cas, la requête réseau pour l'obtention des métadonnées n'est faite que lors de l'instanciation.

Cette séparation entre interfaces d'invocation et interfaces de description n'est pas intuitive. La spécification OSGi UPnP Device Service a d'ailleurs spécifié les méthodes d'invocation dans les feuilles de la représentation objet hiérarchique de la description des équipements UPnP.

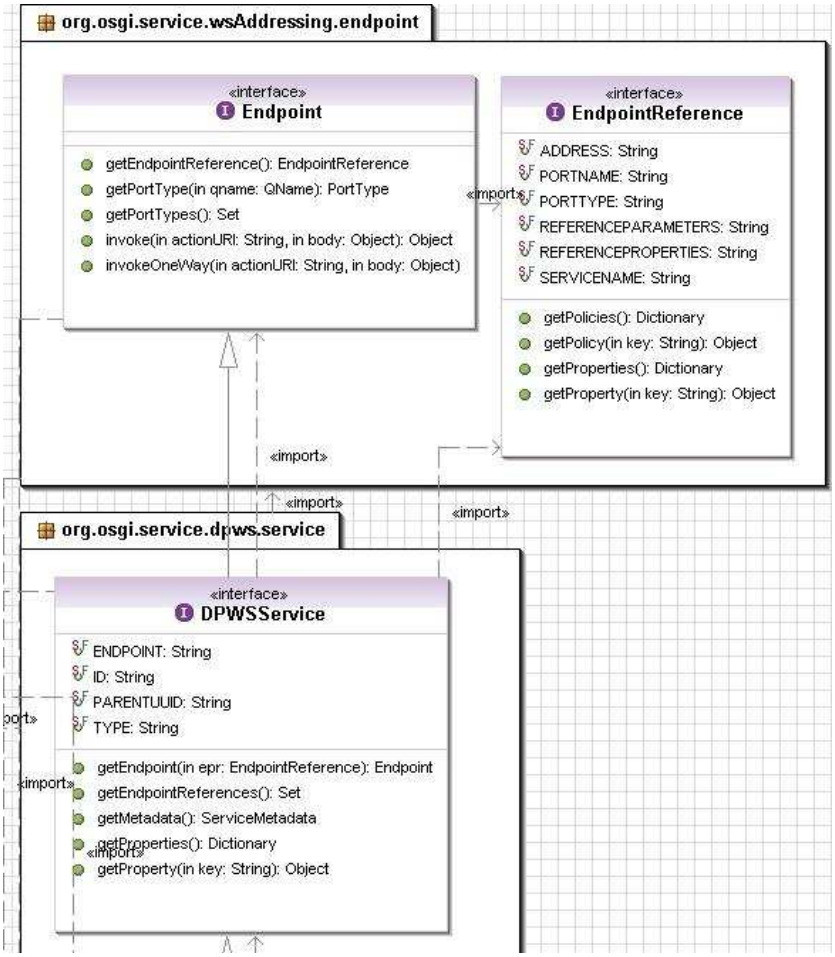


Figure 74 Interfaces de contrôle des services DPWS

3.3 Flexibilité de configuration

La flexibilité du pilote orienté service DPWS est aussi obtenue par des options de configuration. Une première option est définie sur le critère de requête tardive, une deuxième option sur le critère de chargement tardif.

3.3.1 Requête tardive et requête immédiate

Dans le cas de réseaux de nombreux équipements et dont la bande passante est précieuse, le mode de requête tardive permet de sauvegarder la bande passante. Ce mode restreint le nombre de requêtes sur le réseau à la juste demande d'informations. Le pilote n'envoie de requête d'informations exhaustives de description d'équipements que lorsque celles-ci sont demandées par l'appel de fonctions applicatives à l'exécution.

Le mode de requête immédiate vise les réseaux à large bande passante occupé par un nombre suffisamment faible d'équipements. Dans ce cas, le pilote réifie les équipements sur la plateforme avec toutes les informations qui sont possibles d'obtenir sur le réseau.

3.3.2 Chargement tardif et immédiat

En raison de mesures de sauvegarde de capacités de mémoire, le pilote peut être configuré en mode de chargement tardif. Dans ce cas, l'instanciation d'objets représentant les équipements DPWS est retardée jusqu'à l'appel explicite de certaines méthodes. Ce mode ralentit les premiers accès aux objets de représentation d'équipements.

Le mode de chargement immédiat est réservé aux équipements aux fortes capacités de mémoire ou dans le cas de la recherche d'applications à réactivité rapide.

4 PILOTES RAFFINES POUR LE CONTROLE D'EQUIPEMENTS

Le concept de plateforme de services simplifie la gestion de l'hétérogénéité. La pièce maitresse de l'approche proposée est le registre de services. Une médiation technique est effectuée par des pilotes dits "raffinés" qui transforme les représentations d'entités pervasives en objets spécialisés au domaine applicatif visé. Ces pilotes réagissent dynamiquement à l'apparition, la disparition, la modification des entités utiles dans le registre de services en enregistrant, supprimant, modifiant les objets d'adaptation. Le registre de services est ainsi le réceptacle de ces objets à la sémantique proche de celle des applications qui les requièrent. Cette approche donne un corps à un concept de la spécification OSGi [OSGi05] – la notion de "refined driver" y est mentionné. Tandis que les pilotes orientés services masquent les aspects distribués dans des "proxys" catégorisés par protocole, les pilotes raffinés masquent l'hétérogénéité des représentations protocolaires par des adaptations spécialisées. La médiation technique permet d'adapter les services découverts aux requis des applications. Cette approche opportuniste fait partie des techniques de composition de services dans les réseaux pervasifs – définies dans le "Home SOA" [2008-5][2008-4][2007-4][2007-2]. Ces travaux se sont déroulés en partie dans les projets IST ePerSpace et ITEA ANSO.

4.1 Point de contrôle générique sur le réseau domestique

Comme expliqué dans la section Chapitre V.4.2.1, la sémantique de la description des équipements DPWS est la plus générique parmi celle des autres ensembles protocolaires existant sur le réseau domestique. La technologie OSGi étant un ensemble d'interfaces et de mécanismes au-dessus de Java, le standard JWDL est l'ensemble d'interfaces pertinent pour la représentation générique de tous les équipements sur le réseau. En effet, JWSDL est la représentation objet des descriptions de Web Services.

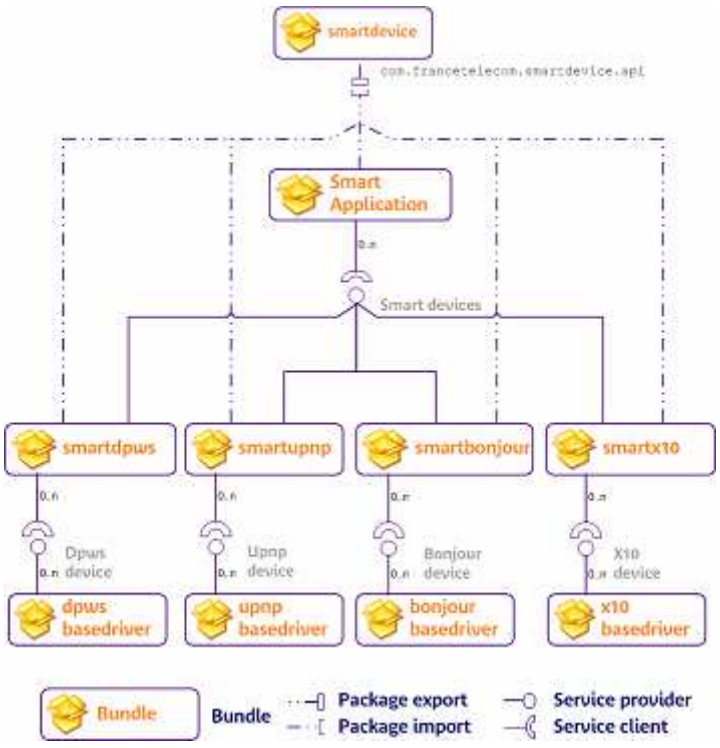


Figure 75 Architecture logicielle de l'application de contrôle générique

Le pivot pour le contrôle générique des équipements sur le réseau domestique (cf. scénario du Chapitre II.4.1.1.3) est un ensemble d'interfaces appelé "API Smart Device" [2008-5][2008-4], bâti à partir de JWSDL. La Figure 75 montre que ce bundle d'interfaces est utilisé par la couche de pilotes raffinés : smartdpws, smartupnp, smartbonjour, smartx10. Cette couche de pilotes raffinent la représentation des équipements réifiés par la première couche de pilotes orientés services : DPWS Base Driver, UPnP Base Driver, Bonjour Base Driver, X10 Base Driver. Les applications développées au-dessus de ce cadriciel Home SOA découvrent et contrôlent tous les équipements au travers de l'API unique Smart Device.

L'API Smart Device est constituée des interfaces de contrôle SmartDevice et SmartService, et des interfaces de représentation des paramètres d'opérations d'invocation étendant l'interface SmartParameter (cf. Figure 76). Le SmartDevice est un SmartService particulier. L'interface SmartService décrit un service générique avec :

- sa méthode d'invocation générale permettant d'appeler des noms d'opérations spécifiques,
- sa méthode d'obtention des descriptions de métadonnées,
- sa méthode d'obtention des services enfants (cf. section 3.2.1),
- sa méthode d'obtention de la description JWSDL.

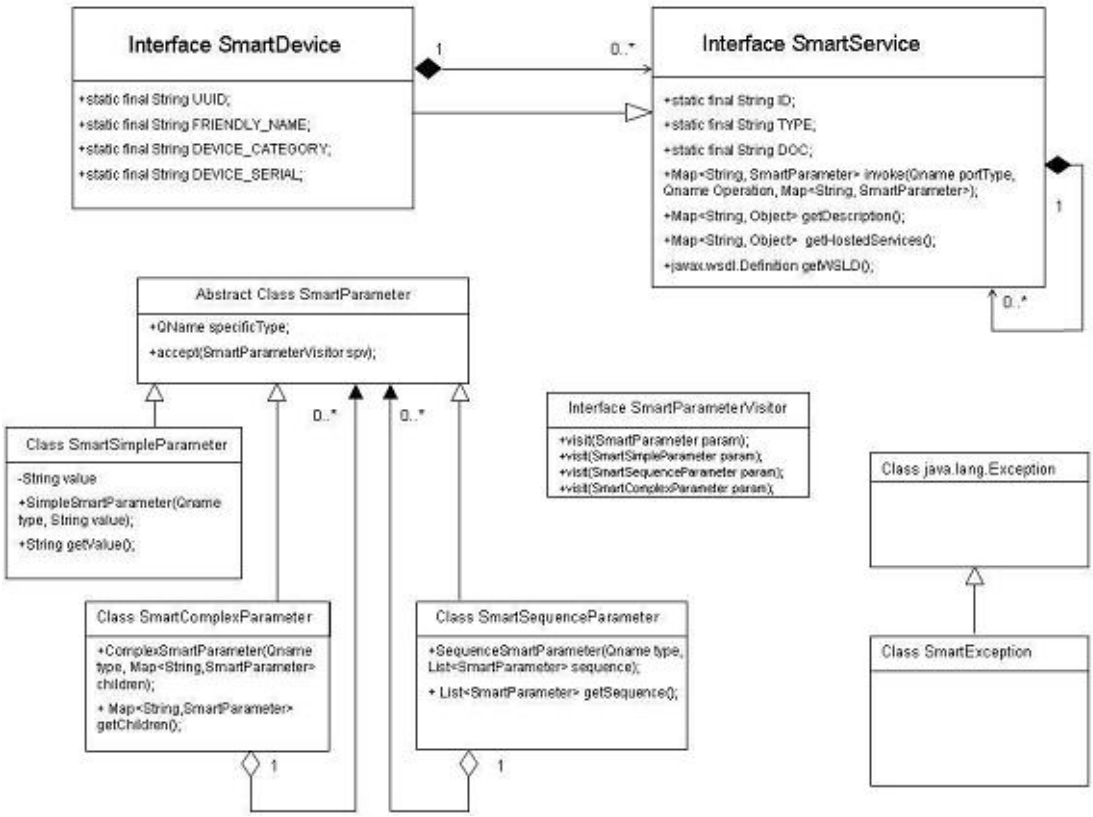


Figure 76 API Smart Device

L'interface SmartParameter et les classes l'implémentant obéissent au patron de conception Visiteur [GHJV95] (cf. Figure 76). Ce patron permet l'ajout de fonctionnalités aux différentes implémentations de SmartParameter sans changer leur code. Dans le cas exposé, le problème provient directement de l'usage de ces interfaces par des points de contrôle générique. Ces derniers découvrent les services et naviguent dans leur représentation pour traiter leurs paramètres un à un, par exemple pour les présenter dans une interface graphique [2008-4]. L'implémentation naturelle de ce traitement est une suite de structures

conditionnelles "if-else" importante testant le type de chaque paramètre avant de le traiter, ce qui est à éviter dans une programmation objet. Les structures conditionnelles sont du type :

```
if (parameter.instanceof(SmartComplexParameter),
treatSmartComplexParameter(parameter);
```

La programmation orientée objet permet de profiter judicieusement de l'héritage lorsqu'il est possible d'ajouter une fonction de traitement à l'interface parent. Les traitements effectués par les applications clientes étant variées et l'API devant demeurer unique pour toutes les applications, l'ajout de fonctions de traitement à chaque changement d'applications n'est pas possible. Le patron Visiteur permet alors l'évolutivité de l'utilisation sans modifier les interfaces communes. La structure conditionnelle est remplacée par l'appel de méthode :

```
parameter.accept(treatmentVisitor);
```

Grâce à l'héritage et au polymorphisme, la méthode "accept" appelée est celle de la classe enfant, soit SmartComplexParameter dans le cas ci-dessus. L'implémentation de celle-ci est écrite pour appeler la méthode adéquate de l'interface SmartParameterVisitor, soit : SmartParameterVisitor.visit(SmartComplexParameter).

La méthode d'invocation prend en arguments d'entrée le nom de l'opération, le nom de l'interface de Web Services (port type) et la table des paramètres d'entrée (du type SmartParameter) de l'opération.

A chaque protocole est associé un pilote orienté service et un pilote raffiné. La chaine de pilotes reflète la présence de chaque équipement par la présence d'un objet de type SmartDevice dans le registre de services. Les applications de contrôle générique découvrent dynamiquement ces objets dans cette représentation uniforme, naviguent dans leur description et leurs opérations afin de pouvoir les invoquer de manière uniforme par la méthode d'invocation décrite.

4.2 Home cinéma intelligent : contrôle multimédia et domotique

Le scénario de Home Cinéma intelligent (cf. Chapitre II.4.1.1.1) est aussi implémenté au-dessus d'une architecture à deux étages : un étage de pilote orienté services et un étage de pilotes raffinés (cf. Figure 77). Les premiers réifient les équipements et leurs services dans un ensemble d'interfaces attaché à chaque ensemble protocolaire. Les seconds transforment les objets de ces interfaces dans des ensembles d'interfaces attachés sémantiquement à des domaines d'applications spécifiques, multimédia et domotique dans le scénario décrit.

4.2.1 Pilotes orientés services

Quatre pilotes orientés services sont utilisés dans le prototype de Home Cinéma intelligent. L'UPnP Base Driver réifie chaque équipement UPnP dans un objet de type org.osgi.service.upnp.UPnPDevice [OSGi05]. Chaque objet UPnPDevice est enregistré et retiré dans le registre de service selon sa disponibilité sur le réseau grâce aux mécanismes de découverte active et passive spécifiés par le protocole SSDP d'UPnP. La spécification OSGi UPnP Device Service définit des interfaces de description hiérarchique dont les feuilles contiennent les méthodes d'invocation. Ce couplage fort entre invocation et description implique que le fichier de description soit requis auprès des équipements avant toute invocation. Une conception au couplage plus lâche peut être proposée (cf. section 3.2.2).

L'ensemble protocolaire Apple Bonjour est traité d'une manière similaire. Le Bonjour Base Driver réifie chaque équipement obéissant à ce protocole dans un objet de type BonjourDevice qui est enregistré dans le registre de service avec les propriétés attachées à l'équipement. Bonjour n'étant toutefois qu'un protocole de découverte, l'interface BonjourDevice n'expose pas de méthodes de contrôle.

4.2.2 Pilotes raffinés pour le multimédia

Les pilotes raffinés multimédias réagissent dynamiquement à la disponibilité des objets UPnPDevice et BonjourDevice afin de présenter des objets dans un ensemble d'interfaces unifiées pour le domaine du multimédia. Le pilote raffiné "upnp mediasource" de la Figure 77 maintient une liste d'objets de type MediaSource (cf. Chapitre V.4.2.3) correspondant à la liste d'objets UPnPDevice du type UPnP Media Server – urn:schemas-upnp-org:device:MediaServer:x avec x appartenant à {1,2,…}. Une propriété enregistrée avec l'objet UPnPDevice précise ce type de service au sens UPnP. De la même manière, le pilote "upnp mediasink" de la Figure 77 enregistre un objet de type MediaSink (cf. Chapitre V.4.2.3) pour chacun des objets UPnPDevice de type UPnP Media Renderer – urn:schemas-upnp-org:device:MediaRenderer:x et modifient cet enregistrement en réaction aux modifications de la description de l'équipement sur le réseau.

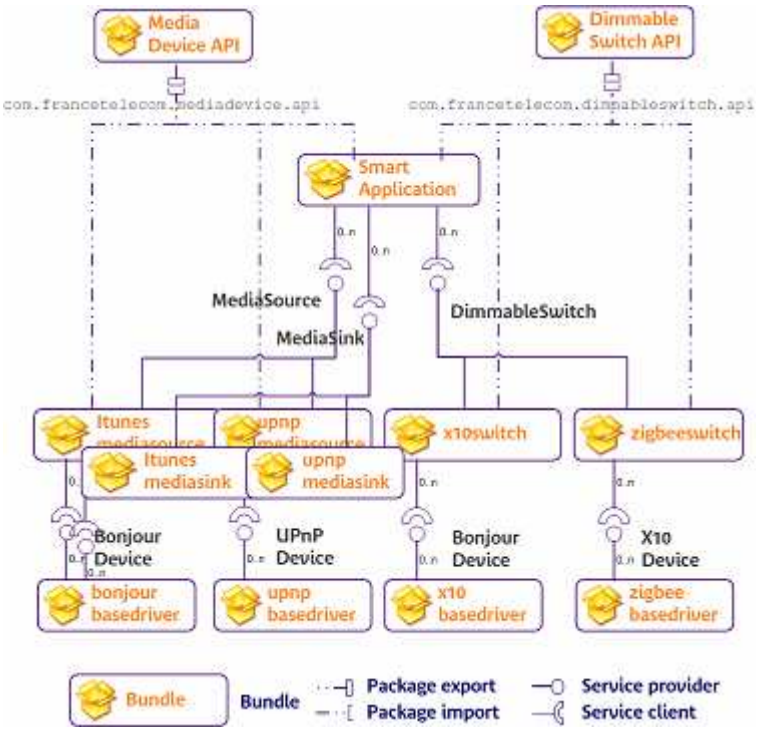


Figure 77 Architecture du "home cinéma" réactif

Comme iTunes est un protocole dédié uniquement au contrôle d'équipements multimédias, il est directement traité par un pilote raffiné multimédia. Le pilote "itunes mediasource" de la Figure 77 est responsable de la découverte dynamique des objets BonjourDevice du type "_daap._tcp.local." dans la syntaxe définie par Bonjour et de leur réification dans l'interface MediaSource.

Les pilotes orientés service et les pilotes raffinés sont utilises ici pour projeter les fonctions des équipements découverts sur les fonctions de source et de puits multimédias. Les pilotes orientés services UPnP et Bonjour peuplent dynamiquement le registre de service OSGi avec des instances des types respectifs UPnPDevice et BonjourDevice. Ces interfaces masquent les détails protocolaires de la communication distribuée tout en maintenant une représentation proche de celle que procure le protocole de description. Les pilotes raffinés réagissent dynamiquement à l'enregistrement de ces instances afin d'enregistrer des objets attachés à des interfaces à la sémantique multimédia. Ce raffinement cache les spécificités d'ensemble protocolaires pour embrasser la spécificité d'un domaine d'application.

4.2.3 Pilotes raffinés pour la domotique

Les équipements du domaine domotique sont l'objet d'un traitement similaire. Tandis que les pilotes X10 et ZigBee reflètent la dynamique des équipements par la dynamique d'enregistrement des instances d'objets de type X10Device et ZigBeeDevice le registre de services, les pilotes raffinés X10switch et ZigBeeswitch peuplent dynamiquement le registre en instanciant des objets de à l'interface DimmableSwitch exposant les fonctionnalités de variateurs de puissance électrique et d'interrupteurs. Ces fonctionnalités sont communes aux lumières, aux volets et à d'autres équipements de la maison.

L'application au cœur du Home Cinéma intelligent – Smart Application dans la Figure 77 – compose les services multimédia et les services domotiques disponibles dans le registre. Il propose à l'utilisateur de naviguer parmi les contenus multimédias qui sont exposées par les sources multimédias et de jouer les contenus choisis sur un ou plusieurs puits multimédias comme la télévision du salon. Le film peut être démarré à partir de l'interface utilisateur de l'application ou à partir de la télécommande de la télévision elle-même. Lorsque la télévision est démarrée, un événement standard (UPnP) est traduit par les pilotes en tant qu'événement de lecture de la part du puits (la télévision). L'application réagit à cet événement en agissant sur les objets DimmableSwitch, ce qui tamise la lumière et baisse les volets dans le salon. Les informations de localisation sont ajoutées aux références de services du registre par des sources de contexte grâce à l'utilisation des moyens de contextualisation de services (cf. section 1.6).

4.3 UPnP Device Generator : génération de code de pilotes raffinés

L'outil UPnP Device Generator implémente les exigences techniques décrites dans le RFP OSGi [2006-3] et les concepts développés dans la section Chapitre V.4.3. L'outil permet de générer des interfaces Java spécifiques ("Generated Interface" de la Figure 78) à partir d'une description XML UPnP. Les sections suivantes introduisent la projection des différentes définitions UPnP sur les éléments de programmation Java.

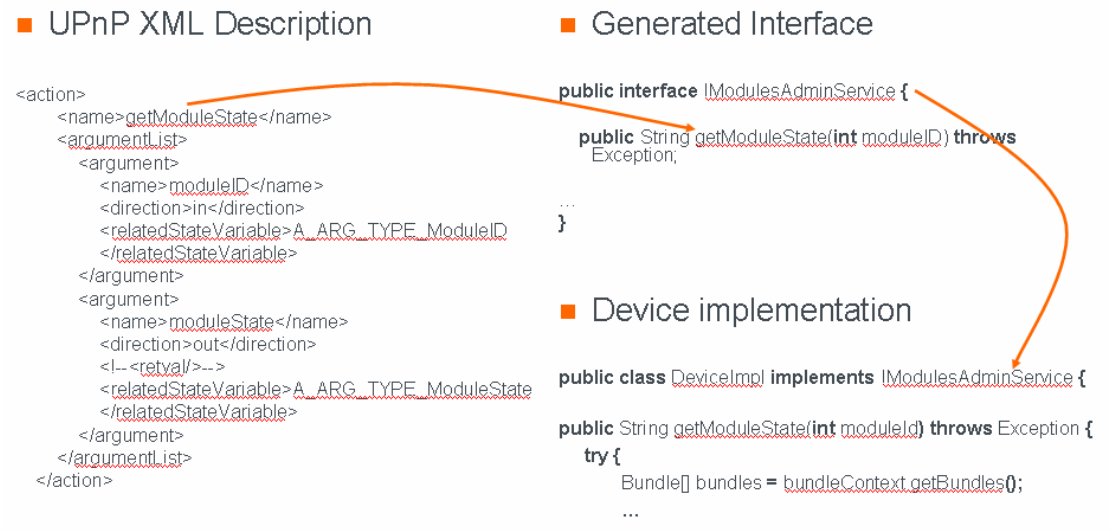


Figure 78 Génération de code d'interfaces Java à partir de descriptions UPnP

4.3.1 Projection des structures UPnP sur les structures Java

L'architecture UPnP [UPnP06] définit 3 entités distinctes : l'Equipement (Device), le Service, et le Point de contrôle (Control Point). L'Equipement embarque des Services et d'autres Equipements. Les Services sont des listes d'actions et de variables d'états. Le Point de Contrôle invoque les actions et souscrit à la notification des changements de variables

d'états des services (cf. Chapitre III.1.1.1). Le Tableau 6 montre les équivalences faites par l'outil UPnP Device Generator entre structures UPnP et structures Java/OSGi.

Tableau 6 Equivalence entre structures UPnP et structures Java/OSGi

UPnP	Java/OSGi
Equipement du type: urn:schemas-upnp-org:device:<DeviceType>:V	Objet Java implémentant l'interface Java <Device-Type>Device et enregistré dans le registre de service.
Propriétés de l'Equipement (paires attribut-valeur)	Propriétés (paires attribut-valeur) de service OSGi attachées à l'objet enregistré avec l'interface <DeviceType>Device.
Liste de services décrite dans la description de l'Equipement aux types : urn:schemas-upnp-org:device:<ServiceType>:V	Liste de méthodes get<ServiceType> dans l'interface <DeviceType>Device.
Service du type : urn:schemas-upnp-org:service:<ServiceType>:V	Objet Java implémentant l'interface Java <Service-Type>Service et enregistré dans le registre de service.
Point de contrôle découvrant et contrôlant les équipements du type urn:schemas-upnp-org:device:<DeviceType>:V et/ou les services du type urn:schemas-upnp-org:service:<ServiceType>:V.	Composant de service (modèle Declarative Services) demandeur des services OSGi à l'interface <Device-Type>Device et/ou l'interface <ServiceType>Service.

4.3.2 Projection des types UPnP sur les types Java

Une projection naturelle des types d'arguments UPnP est proposée par la spécification OSGi UPnP Device Service [OSGi05]. La spécification propose l'utilisation d'objets mais l'utilisation de types Java primitifs semble plus simple (cf. Tableau 7).

Tableau 7 Equivalence entre types UPnP et types Java

UPnP	Java/OSGi
ui1, ui2, i1, i2, i4, int	int.class
ui4, time, time.tz	long.class
r4, float	float.class
r8, number, fixed.14.4	double.class
char	char.class
string, uri, uuid	java.lang.String
date, dateTime, dateTime.tz	java.util.Date
boolean	boolean.class
bin.base64, bin.hex	byte[].class

4.3.3 Projection des actions UPnP sur les méthodes Java

Les actions UPnP définissent une liste d'attributs de sortie qui ne correspond pas directement à l'objet unique retourné par les méthodes Java. Le Tableau 8 spécifie les équivalences entre éléments d'actions et éléments de méthodes.

Tableau 8 Equivalence entre actions UPnP et méthodes Java

UPnP	Java/OSGi
Actions SOAP nommées <ActionName>	Méthodes Java nommées <actionName>
Zéro, un ou plusieurs arguments d'entrée nommés <Argument> qualifié par un type UPnP	Zéro, un ou plusieurs arguments d'entrée nommés <argument> qualifié par un type simple Java
Zéro, un ou plusieurs arguments de sortie nommés <Argument> qualifié par un type UPnP	void, un argument d'un type simple Java ou un Object aux attributs d'objets définis par les noms <arguments> et les types simples Java équivalents. Les attributs sont privés et accessibles via des méthodes get (getters) publiques. Ils sont tous configurables initialement par le constructeur.

4.3.4 Projection de la notification UPnP sur le mécanisme Java

La souscription et la notification d'événements définies par le protocole GENA d'UPnP trouve son correspondant dans le patron "Tableau Blanc" (cf. Chapitre V.2.4.2). Le Tableau 9 définit les équivalences.

Tableau 9 Equivalence entre mécanismes de notification UPnP et notification Java/OSGi

UPnP	Java/OSGi
Variables d'état déclarées comme "notifiables" ("evented"). Chaque fois que la valeur d'une variable change, les valeurs des variables changées sont envoyées dans un message à tous les Points de Contrôle abonnés. Un Point de Contrôle peut souscrire à un service mais pas à une variable spécifique.	Champs d'un objet <ServiceType>StateEvent, argument de la méthode de notification notify<ServiceType>StateEvent de l'interface <ServiceType>StateListener. L'implémentation de cette interface doit être enregistrée en tant que service OSGi afin d'indiquer une souscription. Le tableau blanc nécessite toutefois que le Point de Contrôle indique l'identifiant de service dans les propriétés s'il souhaite souscrire à une instance de service précise. L'observateur, malgré ses défauts, a l'avantage de ne pas demander la précision ci-dessus. L'équivalent de la souscription est alors l'appel à la méthode nommée addStateListener(<ServiceType>StateListener). Des adapteurs d'écouteurs <ServiceType>VariableListener peuvent être proposés dans les deux cas afin que l'application cliente OSGi puisse n'être virtuellement notifiée que par le changement de quelques variables d'état. Avec l'observateur, la méthode add<ServiceType>VariableListener(<ServiceType>VariableListener) est alors ajoutée à l'interface <ServiceType>Service interface. (cf. exemples ci-dessous).
Variables d'état	Objets Java définies avec une méthode setValue(type) déclenchant la notification d'objets écouteurs.
Message de notification reçu par tous les abonnés.	Objet de type <ServiceType>StateEvent à tous les objets de type <ServiceType>StateListener and et <ServiceType>VariableListener chaque fois que l'UPnPEventListener du pilote est notifié de l'objet UPnPEvent adéquat.

Voici par exemple l'interface de notification d'un Point de contrôle pour le service UPnP du type standardisé urn:schemas-upnp-org:device:Dimming:1:

```
public interface DimmingStateListener {
public void notifyDimmingStateEvent(DimmingStateEvent state);   }
```

Et la classe d'événement :

```
public class DimmingStateEvent {
       private Dictionary dict;
       public DimmingStateEvent(Dictionary dict) {…}

       public int getRampRate () {dict.get("RampRate");}
       public int getStepDelta () {dict.get("StepDelta");}
       public boolean getRampPaused () {dict.get("RampPaused");}
       public int getLoadLevelStatus () {dict.get("LoadLevelStatus");}
       public boolean getIsRamping () {dict.get("IsRamping");}   }
```

L'interface DimmingVariableListener à la notification plus naturelle dans la programmation objet. Chaque variable d'état notifiable se voit attachée une méthode :

```
public interface DimmingVariableListener {
       public void rampRateChanged(int arg0);
       public void stepDeltaChanged(int arg0);
       public void rampPausedChanged(boolean arg0);
       public void loadLevelStatusChanged(int arg0);
       public void isRampingChanged(boolean arg0);   }
```

Cela permet la définition d'un adapteur abstrait naturel pour le développeur Java. La classe abstraite ci-dessous implémente toutes les méthodes avec des méthodes vides. Elle peut être étendue et n'implémenter que quelques unes des méthodes afin d'obtenir un écouteur

notifié seulement par le changement de quelques variables spécifiques. De tels adapteurs sont spécifiés pour la notification d'événements d'interfaces graphiques Java Swing par exemple.

```
public abstract class DimmingVariableAdapter implements DimmingVariableListener {
        public void rampRateChanged(Integer arg0) {}
        public void stepDeltaChanged(Integer arg0) {}
        public void rampPausedChanged(Boolean arg0) {}
        public void loadLevelStatusChanged(Integer arg0) {}
        public void isRampingChanged(Boolean arg0) {}   }
```

4.3.5 Utilisation du UPnP Device Generator

L'outil permet de générer le bundle d'interfaces et le pilote raffiné correspondant à une description XML UPnP. Ce pilote traque dynamiquement les objets UPnPDevice du type UPnP mentionné dans la description. Il transforme ces objets présents sur la plateforme en objets obéissant à l'interface spécifique à la description de l'équipement. Les interfaces, plus intuitives pour un développeur Java, facilitent le développement de Points de Contrôle et d'équipements de ce type UPnP.

Cet outil a été utilisé notamment dans le prototypage d'équipements UPnP DM (cf. section 7). A partir de spécifications XML, il est simple de générer des pilotes raffinés et d'implémenter les interfaces avec l'API de gestion de cycle de vie de la plateforme OSGi.

5 COMMUNICATION INTERPERSONNELLE ENRICHIE

Le prototype de communication interpersonnelle enrichie [2008-6] repose sur la flexibilité du cadriciel Home SOA dans un service innovant séparant un terminal de vidéophonie entre les différents équipements de la maison. Le prototype exploite l'architecture du home cinéma intelligent et les capacités de négociation de flux multimédia du protocole SIP. Ce prototype est utilisé dans le projet national Systerminal labellisé par le pôle français de compétitivité "Images et Réseaux".

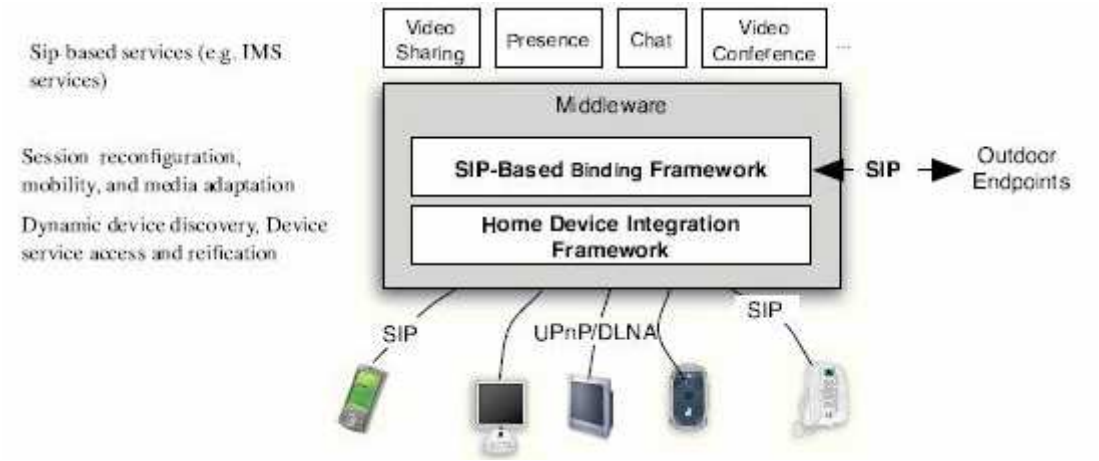


Figure 79 Intergiciel pour communication interpersonnelle enrichie

5.1 Architecture de l'intergiciel de communication enrichie

L'architecture de l'intergiciel permettant l'éclatement d'un terminal SIP entre équipements multimédias et de téléphonie du réseau domestique est schématisée par la Figure 79. Le cadriciel appelé "SIP Based Binding Framework" centralise l'intelligence de l'application. Il gère les sessions de communication entre les interlocuteurs interne et externe à la maison en les modifiant pour les adapter à la configuration matérielle la plus confortable

pour l'interlocuteur interne. A cette fin, il découvre et contrôle des équipements domestiques grâce au cadriciel appelé "Home Device Integration Framework".

La Figure 80 détaille l'architecture de l'intergiciel proposé. Le SIP Binding Manager est le composant principal de l'architecture. Il compose les services multimédias et de téléphonie suivant leur disponibilité dans le registre de service symbolisé par l'entité "Device Discovery". Afin de garantir l'unicité de la session pour les utilisateurs interne et externe, le SIP Binding Manager joue le rôle de tierce partie du mécanisme 3PCC (Third-Party Call Control, IETF RFC 3725, cf. Chapitre III.2.1.2). L"interlocuteur interne n'établit et ne raccroche qu'une seule communication constituée de plusieurs flux à la provenance et la destination d'équipements domestiques. De même, la mobilité de l'utilisateur est masquée à l'utilisateur externe dont le terminal SIP standard réagit aux demandes de modification de session par négociation et signalisation des flux vers des adresses changeantes.

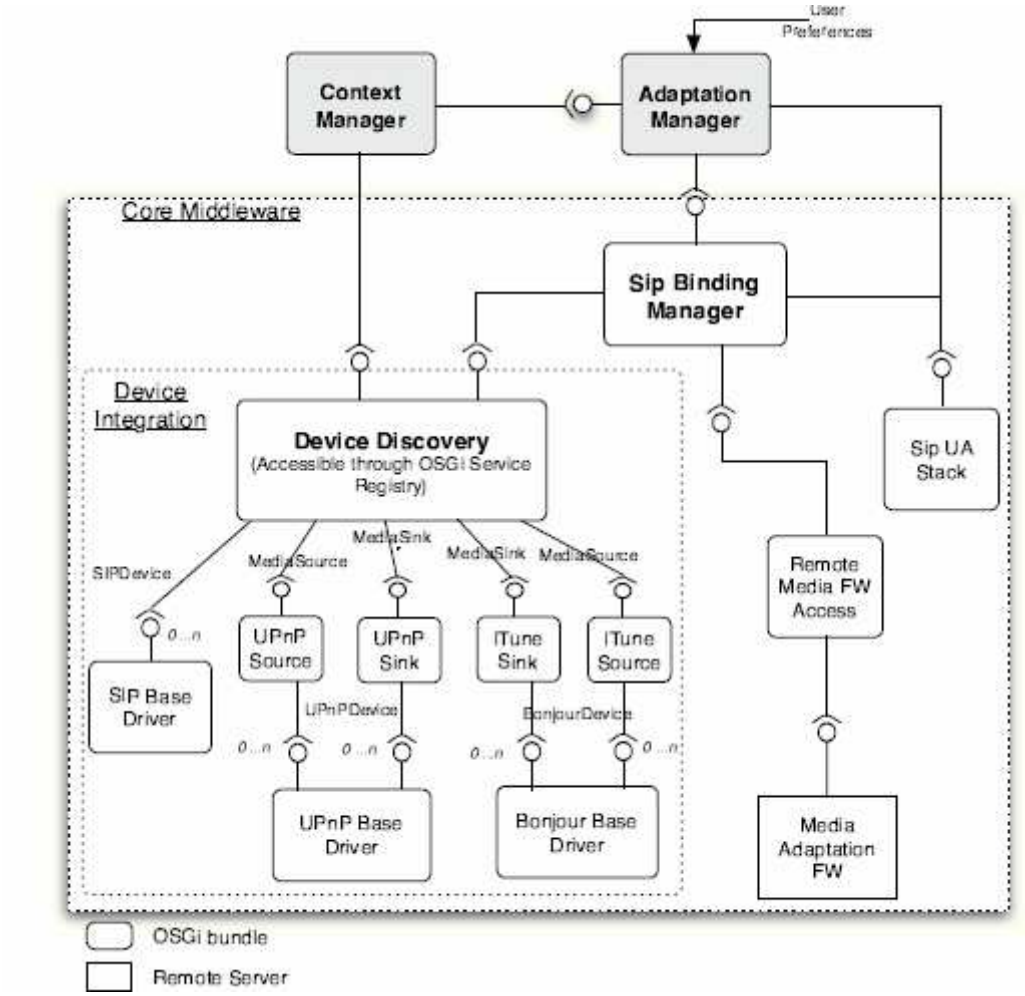


Figure 80 Architecture pour la communication interpersonnelle enrichie

Le cadriciel d'intégration d'équipements réutilise l'architecture logicielle de contrôle d'équipements multimédias (cf. section 4.2). Comme les interfaces de contrôle d'équipements multimédias ne s'adaptent pas tout à fait aux interfaces de contrôle d'équipements de téléphonie, les équipements SIP sont toutefois gérés de manière spécifique aux travers des interfaces SIPDevice fournies par le SIP Base Driver (cf. section 5.2). En effet, les équipements de téléphonie sont premièrement à la fois une source et un puits audio et gèrent deuxièmement l'établissement de session avec l'avertissement de l'utilisateur par sonnerie.

Enfin, la gestion du contexte est effectuée selon l'architecture exposée dans la section 1.6. Le "Context Manager" contextualise le registre de service par ajout d'informations de

contexte aux services enregistrés tandis que le SIP Binding Manager est notifié des changements et ses requêtes de services sont modifiées au travers de l'Adaptation Manager.

5.2 Pilote orienté service SIP

Les téléphones SIP participent à la composition de services de l'application. Ils sont aussi des équipements pervasifs qui se connecte et se déconnecte du réseau domestique avec un protocole de communication standard particulier. Chaque équipement est par conséquent représenté comme un service dans le registre OSGi dès lors qu'il est présent dans l'environnement. C'est pourquoi un pilote orienté service est nécessité pour SIP.

Malheureusement, le protocole SIP n'offre pas de mécanisme explicite de découverte par différents clients. Cependant, SIP définit un annuaire appelé "registrar" qui obtient l'enregistrement des téléphones le connaissant. Le SIP Base Driver est alors un registrar SIP par lequel tous les équipements SIP des habitants de la maison doivent s'enregistrer. Cela ajoute un élément d'architecture SIP.

Les équipements SIP qui s'enregistrent auprès de ce registrar sont réifiés en tant qu'objets de l'interface SIPDevice dans le registre avec la mention de leurs formats multimédias supportés. Ces objets référencent les équipements avec lesquels l'application peut créer des sessions et contrôler les flux. Le SIP Binding Manager et le gestionnaire de contexte sont enregistrés en tant que SIPListener afin de connaître les appels effectués par ces équipements selon le patron Tableau Blanc (cf. Chapitre V.2.4.2). La Figure 81 montre le fonctionnement du pilote SIP sur a plateforme OSGi.

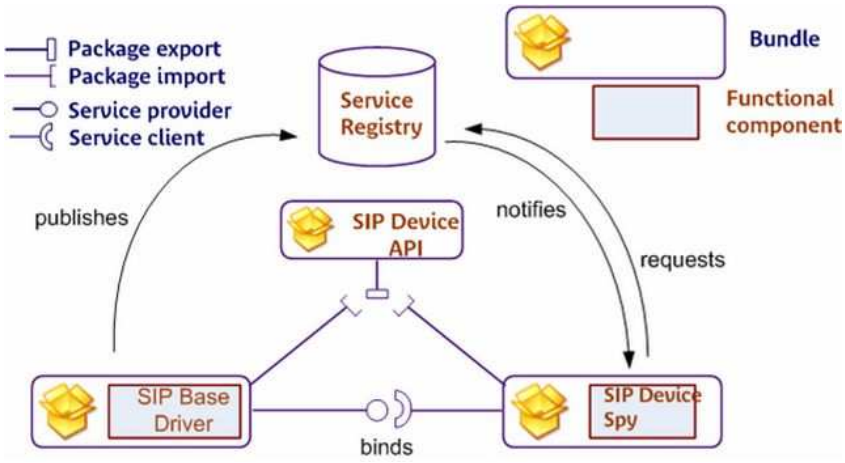


Figure 81 Pilote orienté service SIP (SIP Base Driver)

5.3 Négociation de flux multimédias et capacités de transcodage

Une entité essentielle de l'architecture est l'adaptateur multimédia (Media Adaptation Framework de la Figure 80). L'adaptateur multimédia fournit la fonction indispensable de transcodage pour adresser l'hétérogénéité des protocoles de flux et des formats de données. Il permet en particulier le transcodage de flux RTP en flux HTTP et inversement, flux respectivement supportés par les équipements SIP et les équipements UPnP. Il répond par conséquent à l'exigence technique mentionnée dans le Chapitre II.4.3.2.

Ce transcodeur est contrôlé par le SIP Binding Manager qui le place en tant que proxy entre sources et puits multimédia aux protocoles et formats distincts. Comme le transcodage de flux audio-vidéo est très consommateur de ressources, cette entité est forcément placée sur un équipement riche en ressources du réseau domestique. Dans l'architecture proposée, le transcodeur est placé sur un équipement distinct de la plateforme

qui héberge l'application principale. L'entité nommée "Remote Media Framework Access" de la Figure 80 agit en tant que pilote pour le contrôle du transcodeur.

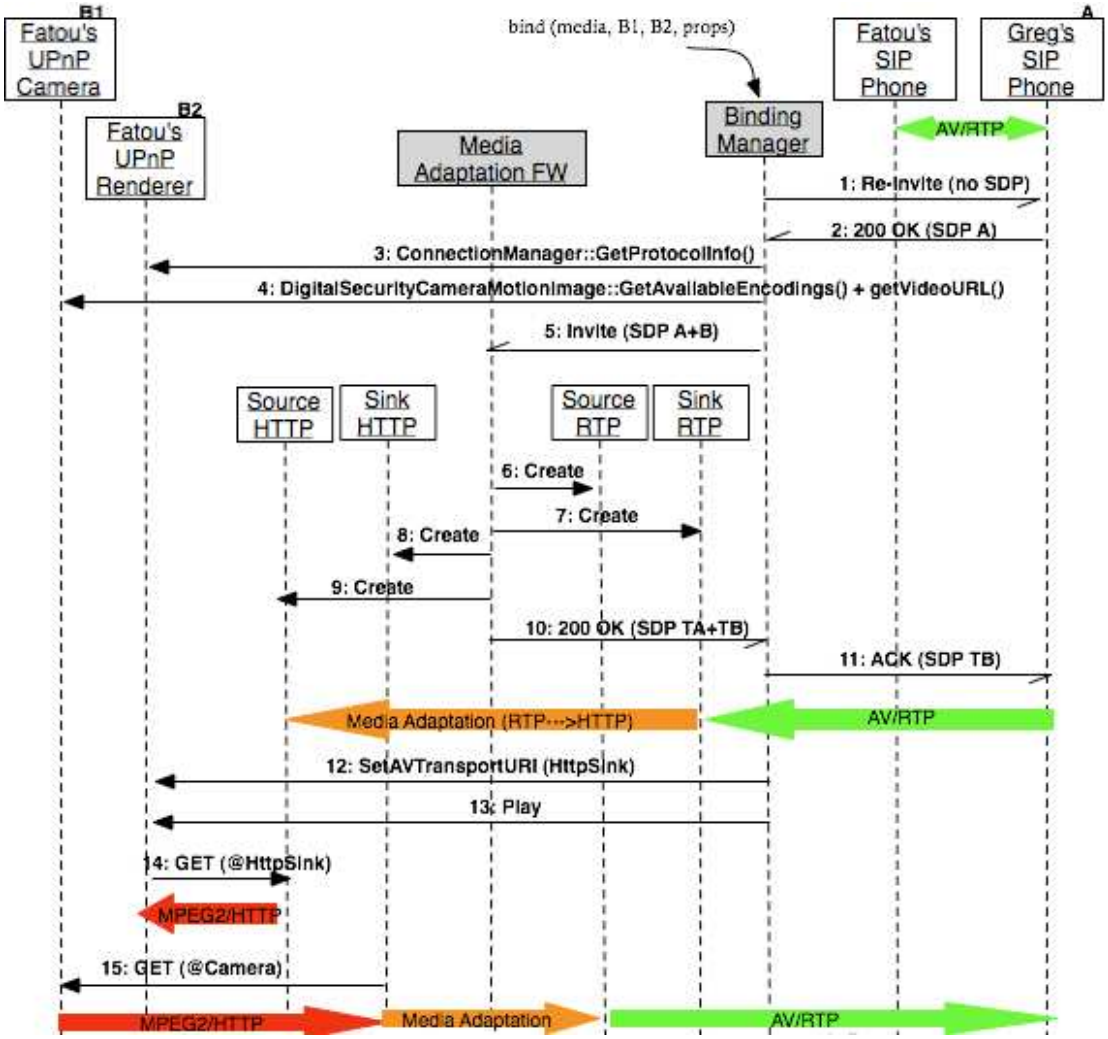


Figure 82 Interactions entre équipements SIP, UPnP et transcodeur

La Figure 82 montre comment le contrôle du transcodeur au travers du protocole SIP (RFC 4117, cf. Chapitre III.2.1.2) est allié, dans l'architecture, à la technique du 3PCC (Third-Party Call Control, IETF RFC 3725, cf. Chapitre III.2.1.2). Initialement, le téléphone de Fatou est en cours de communication avec celui de Greg dans le protocole AV/RTP. Fatou souhaite montrer l'image provenant de la caméra du salon (UPnP Camera) à Greg et afficher l'image fournie par le téléphone de Greg (A) sur la télévision (UPnP Renderer). Le SIP Binding Manager envoie une invite à A (1) qui répond avec la description de ses capacités (2). Il demande alors les capacités de la télévision (3) et de la caméra (4). Comme les capacités du téléphone de Greg et celles des équipements de Fatou ne sont pas compatibles, il envoie un message au transcodeur en précisant les capacités des équipements à mettre en relation (5). Le transcodeur crée alors des sources et des puits compatibles avec les équipements de part et d'autre (6, 7, 8, 9) et répond au Binding Manager les descriptions de ces entités créées (10). D'un côté, le Binding Manager répond alors à A l'adresse du puits RTP vers lequel A doit pousser le flux vidéo tandis qu'il accepte la demande de pousser un flux vidéo de la source RTP vers A (11). De l'autre côté, le Binding Manager demande à la télévision de jouer le flux qui provient de la source HTTP (12, 13, 14) dans l'algorithme UPnP standard. Le Binding Manager demande au puits HTTP de tirer le flux HTTP de la caméra UPnP (15). Grace au transcodage de puits RTP à source HTTP et de puits HTTP vers

source RTP, mis en place à l'instanciation de ces objets, les flux vidéo s'échangent de bout en bout entre les équipements de Fatou et le téléphone de Greg.

6 UPnP-IGRS : COEXISTENCE ET INTEROPERABILITE

L'étude particulière d'IGRS (Internet Grouping and Resource Sharing, cf. Chapitre III.1.1.2) a été demandée opportunément par un représentant de France Télécom au comité ISO/IEC face à la proposition de l'ensemble protocolaire en normalisation ISO/IEC par les IGRS Labs – www.igrs.org – en 2006. Elle a conduit à la demande de modifications [2006-7] acceptée par les IGRS Labs. La modification principale permet aux équipements UPnP et IGRS de coexister sur les réseaux locaux.

6.1 Comparaisons des ensembles protocolaires UPnP et IGRS

IGRS reprend plusieurs protocoles choisis par UPnP : SSDP pour la découverte, SOAP pour le format des messages de contrôle, GENA pour la souscription et la notification d'événements. Toutefois, tous les messages de communication spécifiés par ces protocoles sont modifiés dans IGRS, rendant non-interopérables des protocoles de même nom.

Aussi, IGRS utilise différents protocoles pour les couches de description et de sécurité que n'emploie pas UPnP. WSDL est choisi pour la description des services IGRS alors qu'UPnP se base sur un standard ad hoc UPnP Template Language. Les protocoles de sécurité d'IGRS sont des standards chinois secrets alors qu'UPnP utilise SSL. De plus, IGRS incorpore des notions de sécurité dans toutes les couches protocolaires alors qu'UPnP parvient à séparer la spécification des mécanismes de sécurité, appelés UPnP Security Ceremonies, des autres aspects – découverte, description, contrôle.

Enfin, IGRS définit des couches supplémentaires de communication par "pipe" et de gestion de session que ne possède pas UPnP. La communication par "pipe" spécifie des moyens de communication avec des équipements sur des bus de terrain non-IP. La gestion de session permet une négociation du format des échanges avant que des messages de contrôle puissent être échangés. Ces atouts d'IGRS sont des extensions que ne peut supporter UPnP.

6.2 Non-interopérabilité et interférences

Comme tous les messages de découverte, de contrôle synchrone, de souscription et de notification sont modifiés par IGRS et que les autres couches protocolaires d'IGRS diffèrent de celles d'UPnP, les deux ensembles protocolaires ne sont pas interopérables.

Le protocole de découverte SSDP repris par IGRS présente toutefois lors de sa proposition en Comité ISO en avril 2006 la même adresse IP multicast que celle d'UPnP. Le partage de cette adresse fait interférer les équipements UPnP et les équipements IGRS alors qu'ils ne peuvent pas se comprendre. En effet, les équipements UPnP entendent les messages de découverte IGRS sur l'adresse 239.255.255.250:1900 sans pouvoir les traiter et vice versa.

6.3 Deux issues possibles

Cette analyse des protocoles IGRS en connaissance de l'ensemble UPnP mène à la proposition de deux solutions possibles :

- Pour l'interopérabilité complète (ou quasi-complète) des deux ensembles protocolaires : IGRS doit premièrement remplacer son usage des protocoles SSDP, SOAP et GENA par l'usage de ces protocoles par UPnP. Les mentions de sécurité dans tous les messages de découverte, de description, de contrôle synchrone, de souscription et de notification disparaissent avec ces

changements. L'emploi de WSDL pour la description doit être remplacé par l'UPnP Template Language. Le protocole SSL doit être choisi à la place du protocole chinois. IGRS propose alors une extension d'UPnP pour les équipements non IP (pipe communication) et la gestion de sessions de communication.

- Pour la coexistence simple sans interférence des deux ensembles protocolaires : IGRS doit demander une nouvelle adresse multicast à l'organisation IANA et spécifier cette adresse pour les requêtes multicast actives et les annonces de départ et d'arrivée de pilotes. De cette façon, les équipements des deux types n'interfèrent plus au niveau de la couche découverte et n'interagiront jamais au niveau des autres couches a fortiori.

La première option demandait un réalignement laborieux de la spécification IGRS sur la spécification UPnP. C'est naturellement la 2ème option malgré son caractère négatif qui a été effectué par le consortium chinois. Les équipements UPnP et IGRS peuvent donc coexister sans interopérer comme le font les équipements obéissant à d'autres protocoles IP plug-n-play tels que DPWS, iTunes, SLP, Jini.

7 SPECIFICATION UPNP DEVICE MANAGEMENT

La contribution la plus récente de cette thèse vise la gestion de l'hétérogénéité des plateformes d'exécution dans le domaine précis de l'administration des équipements du réseau domestique. Elle se concrétise dans un effort de normalisation auprès du Forum UPnP [2007-9] – www.upnp.org – qui continue aujourd'hui dans la codirection de l'UPnP Device Management (ex-Execution Platform) Working Committee (Co-directeurs : l'auteur pour le Groupe France Telecom et Jooyeol Lee pour Samsung). Un RFP accepté par l'Alliance OSGi [2007-8] est le miroir de ce travail pour la plateforme d'exécution OSGi. Une réponse (RFC) à ce RFP est en cours d'élaboration par les membres de l'OSGi Residential Expert Group. L'effort de normalisation UPnP a aussi des influences dans le Broadband Forum[1], la Home Gateway Initiative et le DLNA (Digital Living Network Alliance).

Le Comité UPnP DM (Device Management) spécifie un profil d'équipement appelé ManageableDevice hébergeant trois services : le SoftwareManagement Service offre les fonctionnalités de gestion de cycle de vie logiciel, le ConfigurationManagement Service fournit les fonctionnalités de configuration, le DeviceMaintenance Service permet le diagnostique et la supervision de performances de l'équipement. Cette thèse décrit essentiellement la contribution concernant la gestion de cycle de vie logiciel, contribution[2] effectuée au nom de France Telecom. Cette section présente les détails de l'application des concepts présentés dans la section Chapitre V.6 sur le modèle d'interaction UPnP.

7.1 Ensemble d'actions

Un service UPnP est décrit par un ensemble d'actions et de variables d'état. Voici la liste des actions proposée dans la contribution du 21 juillet 2008 pour le SoftwareManagement Service. Les actions sont présentées dans la syntaxe définie par le Forum UPnP. Huit actions de déploiement sont définies : Download(), Remove(), SetUp(), Install(), Update(), Uninstall(), Start(), Stop(). Une action d'interrogation d'état est associée à

[1] Proposition n°dsl2008.689.00 "Proposal for Enhanced Application Management via TR-069" présentée par H. Kirksey, Motive, et tirée d'une présentation de Levent Gurgen et de l'auteur.

[2] Email publié le 21 juillet 2008 sur la liste du Comité

chaque type d'entité logicielle : GetDPState(), GetDUState(), GetEUState(). Quatre actions d'interrogations de l'état des listes d'entités sont spécifiées : GetDPIDs(), GetDUIDs(), GetEUIDs(), GetActiveEUIDs(). Les huit actions de déploiement initient des opérations atomiques et isolées de déploiement. Chacune des actions renvoie un identifiant d'opération. Une action d'interrogation de l'état d'une opération, GetOperationState(), et une action d'interrogation de la liste des opérations en cours, GetOperationIDs() sont définies.

Voici le détail des actions sur les paquetages de déploiement (DPs) :

- Download(in uri DPURI, in boolean HandleDependencies, out ui4 OperationID) : initie le téléchargement d'un DP à partir d'une adresse au format Universal Resource Identifier (DPURI). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre DPs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances manquantes soient aussi téléchargées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse.
- Remove(in ui4 DPID, in boolean HandleDependencies, out ui4 OperationID) : initie le retrait d'un DP à partir d'un identifiant (DPID). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre DPs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances non utilisées soient aussi retirées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse.
- SetUp(in uri DPURI, in string OptionalParameters, in boolean HandleDependencies, out ui4 OperationID) : initie l'installation du contenu d'un DP à partir d'un identifiant (DPID). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre DPs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances soient aussi installées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse.
- GetDPState(in ui4 DPID, out string DPState) : interroge l'état (DPState) d'un DP identifié par son identifiant DPID. La valeur de l'état retournée peut être DOWNLOADING, DOWNLOADED, REMOVING, REMOVED.
- GetDPIDs(out string DPIDs) : retourne la liste des DPs dans l'état DOWNLOADED.

Voici le détail des actions sur les unités de déploiement (DUs) :

- Install(in uri DUURI, in string OptionalParameters, in boolean HandleDependencies, out ui4 OperationID) : initie l'installation d'un DU à partir d'une adresse au format Universal Resource Identifier (DUURI). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre DUs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances soient aussi installées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse. La chaine de caractères d'entrée OptionalParameters permet de spécifier des paramètres supplémentaires spécifiques à la plateforme visée, par exemple, l'indication d'un AppDomain où installer un Assembly .NET ou encore l'indication d'un fichier de configuration sur la plateforme Linux.
- Update(in ui4 DUID, in uri NewDUURI, in boolean HandleDependencies, out ui4 OperationID) : initie la mise à jour d'un DU à partir d'un identifiant (DUID). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre DUs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances soient aussi mises à jour. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse. Le paramètre

d'entrée NewDUURI permet d'indiquer que la nouvelle version souhaitée du DU se trouve à une adresse (URI) différente.

- Uninstall(in ui4 DUID, in boolean HandleDependencies, out ui4 OperationID) : initie la désinstallation d'un DU à partir d'un identifiant (DUID). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre DUs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances non utilisées soient aussi désinstallées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse.
- GetDUState(in ui4 DUID, out string DUState) : interroge l'état (DUState) d'un DU identifié par son identifiant DUID. La valeur de l'état retournée peut être INSTALLING, INSTALLED, RESOLVED, UNINSTALLING, UNINSTALLED.
- GetDUIDs(out string DUIDs) : retourne la liste des DUs dans l'état INSTALLED.

Voici le détail des actions sur les unités d'exécution (EUs) :

- Start(in ui4 EUID, in string OptionalParameters, in boolean HandleDependencies, out ui4 OperationID) : initie le démarrage d'un EU à partir d'un identifiant (EUID). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre EUs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances soient aussi démarrées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse. La chaine de caractères d'entrée OptionalParameters permet de spécifier des paramètres supplémentaires spécifiques à la plateforme et l'application visée, par exemple, l'indication de paramètres d'entrée pour l'invocation de la méthode principale d'exécution d'un programme.
- Stop(in ui4 EUID, in boolean HandleDependencies, out ui4 OperationID) : initie l'arrêt d'un EU à partir d'un identifiant (EUID). La valeur booléenne HandleDependencies indique le souhait que les dépendances entre EUs soient gérées par la plateforme elle-même, c'est-à-dire que les dépendances soient aussi arrétées. L'identifiant d'opération OperationID est généré par l'équipement et retourné en réponse.
- GetEUState(in ui4 EUID, out string EUState) : interroge l'état (EUState) d'un EU identifié par son identifiant EUID. La valeur de l'état retournée peut être STARTING, STARTED, STOPPING, STOPPED.
- GetEUIDs(out string EUIDs) : retourne la liste des EUs existants.
- GetActiveEUIDs(out string ActiveEUIDs) : retourne la liste des EUs dans l'état ACTIVE.

Voici le détail des actions d'interrogation sur l'état des opérations :

- GetOperationState(in ui4 OperationID, out string OperationState, out ui4 UnitID) : interroge l'état (OperationState) d'une opération identifiée par son identifiant OperationID. La valeur de l'état retournée peut être IN_PROGRESS, COMPLETED, ERROR. L'identifiant de l'entité logicielle visée par l'opération est retourné, s'il est disponible, par la valeur UnitID.
- GetOperationIDs(out string OperationIDs) : retourne la liste des operations dans l'état IN_PROGRESS.

7.2 Notification d'événements

Le modèle UPnP définit une notification d'événements liés à des changements de variables d'états. Voici la liste des variables d'état dites notifiables ("evented").

- OperationIDs : la liste des identifiants d'opérations dans l'état IN_PROGRESS. L'ajout d'un identifiant dans la liste suit sa génération durant le traitement d'une des huit actions de déploiement. L'ajout indique donc le début de l'opération, le retrait indique l'achèvement de l'opération. Lors de ces changements, la valeur de la liste est notifiée par le mécanisme d'événement UPnP.
- DPIDs : la liste des DPs dans l'état DOWNLOADED. La liste peut changer lors de l'achèvement d'opérations sur les DPs. Sa valeur est alors notifiée avec la notification de la valeur de la liste OperationIDs.
- DUIDs : la liste des DUs dans l'état INSTALLED. La liste peut changer lors de l'achèvement d'opérations sur les DPs (SetUp()) et des opérations sur les DUs. Sa valeur est alors notifiée dans le même message qui notifie la valeur de la liste OperationIDs.
- EUIDs : la liste de tous les EUs existants. La liste peut changer lors de l'achèvement d'opérations sur les DPs (SetUp()), des opérations sur les DUs. Sa valeur est alors notifiée dans le même message qui notifie la valeur de la liste OperationIDs.
- ActiveEUIDs : la liste des EUs dans l'état ACTIVE. La liste peut changer lors de l'achèvement d'opérations sur les DPs (SetUp()), des opérations sur les DUs. Sa valeur est alors notifiée dans le même message qui notifie la valeur de la liste OperationIDs.

7.3 Gestion partagée des dépendances

L'argument booléen HandleDependencies des huit actions de déploiement permet au Point de Contrôle d'indiquer à l'équipement contrôlé le souhait que ce dernier gère lui-même les dépendances interactives. Dans les deux cas, l'équipement reste le décideur final. Si la plateforme n'est pas capable de gérer les dépendances dans des situations délicates, par exemple la désinstallation d'unités, elle peut refuser cette gestion. A l'inverse, la plateforme peut aussi refuser d'effectuer une opération sans gérer les dépendances elle-même pour des raisons de cohérence du logiciel installé.

7.4 Opérations asynchrones, atomicité et isolation

Un argument supplémentaire est apporté par l'architecture UPnP [UPnP06] en faveur d'une gestion asynchrone des opérations de déploiement. En effet, les actions UPnP sont des "opérations" du type request-response de la nomenclature de la section Chapitre II.2.4 et la spécification générale d'UPnP spécifie que la réponse à une action ne peut être envoyée plus de 30 secondes après la requête. Comme les opérations de déploiement sont à même de durer plus longtemps, les actions UPnP simples ne peuvent pas correspondre directement à ces opérations.

L'asynchronisme nécessaire amène à définir une opération de déploiement comme une opération initiée par une des huit actions de déploiement décrites ci-dessus et s'achevant par la notification de la liste OperationsIDs. L'action de déploiement initiant l'opération place, si possible, les entités impliquées dans l'état transitoire adéquat et retourne immédiatement un identifiant OperationID généré par l'équipement UPnP. L'opération est alors dans l'état IN_PROGRESS. Dès que toutes les instructions de l'opération sont achevées

avec succès, les entités impliquées sont déclarées dans l'état stable adéquat et l'opération entre dans l'état COMPLETED. Si une des instructions ne peut réussir, toutes les entités impliquées doivent revenir dans l'état stable précédent (rollback) et aucune liste d'identifiants parmi les listes DPIDs, DUIDs, EUIDs, ActiveEUIDs ne change.

Dans les deux cas de réussite et d'échec, l'OperationID est par conséquent retiré de la liste OperationIDs. Le changement de la valeur de la variable d'état OperationIDs et de la valeur d'autres listes – parmi les listes DPIDs, DUIDs, EUIDs, ActiveEUIDs – est notifié dans un même message à tous les abonnés à la notification de ce service.

Les états transitoires DOWNLOADING, REMOVING, INSTALLING, UNINSTALLING, STARTING, STOPPING indiquent que les entités logicielles sont impliquées dans des opérations de déploiement. Afin que ces verrous pris sur les ressources soient visibles, il est souhaité que toutes les entités impliquées soient dans l'état transitoire adéquat avant que la réponse à l'action initiant l'opération ne soit retournée au Point de Contrôle.

7.5 Implémentation sur la plateforme OSGi

La Figure 83 montre l'architecture logicielle mise en œuvre pour la représentation de la plateforme OSGi en tant qu'UPnP ManageableDevice. L'UPnP Device Generator de la section 4.3 est utilisé pour la génération du "ManageableDevice Driver". Le bundle "Management client" fournit une implémentation directe des interfaces générées. L'implémentation utilise les interfaces standards de gestion des modules OSGi afin d'installer, désinstaller, mettre à jour, démarrer et arrêter les bundles selon la correspondance entre bundles et les concepts de DU et EU de la section Chapitre III.3.3.1. Le téléchargement et le retrait de bundles dans un cache local est effectué grâce aux interfaces Java.net standards. Aucune implémentation des Deployment Packages ne semble disponible et ces entités se remplacent aisément par des bundles déclarant des dépendances vers des paquetages et des services fournis par d'autres bundles d'une partie applicative cohérente.

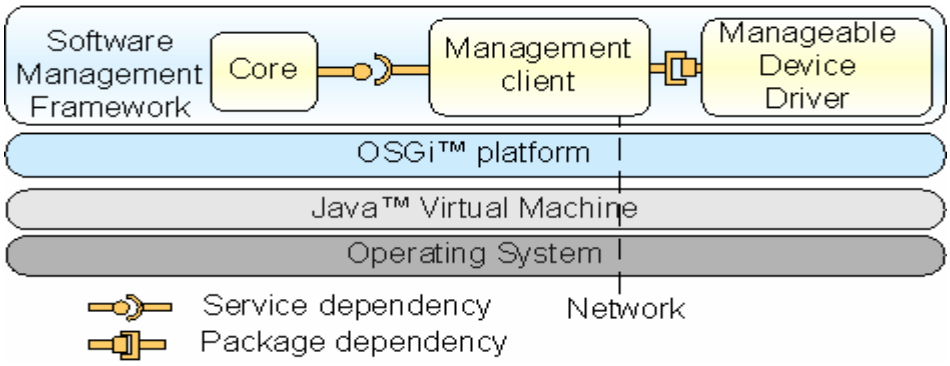


Figure 83 Architecture pour le contrôle UPnP DM de la plateforme OSGi

La gestion des dépendances à l'installation est effectuée grâce à la spécification de l'OBR (OSGi Bundle Repository) et au client d'installation disponible dans le projet Apache felix. L'OBR définit une syntaxe générique de dépendances entre bundles. Les dépendances peuvent être de nature diverses. L'implémentation d'Apache felix gère au moins les dépendances de packages versionnés de code Java à l'installation. Il peut être aisément étendu avec la gestion des dépendances de services OSGi. En revanche, la gestion des dépendances à la mise à jour et à la désinstallation demeurent une extension complexe à réaliser.

Voici les trois parties principales de l'application :

- **Core** : ce module utilise les méthodes de gestion de bundles de la plateforme OSGi et les rend principalement asynchrones suivant le modèle UPnP DM.

- **Management client** : ce module implémente les paquetages générés du Manageable Device Driver. Ce pilote raffiné présente un service de type ManageableDevice étendant le type UPnPDevice reconnu par l'UPnP Base Driver (cf. section 1.5) qui exporte l'interface sur le réseau. L'implémentation utilise le service présenté par le module "Core".
- **Loader** : cette partie est fournie par le cadriciel OSGi. Elle est déjà spécifiée par l'Alliance OSGi et implémentée par des plateformes telles que celle du projet open source Apache felix. Elle fournit les interfaces de déploiement de bundles.

7.6 Implémentation sur la plateforme .NET

La Figure 84 montre l'architecture logicielle mise en œuvre pour la représentation de la plateforme .NET en tant qu'UPnP ManageableDevice. L'implémentation utilise les interfaces standards de gestion des assemblys .NET afin d'installer, mettre à jour, démarrer les assemblys selon la correspondance entre bundles et les concepts de DU et EU de la section Chapitre III.3.3.4. Le téléchargement et la désinstallation d'assemblys dans le "Global Assembly Cache" sont réalisés.

Le démarrage des EUs est implémenté comme l'invocation des assemblys avec les paramètres passés dans l'argument OptionalParameters (cf. section 7.1). La méthode d'arrêt des EUs n'a pas de correspondance directe.

La plateforme .NET montre un défaut au moment pour le déchargement d'un assembly. En effet .NET ne permet que le déchargement d'un domaine d'application entier. Il est alors possible de choisir entre ne pas implémenter l'action de désinstallation de DUs ou l'implémenter par la désinstallation du Domaine d'Application entier et par son rechargement avec l'assembly visé en moins. La première option semble être la plus raisonnable.

Les mécanismes d'isolation stricts sont implantés sur la plateforme .NET entre Domaines d'application. L'implémentation de l'équipement UPnP ManageableDevice pour la plateforme .NET permet d'installer, de mettre à jour et de retirer un assembly sur chacun des Domaines. Pour cela, l'implémentation utilise l'argument OptionalParameters afin de passer le nom du Domaine d'Application. Ainsi l'administrateur peut ajouter lors de l'installation le nom de domaine dans lequel l'assembly sera installé. La plateforme installera donc le module dans ce domaine en ignorant la résolution automatique des dépendances. Cette fonctionnalité est importante pour laisser la flexibilité nécessaire à l'administrateur pour déployer l'application dans un ou plusieurs domaines d'application.

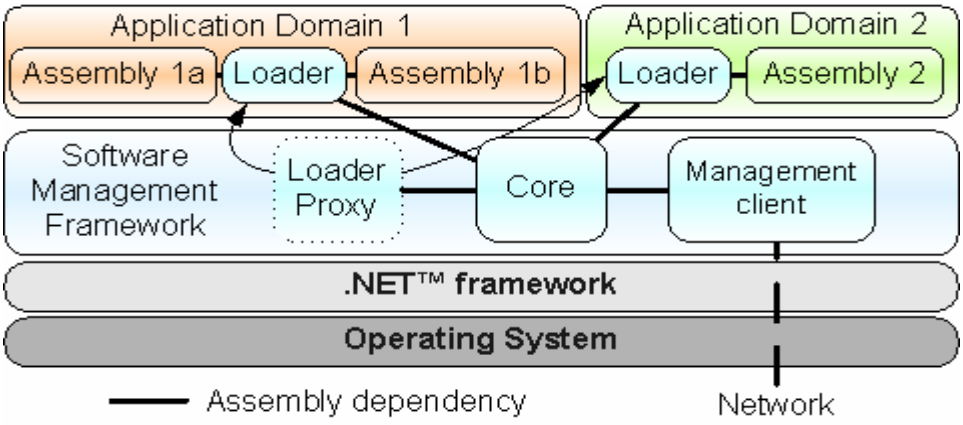


Figure 84 Architecture pour le contrôle UPnP DM de la plateforme .NET

Voici les trois parties principales de l'application (cf. Figure 84) :

- **Core** : ce module gère les domaines d'application et les assemblys chargés.

- **Management client** : ce module fait l'interface entre le réseau UPnP et le module Core. La plupart des classes ont été générées automatiquement grâce à l'outil d'Intel similaire à l'UPnP Device Generator développé à France Telecom. L'outil d'Intel ne gère toutefois pas les événements et le patron de conception "plateforme de services".
- **Loader** : ce module a le rôle le plus complexe. Il permet de charger des assemblys dans tout Domaine d'Application indiqué par une requête. Il crée le Domaine d'Application s'il n'existe pas. La structure de .NET rend nécessaire de charger dans chaque domaine d'application un module spécifique (le Loader) qui offre une interface d'administration au Core via les mécanismes de communication .NET Remoting entre Domaines d'Application. Ainsi une instance de ce module est créée dans chaque domaine d'application géré.

7.7 Implémentation sur la plateforme Linux Ubuntu

Si le package est une unité de déploiement bien définie, Linux ne définit par d'entités uniques comme unités d'exécution. Les concepts sont bien séparés et le noyau linux ne gère par communément le déploiement et l'exécution de modules (cf. Chapitre III.3.3.3).

L'expérimentation sur Linux Ubuntu reprend les modules du logiciel APT pour la gestion des packages. APT permet à un utilisateur d'installer, de mettre à jour et de désinstaller des packages sur sa distribution Linux avec des modules disponibles sur un répertoire distant. Cet outil gère de surcroît les dépendances entre les différents paquetages. L'expérimentation de la distribution Linux Ubuntu ne restreint pas le champ d'application du prototype puisque les deux familles principales de distributions Debian et RedHat utilisent un système équivalent de gestion de paquetages.

Les unités d'exécution sont de différents types. Les démons Linux présentent un cycle de vie proche de celui présenté par les Unités d'exécution du modèle UPnP DM. Ce sont des programmes contrôlés par un script (RC) qui possède deux fonctions principales : le lancement (start) et l'arrêt (stop) du service. L'inconvénient est que les démons ne sont pas réellement des programmes utilisateurs courants, il s'agit principalement de taches passives qui s'exécutent en arrière plan du système d'exploitation (par exemple un serveur http). Ainsi un second type d'unité d'exécution est nécessaire. Le lancement des fichiers exécutables Linux est similaire à celui des assemblys où des paramètres d'invocation sont à spécifier au lancement. Toutefois, dans un paquetage classique de la distribution d'un programme, ils sont difficilement discernables des autres fichiers de l'application. Ainsi pour connaître les fichiers exécutables présents dans le paquetage, seul l'administrateur qui installe l'application est capable de renseigner la plateforme. Pour ce faire, il est prévu d'utiliser les paramètres optionnels de l'action d'installation afin de spécifier des fichiers exécutables assimilables à des unités d'exécution. Comme pour les assemblys, les paramètres optionnels de l'action de démarrage seront aussi utilisés pour invoquer ces unités d'exécution.

La décomposition de la plateforme est à nouveau proposée selon trois modules principaux (voir Figure 85) :

- **Core** : le module principal du prototype reçoit les requêtes depuis les modules UPnP ou console et se charge de les exécuter. Il délègue au module Loader le soin de gérer le cycle de vie des packages, processus et démons
- **Management client** : ce module est construit avec du code obtenu avec l'outil d'Intel. Il fournit un ensemble de fonctions qui gèrent automatiquement les interactions avec le réseau UPnP. Ce module traduit une commande UPnP en l'appel de méthode correspondant sur le Core.

- **Loader** : Il est premièrement responsable de la gestion des paquetages Linux. Il est constitué de modules d'Aptitude (APT) modifiés. Ce module utilise deuxièmement l'API du système d'exploitation qui fournit des méthodes afin d'obtenir des informations sur les processus.

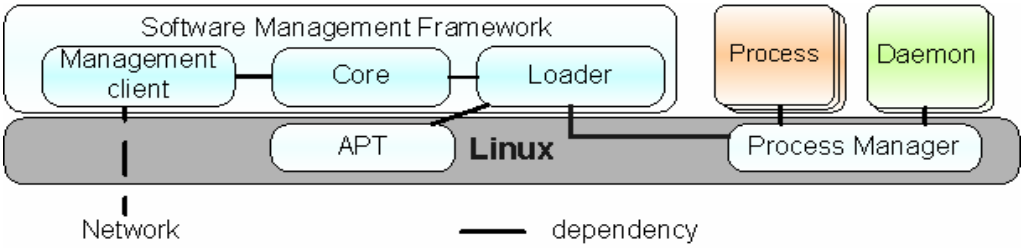


Figure 85 Architecture pour le contrôle UPnP DM de la plateforme Linux Ubuntu

8 SYNTHESE

Le cadriciel Home SOA est bâti principalement au-dessus du standard de plateforme OSGi, promeut certaines techniques avancées utilisées par la communauté OSGi et affine certaines techniques de manière originale. L'implantation des patrons de conception dans plusieurs prototypes montrent leur pertinence dans des situations diverses. Les différentes démonstrations effectuées montrent l'intérêt du Home SOA pour une composition simple des avantages de chaque protocole dans l'édification de logiciels mixant différents domaines d'application.

Les détails de la projection des concepts d'interfaces uniformes d'administration du cycle de vie logiciel sur le protocole UPnP occupe la deuxième partie de ce chapitre. La contribution au Comité UPnP Device Management contourne les limitations d'UPnP et profite de ses atouts pour une découverte et un contrôle dynamique des plateformes d'exécution existantes. L'asynchronisme du modèle est implanté par utilisation des actions synchrones et de la notification d'événements d'UPnP. Les événements concernant les résultats d'opérations de déploiement sont liés à des ajouts et des retraits dans des listes d'entités logicielles. Cette spécification UPnP est suffisamment ouverte pour être implantée au-dessus de plusieurs plateformes existantes telles que Linux Ubuntu, .NET et OSGi.

Chapitre VII. VALIDATION

1 TESTS DU DPWS BASE DRIVER

Les tests de cette section visent à mesurer l'impact des techniques du Home SOA dans la conception de services DPWS. Ils comparent la réalisation d'applications réalisées avec le DPWS Base Driver au-dessus de la plateforme OSGi et celle d'applications réalisées avec la pile Java DPWS nue sans l'enrobage technique Home SOA.

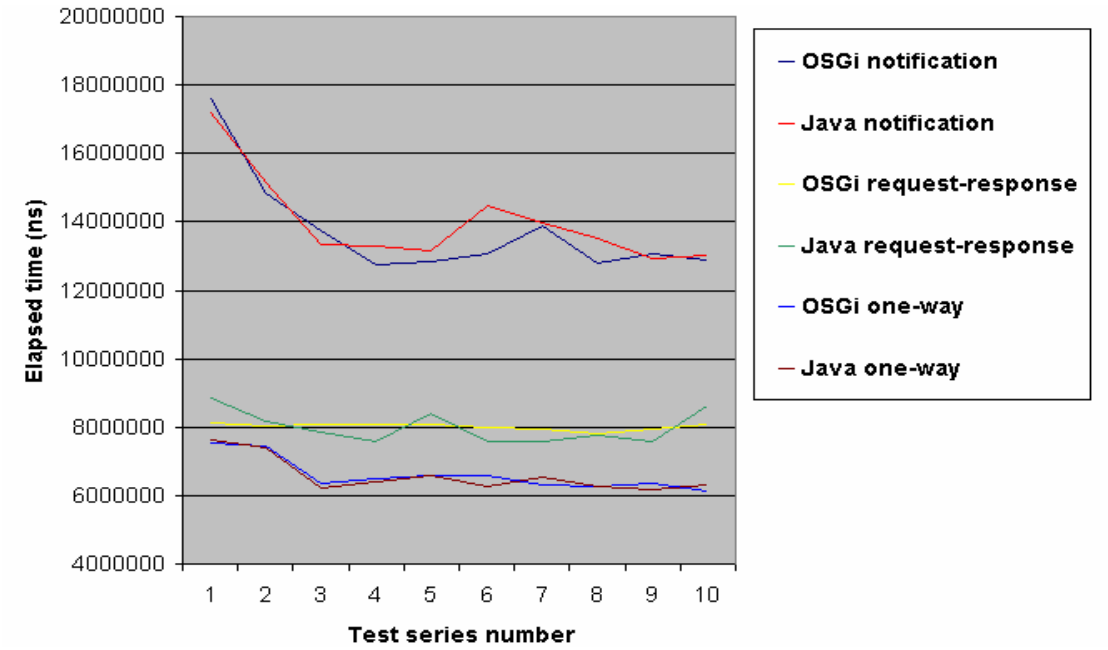


Figure 86 Tests de performance du DPWS Base Driver

Un scenario de test consiste ici en une série de 1000 invocations sur une opération d'une lampe DPWS simulée. Le scénario de test est effectué avec un même client DPWS réalisé avec la pile Java DPWS du projet SODA (www.soda-itea.org) sans enrobage par les techniques du Home SOA sur deux configurations différentes du service DPWS :

- 1ère configuration "Java" : le service est réalisé avec la pile Java DPWS nue.
- 2ème configuration "OSGi" : le service est réalisé avec le DPWS Base Driver au-dessus de la plateforme OSGi Apache felix. Le DPWS Base Driver enrobe la pile Java DPWS de la première configuration.

Dans les deux cas, le même client est utilisé sur la même machine. Le serveur DPWS de chaque configuration est installé sur une même machine, une machine distincte de celle du client. Trois tests sont mis en œuvre :

- test "notification" : le client souscrit à une source DPWS d'événement – l'état de la lampe simulée – et invoque 1000 fois l'opération générant l'événement – opération d'allumage de la lampe. Le client attend le message de notification après chaque invocation d'opération avant de répéter l'invocation.
- test "request-response" : le client invoque 1000 fois une opération de type request-response – opération d'interrogation de l'état de la lampe. Il attend la réponse de l'invocation avant de répéter l'invocation.
- Test "one-way" : le client invoque 1000 fois une opération de type one-way – opération d'allumage de la lampe.

Chaque test est effectué 10 fois (cf. Figure 86) et une moyenne olympique est calculée pour chaque type de test : le résultat des 2 tests les plus lents, souvent correspondant à l'étape de chargement mémoire ("warm-up"), sont supprimés. Le résultatsdes 2 tests les plus rapides est aussi supprimé pour le calcul. La Figure 87 rend les résultats dans un diagramme en bâtons.

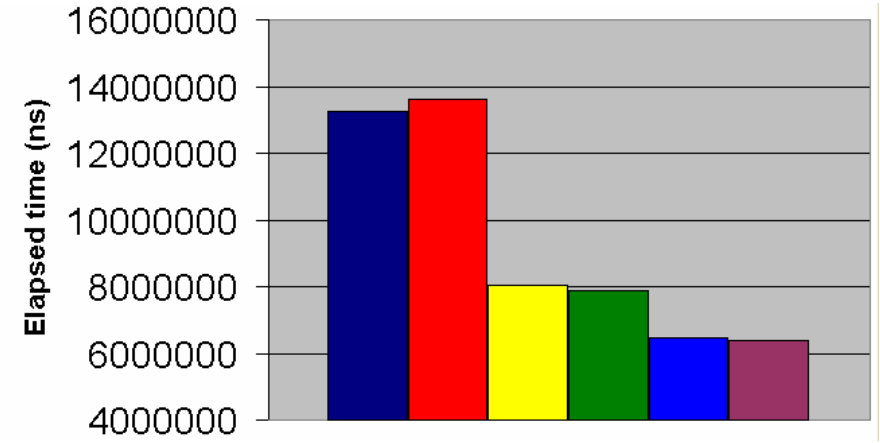


Figure 87 Moyennes olympiques des temps d'exécution des tests

Les mêmes tests ont été effectués sur un client réalisé avec le DPWS Base Driver. Si les détails ne sont pas illustrés ici, le résultat global est chiffré et analysé dans la section 1.2.

1.1 Configuration matérielle

Les tests sont réalisés sur deux machines aux propriétés identiques : la configuration des machines de développement est la même sur les 2 PC : processeur Pentium IV 2,5 GHz, 1Go RAM, Carte réseau Gigabit. Les tests sont effectués avec la machine virtuelle Sun Java SE 1.4 au-dessus du serveur d'exploitation Windows XP.

1.2 Analyse des résultats

Dans le pire des cas, l'accès aux équipements DPWS est 1,88% plus lent dans la configuration OSGi que dans l'autre configuration. Les tests d'un client réalisé avec le DPWS Base Driver au-dessus de la plateforme OSGi Apache felix montrent des résultats similaires. Dans le pire des cas, le client Home SOA est 1,6% plus lent que le client DPWS simple.

De surcroit, le meilleur cas où l'équipement et le client sont co-localisés sur la même plateforme est décrit dans la section 2. Grâce à l'utilisation du patron de conception

"plateforme de services", les appels dans le langage programmatique remplace les appels réseaux. Les coûts de sérialisation et de désérialisation effectuées dans les pilotes orientés services sont alors supprimés et les temps de communication sont beaucoup plus faibles.

Le surcoût apporté par les techniques du Home SOA est raisonnable en comparaison des avantages apportés au développement. Les bénéfices sont la simplification de la découverte dynamique des équipements, la simplicité du maintien de la souscription d'événements par utilisation du tableau blanc, de la flexibilité apportée par la modularité du pilote et de la mutualisation de modules applicatifs tels que le service HTTP, le service de journalisation, le service de configuration, le registre de services, le service d'événement de la plateforme. La simplicité de la découverte dynamique d'équipements provient du passage par les interfaces communes du registre de services. Non seulement la logique métier est décorrélée des préoccupations de répartition mais les conteneurs de type Declarative Services (cf. section Chapitre V.2.3) peuvent alors automatiser la liaison des services.

2 COMPARAISONS DES PILOTES DU CADRICIEL HOME SOA

Les pilotes orientés services et les pilotes raffinés du point de contrôle générique sur le réseau domestique de la section Chapitre VI.4.1 sont testés dans cette section. Les tests visent à chiffrer et à comparer le temps pris par la communication distribuée, les traitements effectués dans les pilotes orientés services et le temps pris par la médiation des pilotes raffinés dans les activités de découverte et de contrôle d'équipements.

Les protocoles pris en compte dans ces tests sont UPnP, Bonjour/iTunes et DPWS. Les pilotes testés sont les suivants :

- Pour UPnP
  - UPnP Base Driver : pilote issu du projet Apache felix et amélioré dans les travaux de cette thèse pour la compatibilité à des équipements existants.
  - SmartUPnP Refined Driver : pilote raffiné représentant les objets UPnPDevice issus du pilote précédent dans les interfaces SmartDevice.
- Pour Bonjour/iTunes
  - Bonjour Base Driver : pilote orienté service reprenant la base de code open source jmdns (jmdns.sourceforge.net) et l'emballant dans un bundle OSGi en suivant les patrons de conception énuméré dans ce mémoire.
  - DMAP (ou iTunes) Refined Driver : pilote raffiné représentant les objets BonjourDevice dans les interfaces associées à la classe DMAPServer. Ce pilote n'a pas été cité dans les chapitres précédents par simplicité. Bonjour n'est qu'un ensemble protocolaire restreint aux couches d'adressage, de découverte et de nommage. Apple iTunes utilise Bonjour et définit le protocole DMAP, pour le contrôle et la notification spécifique à iTunes.
  - SmartBonjour Refined Driver : pilote raffiné adaptant les objets BonjourDevice et DMAPServer dans les interfaces SmartDevice.
- Pour DPWS
  - DPWS Base Driver : pilote orienté service développé durant ces travaux de thèse et délivré en open source au projet Amigo[1]

[1] France Telecom DPWS Base Driver specification, Java API, implementation and examples: http://gforge.inria.fr/frs/?group_id=160&release_id=1804

- SmartDPWS Refined Driver : pilote raffiné représentant les objets DPWSDevice issu du pilote précédent dans les interfaces SmartDevice.

2.1 Configuration matérielle

Trois machines ont été utilisées pour la réalisation de ces tests :

- Deux équipements appelés HomeBox comprenant une carte VIA C7, un processeur cadencé à 1000 MHz, une mémoire vive de 1Go, un noyau linux v2.6.26-1-486 avec le un serveur d'exploitation Debian 4.1.2-23.
- Un PC comprenant un processeur XEON cadencé à 2,20 GHz, une mémoire vive de 1Go, un serveur d'exploitation Windows XP.

Les pilotes Home SOA sont déployés sur la plateforme Apache felix hébergée sur la HomeBox. Pour tous les tests, le PC est utilisé pour collecter les mesures de temps d'exécution par utilisation d'eclipse et du plugin TPTP (version 4.4.1). Ce dernier utilise l'API JVMTI. Les deux machines utilisent une machine virtuelle compatible Java SE 1.5. Elles sont connectées sur un simple réseau local Ethernet au moyen d'un routeur LiveBox.

2.2 Tests des pilotes UPnP

Les tests mettent en œuvre le contrôle d'une lampe UPnP simulée sur le PC et sur la HomeBox. Les temps d'exécution sont relevés sur les opérations suivantes :

- Test "binding" : découverte de la lampe par un client simple sur la HomeBox.
- Test "invocation" : invocation de l'action d'interrogation de l'état de la lampe (action UPnP SwitchPower::GetTarget)

Pour chaque opération la même série de test est effectuée selon trois configurations différentes :

- 1ère configuration sur deux HomeBox : la première contrôle l'équipement simulé sur la deuxième. La lampe simulée est l'exemple du projet Apache felix.
- 2ème configuration avec deux plateformes Home SOA sur une seule HomeBox : la première héberge le client et la deuxième, l'équipement simulé.
- 3ème configuration avec une seule plateforme sur une HomeBox : le client et l'équipement simulé fonctionnent sur la même plateforme Home SOA.

Le Tableau 10 montre les moyennes olympiques des séries de 10 tests dans chaque configuration et chaque type de test.

Tableau 10 Temps d'exécution moyens dans les deux couches de pilotes UPnP

	Base Driver (s)	Refined Driver (s)	Total time (s)	Base Driver (%)	Refined Driver (%)
binding 1	0,105801	0,040277	0,146078	72,43	27,57
binding 3	0.045606	0.069162	0.114767	39.74	60.26
invocation 1	0,294412	0,000865	0,295277	99,71	0,29
invocation 2	0,299627	0,000846	0,300473	99,72	0,28
invocation 3	0,000464	0,001196	0,001660	27,97	72,03

Les différences de temps entre les deux premières configurations montrent les temps pris par la communication réseau entre les machines de la configuration 1. Ils s'avèrent négligeables, les deux machines étant très proches. La différence importante des temps entre situations 2 et 3 met en évidence l'importance du gain lors de la co-localisation des serveurs et clients sur une même plateforme. Le gain correspond à l'absence de sérialisation et de dé-

sérialisation dans l'interaction demandeur-fournisseur dans la situation 3. Ce meilleur cas est un point important pour la conception des pilotes orientés services.

Il est notable que la réification et l'invocation d'objets SmartDevice de la deuxième couche s'effectuent en un temps négligeable en comparaison des traitements effectués par le Base Driver. L'adaptation des Refined Driver a un coût négligeable par rapport au coût de sérialisation-désérialisation du Base Driver.

Le temps de traitement des deux pilotes dans le scénario "binding" est du même ordre de grandeur. Ce temps correspond à l'instanciation d'objet pour la représentation de l'équipement découvert. Cette instanciation d'objets en mémoire est importante mais n'est effectuée que lors de l'apparition de chaque équipement. Ces événements sont généralement moins fréquents que les opérations d'invocation des équipements. Le coût est donc relativement acceptable.

2.3 Tests des pilotes Apple Bonjour/iTunes

Les deux types de test sont effectués avec le logiciel RythmBox du type iTunes dans les deux premières configurations logicielles. La HomeBox découvre et contrôle le logiciel iTunes démarré sur une autre HomeBox dans la 1ère configuration, sur la même HomeBox dans la 2nde configuration. Le logiciel RythmBox n'étant pas Java, une 3ème configuration n'a pas de sens. Le Tableau 11 montre les résultats moyens des tests. La colonne Base Driver concerne le Bonjour Base Driver dans le scénario "binding" et concerne le DMAP Refined Driver dans le scénario "invocation". La colonne Refined Driver ne concerne que les temps d'exécution du SmartBonjour Refined Driver.

Tableau 11 Temps d'exécution moyens dans les deux couches de pilotes Apple

	Base Driver (s)	Refined Driver (s)	Total time (s)	Base Driver (%)	Refined Driver (%)
binding 1	0,082003	0,069600	0,151603	54.09	45.91
invocation 1	0,412695	0,006423	0,419118	98,47	1,53
invocation 2	0,487090	0,009413	0,496503	98,17	1,90

Le coût apporté par la l'adaptation de l'invocation des objets BonjourDevice en une invocation d'objets DMAPServer est de l'ordre de 2% en comparaison du temps total, ce qui est négligeable.

2.4 Tests des pilotes DPWS

Les tests sur les pilotes DPWS complètent ici les tests sur le DPWS Base Driver de la section 1. Le test d'invocation d'une opération (request-response) d'interrogation de l'état de la lampe est choisi ici, et la configuration matérielle est différente. Le temps d'exécution dans le Base Driver (cf. ligne "invocation 1" du Tableau 12) demeure similaire au temps chiffré des tests précédents. La ligne "invocation 2" montre une situation où fournisseur et demandeur sont déployés sur des plateformes différentes sur la même machine. La ligne "invocation 3" correspond à la situation où fournisseur et demandeur partage la même plateforme OSGi.

Tableau 12 Temps d'exécution moyens dans les deux couches de pilotes DPWS

	Base Driver (s)	Refined Driver (s)	Total time (s)	Base Driver (%)	Refined Driver (%)
Invocation 1	0,023011	0,000757	0,023768	96,81	3.19
Invocation 2	0,032650	0,001142	0,033792	96,64	3,36
Invocation 3	0,000373	0,000823	0,001196	31,04	68,96

Le coût apporté par la l'adaptation de l'invocation des objets DPWSDevice en une invocation d'objets SmartDevice est de l'ordre de 3% en comparaison du temps total, ce qui

est négligeable. Il est notable que la co-localisation des deux plateformes OSGi sur la même machine augmente toutes les valeurs absolues de manière importante.

2.5 Analyse globale des résultats

Le constat le plus visible est que l'adaptation réalisée par les pilotes raffinés a un coût négligeable dans l'application du point de contrôle générique. La mutualisation de la première couche de pilotes pour différentes applications est un premier avantage important de cette architecture. Aussi, l'abstraction apportée par la deuxième couche de pilotes structure et simplifie les développements au niveau supérieur (logique métier).

Le deuxième constat est le gain important apporté par la co-localisation des serveurs et clients sur une même plateforme OSGi. Le gain correspond à l'absence de sérialisation et de dé-sérialisation dans l'interaction demandeur-fournisseur ("invocation 3"). Ce meilleur cas est un avantage important de la conception modulaire d'applications avec le partage de pilotes orientés services.

Une nuance est à apporter aux valeurs des temps d'exécution dans les pilotes raffinés. Le coût de l'adaptation dépend naturellement de la complexité des transformations visées. Ici, la transformation des objets dans un ensemble d'interfaces plus génériques implique une instanciation d'objets plus complexes que les objets réifiés initialement par la première couche de pilotes. Le coût de l'adaptation demeure négligeable avec ce niveau de complexité.

Parmi les résultats indiqués, les résultats concernant UPnP sont surprenants. Le temps de traitement à l'invocation d'équipements de l'UPnP Base Driver du projet Apache felix est plus important d'un ordre de grandeur avec les temps de traitement des pilotes orientés service développés à France Telecom. Des modifications doivent a priori pouvoir être apportées au pilote de la communauté Apache.

3 COMPOSITION DISTRIBUEE DE SERVICES

Le chapitre "UPnP Device Service" de la spécification OSGi définit l'interface d'un équipement UPnP sur la plateforme OSGi et l'usage de l'interface du registre de service pour la découverte des équipements UPnP. Cette spécification a donné lieu à l'implémentation d'UPnP Base Drivers par différents acteurs (par exemple, ISTI-CNR, Prosyst, Knopflerfish). [DFKM02] tente une définition de "Base Drivers", ce que reprend l'approche Home SOA. Cette dernière reprend alors le patron "export-bind orienté service" afin de détailler l'implémentation des concepts dans les pilotes orientés services (cf. Chapitre V.3).

R-OSGi [RAR07] est probablement le travail le plus achevé à ce jour pour la distribution de services sur des plateformes de technologie identique. SLP est utilisé par R-OSGi et l'Extended Service Binder (cf. Chapitre VI.2) pour la découverte de services distribuée sur plusieurs plateformes OSGi. Des techniques de génération de classes mandataires, le téléchargement de bundle mandataire et l'enregistrement d'objets mandataires sur la plateforme OSGi sont aussi des points communs fondamentaux. Toutefois, R-OSGi montre son avance sur l'Extended Service Binder (ESB) sur plusieurs points. Premièrement, avantage mineur, R-OSGi utilise son propre générateur à base d'ASM – asm.objectweb.org – alors que l'ESB utilise la compilation RMI statique ou la compilation dynamique effectuée automatiquement par Java SE 5. Surtout, R-OSGi analyse les dépendances de code associées à un mandataire afin de construire et de déployer les classes utiles lors du déploiement d'application. Enfin, R-OSGi projettent les erreurs dues à la répartition sur les erreurs de chargement de modules logiciels (cf. Chapitre II.3.4.4).

4 GESTION DE L'HETEROGENEITE

L'approche Home SOA apporte l'intelligence des applications visées sur une plateforme de l'environnement. Le cadriciel permet aux applications sus-jacentes de composer les équipements présents selon leurs propres protocoles. Cette adaptation logicielle des applications à différents protocoles se retrouvent dans d'autres propositions de la littérature technico-scientifique (cf. Chapitre II.4.6).

4.1 Contrôle d'équipements

REMMOC [GBS03] est un cadriciel pour la découverte des protocoles utilisés dans l'environnement et la reconfiguration d'applications à composants pour s'adapter à cette offre de protocoles. L'approche Home SOA va plus loin que la proposition REMMOC en proposant le partage d'une interface de découverte de services unique pour toutes les fonctions locales à la plateforme et disponibles dans l'environnement. Cette interface unique est offerte par le registre de services de la plateforme (cf. Chapitre V.3.2). Aussi, cette interface unique permet la déclaration de dépendances génériques et l'automatisation de la résolution de celles-ci à l'exécution. Enfin, REMMOC ne considère pas le problème d'hétérogénéité des interfaces de services des équipements, problème qui amoindrit le bénéfice de l'adaptation de REMMOC aux protocoles utilisés dans l'environnement.

Bushmitch et al. [BKBL06] ont proposé comme les travaux de cette thèse [2008-6] la représentation d'équipements SIP sur la plateforme OSGi. Toutefois, la programmation orientée service d'OSGi n'a pas été exploitée dans ce travail puisque l'API proposée pour les équipements SIP montre un annuaire propre d'objets mandataires où les détails protocolaires ne sont cachés en aucune façon. Le développeur doit encore formater les messages d'invocation sous forme de chaines de caractères.

Le cœur de la plateforme OSGi permet l'installation dynamique de modules logiciels (bundles) et le chapitre "Device Access" [OSGi05] l'utilise afin de mettre en place un mécanisme de téléchargement de pilotes à la demande. Toutefois, même si ce chapitre mentionne le nom de "Refined Driver", et même si [DFKM02] soutient cette spécification pour la composition d'équipements distincts, ils n'indiquent pas les patrons de conception qui peuvent leur être associés pour un usage de médiation technique, ce que définit le Home SOA (cf. Chapitre V).

4.2 Communication interpersonnelle

Le prototype décrit dans le Chapitre VI.5 et [2008-6] pour l'enrichissement de la communication interpersonnelle devance le travail le plus abouti dans la littérature jusqu'alors. Ce dernier est certainement celui de [CCV06] et [VCC07] qui éclate un terminal de communication interpersonnelle entre un téléphone SIP, un serveur de contenu et une télévision UPnP. Le prototype de cette thèse propose premièrement, grâce au cadriciel Home SOA, une gestion plus générale de l'hétérogénéité des équipements en permettant à d'autres protocoles qu'UPnP et SIP d'être intégrés. Deuxièmement, l'insertion d'un transcodeur dans l'architecture permet aussi la gestion de l'hétérogénéité des flux lorsque la phase de négociation de formats échoue. Troisièmement, la découverte d'équipements SIP est permise de manière analogue à la découverte d'autres équipements Plug-n-Play.

4.3 Gestion de cycle de vie logiciel

La contribution au Comité de Travail UPnP Device Management (UPnP DM) est concurrente de travaux tels ceux de l'OMA nommés SCOMO [SCOM08] et de ceux du

DMTF [DMTF06]. Elle partage aussi des points de comparaison avec le modèle Home Gateway Linux de [Roy07].

SCOMO est en effet une spécification aux objectifs similaires du service de gestion de cycle de vie logiciel proposé par cette thèse. Il est de même agnostique aux technologies de plateforme. SCOMO définit deux unités logicielles : le Delivery Package et le Deployment Component, unités similaires aux Delivery Package et Deployment Unit proposés. Le passage par un état incomplet de l'installation du DU (état Unresolved) peut être associé à l'état Inactive de SCOMO.

En revanche, SCOMO ne présente premièrement pas la généralité d'une troisième unité. L'unité d'exécution est un sujet d'administration important pour celui qui souhaite activer des services à distance et l'unité correspond effectivement à des entités existant sur les technologies étudiées. Deuxièmement, les états transitoires manquent dans le modèle SCOMO. Des notifications d'erreur interviendront lorsque que des requêtes d'administration concurrentes se succèderont sur l'équipement SCOMO alors que les verrous pris par la première requête seront visibles à l'acteur de la deuxième requête. Troisièmement, SCOMO définit les opérations comme un effet de bord d'une opération de configuration de l'arbre de données de l'équipement (cf. les nœuds "Activate", "Deactivate" et "Remove" de la Figure 13 du Chapitre II.4.5.4) alors que les actions UPnP ou les appels de procédure distante (RPC) TR-069 [TR6904] sont plus proches de la sémantique d'une opération.

Le groupe DMTF a spécifié le modèle CIM [DMTF06] qui est une vue conceptuelle de l'environnement administré. En particulier, le modèle d'administration d'application (Application Management Model) de CIM décrit les informations communément requises pour le déploiement logiciel. De manière similaire aux travaux de cette thèse, ils définissent plusieurs entités logicielles : "Software Product", "Software Feature", "Software Element" et "Application System", avec les états "deployable", "installable", "executable" et "running". Ce modèle, plus complexe, tente de faire apparaître des notions de nature intelligible à l'utilisateur comme les "Software Features" qui sont des fonctions logicielles faites de "Software Elements", les unités binaires utiles.

La complexité du modèle provient premièrement du mélange des concepts intelligibles à l'utilisateur et des concepts intelligibles à l'administrateur dans un même modèle. La dissociation entre unités de déploiement et unités d'exécution ne se retrouvent pas aisément dans les diagrammes CIM où tous les états sont portés par la seule entité "Software Element". Il apparaît deuxièmement qu'un état important manque : un état correspondant à une unité déployée mais non résolue. Cet état n'apparaît que de manière ambiguë dans les conditions qui doivent être satisfaites pour qu'un "Software Element" passe de l'état "installable" à "executable".

Les travaux de [Roy07] rejoignent l'effort de cette thèse sur la définition d’une interface de gestion du logiciel d'équipements domestiques. En effet, les concepts d’ajout, de suppression, de lancement et d'arrêt de modules logiciels se trouvent sur le prototype Home Gateway Linux (HGL). Yvan Royon a implémenté ce modèle proche du modèle OSGi sur le gestionnaire de paquetages d'une distribution Linux Slackware. La construction des paquetages est toutefois modifiée afin de définir les fonctions de lancement et d’arrêt comme sur les paquetages comme elles sont déclarées dans le manifeste d'un bundle OSGi™.

Néanmoins, l’utilisation d’un format propre à l’application HGL limite l’utilisation de la plate-forme. De plus, l'utilisation d'une distribution Slackware ne permet pas d'appréhender les possibilités de gestion automatique de dépendances, possibilités disponibles sur les autres familles de distributions Linux. Enfin, tout comme le modèle DMTF, le modèle HGL ne montre pas d'état similaire à l'état "Unresolved". Les problèmes liés à la présence de paquetages aux dépendances non résolues ne sont pas rendus visibles.

4.4 Hétérogénéité sémantique

uMiddle [NTKR06] est basé sur une approche nouvelle de composition à granularité fine. Le grain de composition n'est plus le service mais les opérations ou les données proposées le service. Ce grain fin augmente les possibilités de composition. Toutefois, lier les entités pervasives sur des ressemblances de types de données fournies et requises peut mener à des compositions indésirables si ces types ne sont pas spécifiques (de nombreux services fournissent et requièrent des chaines de caractères par exemple). Les liaisons sur des critères de ressemblance partielle de services nécessitent une plus grande richesse descriptive. Sans montrer une application aussi importante que [NTKR06], le système de génération d'adapteurs à la volée du Chapitre V.4.4, couplé à des répertoires d'ontologies [WSMO06], offre des capacités de recombinaison qui ne dépendent que de la richesse des informations attachées et de la qualité d'analyse des raisonneurs sémantiques. Ce système n'a toutefois pas été réalisé complètement et est en marge de l'approche syntaxique du Home SOA.

5 COMPOSITION CONTEXTUELLE

5.1 Contextualisation par un système de gestion de contexte

Le cadriciel Home SOA utilise le registre de service local pour la contextualisation des services fournis et des requêtes de services. Il allie ce mécanisme de contextualisation à un mode de requête riche de services par utilisation des outils de sélection de services suivants : filtre obligatoire, filtres optionnels, fonction d'utilité. Cette alliance permet le classement proactif des services qui initie les réactions de recomposition adéquate des demandeurs de services [2007-3].

Le cadriciel Home SOA partage l'originalité des architectures telles celles de [PaT06] et [ABR05], qui promeuvent la séparation des sources de contexte et des acteurs à services (demandeurs et fournisseurs). Ce modèle simplifie le développement en décomposant celui-ci en domaines orthogonaux. Le Home SOA reprend ce concept pour l'appliquer dans le langage programmatique de la plateforme de services locale. L'idée de la contextualisation du registre de service local à la plateforme est originale.

David et al. [DaL06] décrivent l'implémentation de mécanismes de reconfiguration automatique au-dessus du modèle Fractal [BCLQ06]. Les auteurs définissent des politiques de réactions selon des règles Evénement-Condition-Réaction. Le cadriciel proposé, nommé Safran, possède un avantage que Home SOA ne présente pas : les règles de réaction sont configurées de manière externe aux composants clients. Par utilisation de technique d'ingénierie de code binaire et de programmation orientée aspects, Safran tisse le code de réaction avec celui du composant visé à l'exécution. Cet avantage pourrait être une piste d'amélioration du Home SOA dans le futur. Il semble être complémentaire de la composition paire-à-paire effectuée par les composants du Home SOA. En effet, L'auto-description des composants et les mécanismes génériques de courtage lâche de service sur une plateforme locale sont plus à même de représenter les éléments pervasifs des environnements visés que la description à un haut niveau de la composition de Safran. Les mécanismes du Home SOA permettent l'émergence spontanée de la composition logicielle à l'exécution.

Le cadriciel Home SOA donne les outils au développeur pour ordonnancer la reliaison des services pour transférer l'état du service selon les requis d'une application. Il laisse toutefois le soin au développeur de transférer cet état, ce transfert d'état semblant propre à chaque application. Certains travaux tels [FEL07] montrent la faisabilité de l'automatisation du transfert avec l'hypothèse suivante : chaque composant voit son état décrit par la séquence des opérations qu'il a reçue. Avec cette hypothèse forte, le système proposé attache des mandataires à chaque service utile afin d'enregistrer cette séquence et de la

rejouer sur le service remplaçant si un service vient à être délié. L'hypothèse est forte puisqu'elle ne prend pas en compte l'évolution possible du service avec le temps passé entre chaque opération (par exemple, le temps déroulé après l'appel de la méthode Play() indique la durée du film à partir de laquelle recommencer à jouer sur un lecteur de films. Cet exemple montre que le problème de l'automatisation générique ne peut être résolu sans que les services fournissent une interface dédiée à leur transfert d'état. L'exemple confirme que la politique de transfert dépend de l'application visée.

Le cadriciel Home SOA suit partiellement des concepts de l'Informatique Autonomique. Tout d'abord, il se base essentiellement sur l'utilisation d'un registre de services tels que celui prôné par Kephart et al. dans [WHWC04]. Surtout, tous les composants de l'application et toutes les entités pervasives réifiées par le cadriciel Home SOA montrent des services auto-décrits, ce qui permet aux composants demandeurs de services de classer les services fournis selon des politiques internes de haut niveau. Les techniques du Home SOA laissent émerger l'intelligence des applications par composition paire-à-paire spontanée des composants présents sur la plateforme de services. L'auto-assemblage sans requérir de planification centralisée est une caractéristique saillante des systèmes autonomiques [Kep05].

6 PERTINENCE DU HOME SOA POUR L'OPERATEUR DE TELECOMMUNICATION

Plusieurs projets de la division Recherche et Développement de France Telecom reposent sur une approche logicielle modulaire afin d'apporter la flexibilité à l'écosystème de la LiveBox dans les années à venir. Cette flexibilité apportée au niveau logiciel s'oppose à la faible flexibilité rendue aujourd'hui par l'ajout et le renouvellement d'équipements ("box") proposés aux clients au gré de l'évolution de la demande et de l'offre d'applications.

Les bénéfices de l'approche Home SOA sont, pour le client :

- **Adaptation des services** : les services peuvent être particularisés selon les préférences du client grâce à la modularité des applications. Seules les applications en accord avec ses équipements lui sont proposées.
- **Evolutivité** : selon ses préférences, le client peut recomposer son ensemble de services sans devoir changer l'équipement composeur d'application (Passerelle domestique, Set-Top-Box, etc.).
- **Simplicité d'installation/configuration** : l'utilisation de protocoles domestiques plug-n-play existants garantit une installation automatique. L'installation de nouveaux équipements au cours du temps est prise en compte dans une configuration dynamique.
- **Confort** : les mises à jour deviennent plus rapides et invisibles dans de nombreux cas pour le client. La mise à jour de modules est plus rapide que celle du micrologiciel complet. La mise à jour sont effectuées à chaud, c'est-à-dire sans que les applications indépendantes du module mis à jour soient arrêtées.

Les bénéfices pour l'opérateur sont :

- **Simplicité de supervision** : la représentation uniforme des équipements permet des outils de supervision simples et s'adaptant à la venue de nouveaux protocoles.
- **Baisse des coûts hardware à moyen terme** - en raison de l'évolutivité de la plateforme : le coût plus grand d'une plateforme évolutive est amorti par sa plus

grande longévité puisqu'elle est disposée à héberger les applications renouvelées au fil du temps et à s'adapter à l'évolution des méthodes d'administration.

- **Baisse des coûts de mise à jour** : les mises à jour modulaires sont rapides. La rapidité entraîne des gains de temps d'utilisation des serveurs de mise à jour et des temps de hotline (dans le cas où les mises à jour sont effectuées par la hotline). De plus, la modularité entraîne un gain de bande passante.
- **Optimisation des capacités** : seuls les modules applicatifs pertinents sont installés. Cela minimise l'empreinte hardware nécessaire à l'application.

Le choix de la technologie OSGi apporte aussi des avantages industriels de rapidité de développement et de partage d'une norme publique (cf. section Chapitre VI.1.1).

7 SYNTHESE

Le test des briques Home SOA développées montrent les résultats suivants :

- La modularité de la conception proposée pour les pilotes orientés service a un coût minime en comparaison des gains de simplicité de développement et d'évolutivité de l'architecture.
- La réification d'objets dans des interfaces adaptées à la demande des clients est le rôle des pilotes raffinés. Ces transformations ont un coût négligeable et permettent la mutualisation du premier niveau de pilotes entre les applications.

Le cadriciel proposé montre des avantages architecturaux en comparaison des approches de l'état de l'art. Premièrement, le courtage de services en tant qu'ensemble de méthodes est plus cohérent que l'approche ambitieuse de courtage de simples opérations sur de complexes inférences sémantiques. Les approches sémantiques pourraient toutefois montrer leurs atouts avec la maturation des travaux dans ce domaine.

Concernant la gestion de la distribution et de l'hétérogénéité, la découverte d'équipement et d'objets adaptés au travers de l'interface unique offerte par le registre de services est un atout important du mode proposé de conception des pilotes. Cette interface unique rend indépendants le développement de pilotes et le développement des applications découvrant les équipements. De plus, objets locaux et objets distants se découvrent de la même façon, ce qui accroit la transparence des objets mandataires.

Concernant la gestion de la dynamique de l'environnement, le mode de contextualisation des services sépare la gestion du contexte de la fourniture de services. L'automatisation des dépendances contextualisées de services semble de plus être un atout rare dans les applications décrites dans la littérature.

Aussi, cette thèse propose des architectures à la généralité inédite : la proposition technique au Forum UPnP saisit la modularité et la dynamique des applications sur les plateformes d'exécution modulaires d'aujourd'hui tout en appliquant les concepts sur un protocole orienté services. La généralité du prototype de communication interpersonnelle enrichie est par ailleurs en avance sur les travaux concurrents proposés.

Enfin, l'approche Home SOA demeure une piste d'anticipation pertinente pour l'évolutivité des offres l'opérateur France Telecom sur le réseau domestique.

Chapitre VIII. CONCLUSION

Face aux défis de l'Informatique Pervasive dans les réseaux locaux, cette thèse répond par une ligne de conduite et l'approche logicielle Home SOA. Cette ligne de conduite distingue les situations où les solutions protocolaires sont pertinentes et ramène les autres situations à des problèmes de génie logiciel. Parmi les solutions protocolaires, la proposition d'une interface uniforme de gestion de cycle de vie logiciel dans le Comité de Travail UPnP Device Management est une des contributions importantes de cette thèse.

Le Home SOA est l'association de technologies de développement modulaire et d'un ensemble de patrons de conception orientés objets. Au-delà de l'orientation objet, le Home SOA exploite les modèles récents de composants à services. L'implémentation du cadriciel est illustrée par la description d'applications innovantes dans le réseau domestique. Cette section résume la problématique, expose les réponses apportées avec leurs limites et annonce les perspectives de ce travail.

1 LES DEFIS DE L'INFORMATIQUE PERVASIVE DES RESEAUX LOCAUX

Cette thèse propose un ensemble de technique et un cadriciel nommé Home SOA face à trois défis importants de l'Informatique Pervasive. Les solutions tentent de simplifier la gestion de la distribution, la gestion de l'hétérogénéité et la gestion de la dynamique des équipements dans les applications les composant dans les réseaux locaux. Les réseaux locaux comme le réseau domestique, le réseau véhiculaire, le réseau mobile, le réseau d'entreprise sont les cibles pertinentes pour les techniques présentées dans ce mémoire. Les expérimentations de cette thèse ont en particulier été menées dans le contexte de projets sur le réseau domestique.

D'abord, l'ouverture des environnements pervasifs et la modularité des plateformes d'exécution demandent aux applications de se lier dynamiquement à des entités logicielles non précisément connues à l'avance. Bien que des techniques de l'Informatique Distribuée permettent depuis quelques années la transparence des appels de procédures et d'objets distants, la transparence de la liaison dynamique d'objets inconnus à l'avance requiert encore des raffinements des langages de programmation.

Aussi, les équipements électroniques envahissent aujourd'hui les réseaux locaux tels le réseau domestique en raison de la baisse des coûts de l'électronique et la venue d'usages permis par les progrès technologiques. Afin de gérer l'hétérogénéité de ces équipements, une grande flexibilité est demandée aux architectures logicielles embarquées. En effet, celles-ci doivent évoluer en s'adaptant à de nouveaux protocoles et interfaces d'équipements.

Enfin, la dynamique de l'activité de l'utilisateur, le nombre de ceux-ci et la dynamique propre des équipements et de l'environnement exigent un niveau de réactivité élevé des applications visées. L'automatisation des réactions applicatives à un ensemble de contraintes évoluant dans le temps demeure un domaine de recherche important. La conscience du contexte est une intelligence que les chercheurs de l'Informatique Pervasive tentent d'inculquer à leurs applications.

2 LES REPONSES DE CETTE THESE

En réponse à ces défis, cette thèse présente tout d'abord une ligne de conduite face aux problèmes d'hétérogénéité et un ensemble cohérent de techniques logicielles à appliquer selon les problèmes donnés. Les contributions principales consistent premièrement en la présentation des techniques du Home SOA et de leur application dans des scénarios nouveaux du réseau domestique et deuxièmement dans la réponse protocolaire au manque de protocoles existant dans la gestion du cycle de vie logiciel.

2.1 Une ligne de conduite

Tous les protocoles qui coexistent sur un même réseau peuvent inter-opérer grâce à des techniques de médiation. Les techniques appliquées dans cette thèse sont opportunistes :

- Séparer les ressources empruntées par les différents protocoles s'ils ne peuvent coexister sur ces mêmes ressources (cf. analyse d'IGRS Chapitre VI.6 [2006-7]),
- Appliquer ensuite des techniques de passerelles si les protocoles sont utilisés sur des réseaux aux topologies différentes (terrain, LAN, WAN) et appliquer des techniques d'adaptation logicielles pour lier des protocoles distincts sur ces passerelles (cf. Chapitre V.4.2.2 [2008-5]),
- Appliquer une approche protocolaire pour lier des plateformes hétérogènes sans protocole commun (cf. création du Comité UPnP DM Chapitre V.6 [2007-9]),
- Appliquer enfin des techniques d'adaptation logicielle pour simplifier la gestion de la distribution, de la dynamique et de l'hétérogénéité sur une plateforme au cœur de réseaux pervasifs à la topologie homogène. Cette plateforme intelligente héberge les applications innovantes du réseau visé et porte l'attention principale des techniques du Home SOA (cf. Chapitre V et articles [2008-5][2007-4][2007-2] associés aux travaux des projets ANSO et d'Amigo).

2.2 Les techniques du Home SOA

Le Home SOA modélise les entités pervasives du réseau domestique sous la forme de services [2008-5][2008-4]. Le paradigme de l'orientation service convient à la composition de ces entités distribuées dans l'environnement, qui se connectent et se déconnectent dynamiquement du réseau et qui présentent des interfaces aux protocoles hétérogènes et aux sémantiques diverses. Cette vision est accompagnée d'un ensemble cohérent de patrons de conception dans le paradigme orienté services. Cet ensemble ramène le problème de composition à la programmation d'applications locales à une plateforme tout en simplifiant la gestion de la dynamique et de l'hétérogénéité des équipements.

Raffinant le patron export-bind et l'appliquant au-dessus du patron de plateforme de service, les pilotes orientés services réifient les entités pervasives de l'environnement sur la plateforme (cf. Chapitre V.3 [2008-2][2007-6][2006-4], projet européen IST Amigo [2006-5], projet national Pise [2006-6]). Le registre local de services est le réceptacle des objets de services locaux et distribués. L'interface unique offerte par le registre pour la découverte de

services affranchit les applications de tout couplage avec les protocoles de découverte existants. Chaque pilote orienté service est responsable de la découverte et de la communication synchrone et par événement selon un ensemble protocolaire donné. La pluralité des ensembles protocolaires présents dans le réseau domestique est alors couverte par la pluralité des pilotes orientés services.

La gestion de l'hétérogénéité est simplifiée par une couche de composants adapteurs transformant dynamiquement des objets de service à la sémantique proche de définitions protocolaires en des objets de services à la sémantique épurée du domaine d'application visé (cf. Chapitre V.4 [2008-5]). Ces composants réactifs raffinent le travail des pilotes orientés services. De tels pilotes raffinés sont développés pour le contrôle d'équipements multimédia et d'équipements domotiques dans une application de communication interpersonnelle du projet national Systerminal (cf. Chapitre VI.5 [2008-6]) et dans un outil de supervision et de contrôle générique de tous les équipements de la maison [2008-4].

La dynamique de l'environnement fait l'objet de techniques de contextualisation des services fournis et des requêtes de service. Cette contextualisation est alors utilisée par des algorithmes de sélection de services déclarés a priori et évoluant avec le contexte des requêtes. Ces techniques sont employées dans le projet ANSO (Chapitre V.5 [2007-7][2007-3]) dans une application de messagerie ambiante mono-utilisateur.

2.3 UPnP DM : administration de plateformes hétérogènes

L'auto-organisation des équipements du réseau domestiques peut être enrichie par les capacités de déploiement logiciel à la demande sur les plateformes modulaires de l'environnement. La création du Comité UPnP Device Management (ex-Execution Platform) (DM) [2007-9] est partie de cette idée et du constat du manque d'un protocole commun pour administrer les plateformes d'exécution existantes. Quelques modèles de plateformes de déploiement modulaire existent dans la normalisation des protocoles d'administration mais sont incomplets et n'adressent pas la dynamique et l'hétérogénéité du réseau local. Les compétences acquises dans l'implémentation d'applications modulaires sur les plateformes innovantes d'aujourd'hui – notamment les plateformes OSGi, .NET et MIDP – a permis la proposition d'une analyse et d'un modèle général de plateforme d'exécution (cf. Chapitre V.6). La proposition applique ce modèle dans la proposition concrète d'une interface de service UPnP (cf. Chapitre VI.7). La proposition de ce modèle est validée à France Telecom par son implémentation sur trois technologies différentes : Linux Ubuntu, .NET et OSGi (cf. Chapitre VI.7). Ces preuves de concept seront suivies par la fourniture d'une implémentation de référence au Forum UPnP en 2009.

L'impact de cette proposition technique est non seulement direct sur la spécification en cours du Comité de Travail UPnP Device Management mais aussi sur d'autres organisations de normalisation. Un travail parallèle a été initié dans l'OSGi Residential Expert Group par un RFP (Request For Proposal) soumis à l'Alliance OSGi [2007-8] afin de proposer une implémentation OSGi de la spécification UPnP. Une proposition d'exigences techniques concernant la gestion de plateformes d'exécution a été présentée au Broadband Forum par un partenaire du Comité UPnP. Le montage du Comité CSPR (Content and Service Providers Requirements) de l'Alliance DLNA et certains documents de la Home Gateway initiative mentionnent aussi des liens à ce travail.

3 LES ENSEIGNEMENTS ET LES LIMITES DES PROPOSITIONS

Les travaux de ces trois années de thèse sont riches d'enseignements concernant la conduite de projet et l'évaluation des limites de propositions techniques. Cette section présente une leçon tirée dans la recherche de la transparence dans chacun des thèmes principaux : gestion de la distribution, gestion de l'hétérogénéité, gestion de la dynamique.

3.1 Plateforme de services répartie et contrôle d'équipements

Les travaux qui ont débuté cette thèse ont porté sur la programmation d'applications distribuées de façon transparente. L'objectif était très technique et ne répondait pas a posteriori à des cas réel d'application dans les réseaux locaux tels que le réseau domestique. La prise de recul sur les applications produites la première année en comparaison des attentes des projets a permis un virage dans les orientations des travaux de cette thèse. L'aversion de Richard S. Hall, présent dans les réflexions techniques de première année, a aussi influencé ce virage au tournant de l'année 2006 [2007-3].

L'Extended Service Binder (cf. Chapitre VI.2) est un cadriciel qui permet la répartition d'une application sur plusieurs plateformes OSGi. Il est notable que plusieurs autres projets étaient concurrents à ce travail. Le travail de référence dans la communauté OSGi est celui de Jan S. Rellermeyer [RAR07] qui offre une solution complète, détaillée et validée pour la répartition d'applications post-développement en regard des possibilités procurés par les nouvelles technologies de déploiement modulaires telles qu'OSGi [OSGi05]. Newton et D-OSGi [MSSG08] sont d'autres solutions de l'état de l'art. L'Alliance OSGi publie aujourd'hui une première spécification dans ce domaine, nommée "Distributed OSGi".

Tous ces travaux concurrents, exceptés la spécification "Distributed OSGi", se sont aussi trompé de cible puisque les scénarios proposés sont aussi des scénarios du réseau domestique. En effet, le réseau domestique ne présente pas de besoins pour la mobilité et le déploiement de code sur des plateformes homogènes. Il montre plutôt des plateformes et des équipements hétérogènes, ce qui met en avant le besoin d'un cadre de développement d'applications de contrôle des équipements existants. La mobilité de code et le déploiement de parties d'applications sur des plateformes homogènes sont associés principalement à des besoins d'équilibre de charge. Ces besoins ne sont pas pertinents dans le réseau domestique. Ils le sont certainement dans le domaine des serveurs d'application, ce qui explique pourquoi la spécification "Distributed OSGi" est en cours dans l'Enterprise Expert Group.

3.2 Le coût de la flexibilité

Le Chapitre VII montre des résultats de tests positifs validant l'apport des concepts de pilote orienté service et de pilote raffiné. En effet, les avantages de la modularité de l'approche Home SOA pour le contrôle uniforme d'équipements dans un domaine d'application donné ne semblent pas avoir de défaut majeur.

Pourtant, la flexibilité du cadriciel Home SOA et de la plateforme OSGi a un coût. Leurs avantages sont bâtis sur les atouts de la machine virtuelle Java : la programmation objet, le typage fort, la sécurité, le chargement de classes dynamique, le ramasse-miettes. Aujourd'hui, la machine virtuelle Java nécessaire requiert quelques méga-octets et duplique certaines fonctions que possèdent déjà les serveurs d'exploitation. Cela explique pourquoi la machine virtuelle Java standard (la machine virtuelle des applications mobiles exceptée) et OSGi sont demeurés des succès sur les serveurs d'exploitation et n'ont pas réussi à se répandre largement sur le marché de l'embarqué.

La deuxième leçon de cette thèse est que la simplicité de développement n'est pas corrélée à l'efficacité des programmes. Home SOA, comme tous les ensembles de patrons de conception, privilégie l'évolutivité des applications devant leur efficacité.

3.3 Les treize écueils de l'Informatique Pervasive

Aux limites de la transparence de la répartition montrées par Peter Deutsch, s'ajoutent cinq autres écueils de la recherche de transparence de la dynamique pour le développeur. Cette section édifie donc la liste des treize écueils de l'Informatique Pervasive

[2007-2] comprenant les huit écueils de l'Informatique Distribuée énoncés par Peter Deutsch et les cinq faux 'a priori' suivants :

- Les services sont disponibles immédiatement (Services are available at once) : en raison de l'utilisation des équipements, de mesures de sauvegarde d'énergie, de la mobilité des équipements, les applications pervasives se doivent d'être programmées avec l'hypothèse que les services recherchés seront absent la plupart du temps. Les applications se doivent donc d'être réactives aux événements d'apparition de services.
- Un service découvert est disponible sur une longue période (A discovered service is available in the long run) : pour les mêmes raisons citées ci-dessus et pour des raisons de non-fiabilité de certains réseaux (wifi par exemple) et de certains équipements, les applications se doivent de prévenir les départs de service à tout moment, même en cours d'utilisation d'un équipement associé. En cas de départ d'un service, des mécanismes de reliaison doivent être mis en place. Le Chapitre VI.1.2 propose des mécanismes de reconfiguration de services. Toutefois, rétablir le fonctionnement d'une application au moment où un service part alors qu'il est utilisé est s'avère souvent difficile.
- Un service n'a pas d'état (A service has got no state) : les services fournis par les équipements embarqués d'un réseau local possèdent généralement un état. La machine à laver à un état et ne peut pas être lancée par un utilisateur différent au moment où elle fonctionne pour un utilisateur.
- Toutes les méthodes de l'interface de service sont implémentées par un service (All service interface methods are implemented). Un service enregistré avec une certaine interface peut implémenter une, plusieurs ou toutes les méthodes de cette interface. UPnP normalise d'ailleurs des types d'équipements comportant des services obligatoires et des services optionnels, et, parmi les actions définies dans les services, seuls quelques unes d'entre elles sont définies "obligatoires".
- Les services utilisés sont connus durant la phase de développement (Used services are known at development time). Cette assertion est fausse dans les réseaux pervasifs et c'est pour cela que le développement dans ces environnements fait appel au paradigme d'orientation service. Seules les interfaces protocolaires et programmatiques sont connues à l'avance. Les détails d'implémentation demeurent généralement spécifiques.

Tous ces écueils rendent le développement difficile même en présence d'outils et de cadriciels annonçant masquer la dynamique des environnements. Les situations dues à la dynamique doivent être anticipées durant la phase de conception afin de penser à toutes les réactions à mettre en place au développement.

4 LES PERSPECTIVES DE CES TRAVAUX

Les travaux de cette thèse ont apportés des réponses dans un domaine précis et été validée au-dessus d'un type particulier de technologies. L'élargissement à d'autres domaines et la transposition des concepts sur des technologies aux avantages de coût plus évidents offre des perspectives de recherche.

4.1 Gestion d'autres aspects dans le Home SOA : sécurité, QoS.

Distribution, hétérogénéité et dynamique sont les aspects techniques adressés par cette thèse. Cependant, ces aspects ne constituent pas les seuls défis de l'Informatique Pervasive. L'ouverture des réseaux à des entités inconnues a priori et l'ouverture des

plateformes à des composants développés par des parties tierces posent des problèmes de sécurité. La multiplicité des utilisateurs et de leurs activités simultanées posent aussi le défi de la juste gestion des ressources pour assurer la QoS (qualité de service).

Face aux défis posés par la sécurité d'environnements ouverts et dynamiques, certains apportent des solutions de reconfiguration dynamique du système de sécurité corrélé au degré d'ouverture et de niveau de confiance des entités présentes [LPR07]. Les pilotes orientés services du Home SOA pourraient être l'objet de l'implémentation de mécanismes d'authentification des équipements et participer ensuite à la reconfiguration dynamique du niveau d'authentification requis grâce aux mécanismes proposés. D'autres apportent des solutions techniques accélérant l'évaluation des contrôles d'accès à l'exécution d'un programme composés de composants interdépendants [PaF08]. La particularisation du contrôle d'accès pour les équipements du réseau domestique pourrait être l'objet d'une extension de l'intergiciel Home SOA. La réification de l'authentification et du contrôle d'accès protocolaires aux équipements dans l'authentification et le contrôle d'accès programmatiques simplifierait la programmation d'applications sécurisées.

Aussi, cette thèse n'a pas approfondi la gestion dynamique de la QoS. Pourtant, assurer la qualité de service dans les nombreuses applications du réseau domestique demeure une fonction demandée à l'opérateur. L'ajout de l'affectation et de l'évaluation de propriétés de QoS aux mécanismes de sélection automatique du Home SOA est une piste pour la complétion du cadriciel face aux exigences des nouvelles applications des réseaux locaux, en particulier multimédia [AlL08].

4.2 L'application du Home SOA dans un environnement contraint

Pour répondre aux besoins de l'Informatique Pervasive dans les réseaux locaux, l'approche les techniques du Home SOA exploitent la modularité de plateformes telles qu'OSGi et .NET. La plateforme OSGi et son système de partage et d'isolation de code à grain fin est particulièrement adaptée à l'implémentation des patrons de conception orientés service, c'est-à-dire les réponses à la demande de couplage lâche des applications pervasives.

Malheureusement, ces plateformes s'avèrent couteuses pour le marché domestique aujourd'hui. La machine virtuelle Java nécessaire au fonctionnement de la plateforme OSGi nécessite par exemple des capacités de mémoires (volatiles et non-volatiles) de l'ordre du Mo, capacités qui s'ajoutent à celle du serveur d'exploitation sous-jacent.

Une suite possible de cette thèse est la tentative de porter les techniques de Home SOA sur une plateforme moins couteuse. L'analyse des besoins et du coût des techniques du Home SOA peut amener à des choix sur des critères de performances. Le partage et l'isolation de code d'une part et les capacités de liens dynamiques entre parties de code d'autre part sont des besoins difficiles à satisfaire sur des équipements aux ressources contraintes.

Certains langages associés à des techniques particulières de compilation permettent le développement d'applications modulaires dont les liens sont dynamiques à l'exécution. La plateforme THINK (think.objectweb.org, [FSLM02]) est un exemple extrême de plateforme dont l'empreinte est de l'ordre de 10Ko, THINK étant son propre serveur d'exploitation.

Sans penser à des plateformes aux contraintes si extrêmes, il est pensable que la demande montante de flexibilité des applications et l'évolution des plateformes orientées composants à service vers l'embarqué puissent un jour se rencontrer sur un terrain d'entente. Le Home SOA sera alors peut-être devenu une réalité de marché!

Chapitre IX. BIBLIOGRAPHIE

1 PUBLICATIONS

[2008-6] Mourad Alia, André Bottaro, Fatoumata Camara, Briac Hardouin, "On the Design of a SIP-based Binding Middleware for Next Generation Home Network Services", 10th International Symposium on Distributed Objects, Middleware, and Applications (DOA'08), Monterrey, Mexico, November 2008

[2008-5] André Bottaro, Anne Gérodolle, "Home SOA - Facing Protocol Heterogeneity in pervasive Applications", 5th IEEE International Conference on Pervasive Services (ICPS 2008), Sorrento, Italy, July 2008

[2008-4] André Bottaro, Anne Gérodolle, Levent Gurgen, "Home SOA - Controlling Home Pervasive Devices", 5th IEEE International Conference on Pervasive Services - Demonstration proposals (ICPS 2008), Sorrento, Italy, July 2008

[2008-3] Levent Gurgen, Claudia Roncancio, Cyril Labbe, André Bottaro, Vincent Olive, "SStreaMWare: a service oriented middleware for heterogeneous sensor data management", 5th IEEE International Conference on Pervasive Services (ICPS 2008), Sorrento, Italy, July 2008

[2008-2] André Bottaro, Eric Simon, Stéphane Seyvoz, Anne Gérodolle, "Dynamic Web Services on a Home Service Platform", 22nd IEEE International Conference on Advanced Information Networking and Applications (AINA-08), Ginowan, Okinawa, Japan, March 2008

[2008-1] Stéphanie Chollet, Philippe Lalanda, André Bottaro, "Transparently adding security properties to service orchestration", 3rd IEEE International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE-08), Ginowan, Okinawa, Japan, March 2008

[2007-9] André Bottaro, Jooyeol Lee, Kai Hackbarth, Didier Donsez for the UPnP Forum, "Execution Platform Working Committee Charter", October 2007

[2007-8] André Bottaro, Peter Kriens, Jae Shin Lee, Kiran Bharadwaj Vedula, Hyun-Gyoo Yook for the OSGi Alliance, "RFP 101 The OSGi service platform as a UPnP device", October 2007

[2007-7] André Bottaro, Johann Bourcier, Clément Escoffier, Philippe Lalanda, "Autonomic Context-Aware Composition", 4th IEEE International Conference on Pervasive Services (ICPS'07), Istanbul, Turkey, July 2007

[2007-6] André Bottaro, Anne Gérodolle, Sylvain Marié, "Combining OSGi technology and Web Services to realize the plug-n-play dream in the home network", 2nd OSGi Community Event, Munich, Germany, June 2007

[2007-5] André Bottaro, Anne Gérodolle, Sylvain Marié, Stéphane Seyvoz, Eric Simon for the OSGi Alliance, "RFP 86 DPWS Discovery Base Driver", May 2007

[2007-4] André Bottaro, Anne Gérodolle, Philippe Lalanda, "Pervasive Service Composition in the Home Network", 21st IEEE International Conference on Advanced Information Networking and Applications (AINA-07), Niagara Falls, Canada, May 2007

[2007-3] André Bottaro, Richard S. Hall, "Dynamic Contextual Service Ranking", 6th International Symposium on Software Composition (SC'07), Braga, Portugal, March 2007

[2007-2] André Bottaro, "Realizing the Plug-n-Play Dream on the Home Network", Short Talk, 1st OSGi Developer Conference (EclipseCon'07), Santa Clara, CA, USA, March 2007

[2007-1] André Bottaro, Johann Bourcier, Clément Escoffier, Didier Donsez, Philippe Lalanda, "A Multi-Protocol Service-Oriented Platform for Home Control Applications", 4th IEEE International Consumer Communications and Networking Conference - Demonstration proposals (CCNC'07), Las Vegas, NV, January 2007

[2006-7] André Bottaro, Anne Gérodolle on behalf of France for the ISO/IEC, "Comments on ISO/IEC WD 14543-5-1: Intelligent Grouping and Resource Sharing (IGRS) for HES Class 2 & Class 3 – Part 5-1: Core Protocol", September 2006

[2006-6] Françoise Baude, André Bottaro, Jean-Michel Brun, Antonin Chazalet, Arnaud Constancin, Didier Donsez; Levent Gurgen, Philippe Lalanda, Virginie Legrand, Vincent Lestideau, Sylvain Marié, Christina Marin, Alain Moreau, Vincent Olive, "Extension de passerelles OSGi pour la grande échelle : Modèles et outils", Atelier OSGi des 3èmes Journées Francophones Ubiquité et Mobilité (UbiMob), Paris, September 2006

[2006-5] Anne Gérodolle, André Bottaro, "OSGi et le projet IST Amigo", Atelier OSGi des 3èmes Journées Francophones Ubiquité et Mobilité (UbiMob), Paris, September 2006

[2006-4] André Bottaro, Anne Gérodolle, Philippe Lalanda, "Pervasive Spontaneous Composition", 1st International Workshop on Service Integration in Pervasive Environments (SIPE'06), Lyon, France, June 2006.

[2006-3] André Bottaro for the OSGi Alliance, "RFP 72 Extended Mapping for UPnP Discovery Transparency", April 2006

[2006-2] André Bottaro, Anne Gérodolle, "Extended Service Binder: Dynamic Service Availability Management in Ambient Intelligence", 1st Workshop on Future Research Challenges for Software and Services, Vienna, Austria, March 2006.

[2006-1] Fano Ramparany, Jérôme Euzenat, Tom Broens, Jérôme Pierson, André Bottaro, Remco Poortinga, "Context Management and Semantic Modeling for Ambient Intelligence", 1st Workshop on Future Research Challenges for Software and Services, Vienna, Austria, March 2006.

[2005-1] André Bottaro, Anne Gérodolle, Vincent Olive, "Procédé de gestion automatique des associations entre composants demandeurs de services et composants fournisseurs de services dans un environnement distribué", Demande de dépot de brevet INPI n° 05 53085 du 11 novembre 2005 effectivement déposée en Europe et à l'International en janvier 2008 sous le numéro WO/2007/042713.

2 REFERENCES

[ABR05] Alessandra Agostini, Claudio Bettini, Daniele Riboni, "Loosely Coupling Ontological Reasoning with an Efficient Middleware for Context-awareness", 2nd International Conference on Mobile and Ubiquitous Systems (MobiQuitous'05), San Diego, CA, USA, July 2005

[ABR06] Alessandra Agostini, Claudio Bettini, Daniele Riboni, "Experience report: ontological reasoning for context-aware Internet services", 4th IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Pisa, Italy, March 2006

[ACDK96] Cristiana Amza, Alan L. Cox, Sandhya Dwarkadas, Pete Keleher, Honghui Lu, Ramakrishnan Rajamony, Weimin Yu, Willy Zwaenepoel, "TreadMarks: shared memory computing on networks of workstations", Computer Journal, Volume 29, Issue 2, pp. 18-28, February 1996

[Aie06] Marco Aiello, "The Role of Web Services at Home", 2nd Advanced International Conference on Telecommunications (AICT/ICIW'06), Guadeloupe, French Caribbean, February 2006

[AlL08] Mourad Alia, Marc Lacoste, "A QoS and Security Adaptation Model for Autonomic Pervasive Systems", 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'08), Turku, Finland, August 2008

[BCLQ06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, Jean-Bernard Stefani, "The Fractal Component Model and Its Support in Java", Software Practice and Experience, Special Issue on Experiences with Auto-adaptive and Reconfigurable Systems. 36(11-12), 2006

[BKBL06] Alan Brown, Mario Kolberg, Dennis Bushmitch, Gennady Lomako, Matthew Ma, "A SIP-based OSGi Device Communication Service for Mobile Personal Area Networks", 3rd IEEE International Consumer Communications and Networking Conference (CCNC'06). Las Vegas, NV, USA, January 2006

[BLBK04] Dennis Bushmitch, Wanrong Lin, Andrzej Bieszczad, Alan Kaplan, Vasilis Papageorgiou, Algirdas Pakstas, "A sip-based device communication service for osgi framework", 1st IEEE International Consumer Communications and Networking Conference (CCNC'04), Las Vegas, NV, USA, January 2004

[BLC02] Eric Bruneton, Romain Lenglet, Thierry Coupaye, "ASM: A Code Manipulation Tool to Implement Adaptable Systems", Technical report, France Telecom R&D, November 2002

[Bern96] Bernstein, Philip A., "Middleware: A Model for Distributed Services", Communications of the ACM 39, 2, pp. 86-97, February 1996

[BFTZ05] Frank Bormann, Stephan Flake, Jurgen Tacken, Carsten Zoth, "Towards Context-Aware Service Discovery: A Case Study for a new Advise of Charge Service", 14th Mobile & Wireless Communications Summit, Services, Security & Context Management Workshop, Dresden, Germany, June 2005

[BiN84] Andrew D. Birrell, Bruce Jay Nelson, "Implementing remote procedure calls", ACM Transactions on Computer Systems, volume 2, Number 1, pp. 39-59, February 1984

[BrI05] Yérom-David Bromberg, Valérie Issarny, "Indiss: Interoperable Discovery System for Networked Services", 6th International Middleware Conference (Middleware'05), Grenoble, France, December 2005

[BSD03] Boualem Benatallah, Quang Sheng, Marion Dumas, "The Self-Serv Environments for Web Services Composition", Internet Computing, IEEE Volume 7, Issue 1, pp. 40–48, January-February 2003

[CCCS07] Philippe Collet, Thierry Coupaye, Hervé Chang, Lionel Seinturier, Guillaume Dufrêne, "Components and Services: A Marriage of Reason", Technical Report I3S/RR-2007-17-FR, May 2007

[CCV06] Gabriella Convertino, Fabrizio Crudo, Antonio Vilei, "A new UPnP architecture for distributed video voice over IP", 5th International Conference on Mobile and Ubiquitous Multimedia (MUM'06), Stanford, CA, USA, December 2006

[CeH03] Humberto Cervantes, Richard S. Hall, "Automating Service Dependency Management in a Service-Oriented Component Model", 6th Workshop of Component Based Software Engineering, Portland, OR, USA, May 2003

[CeH04] Humberto Cervantes, Richard S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model", 26th International Conference on Software Engineering, Edinburgh, Scotland, UK, May 2004

[ChK03] Guanling Chen, David Kotz, "Context-Sensitive Resource Discovery", 1st IEEE International Conference on Pervasive Computing and Communications (Percom'03), Fort Worth, Texas, March 2003

[CORB95] Object Management Group, "The Common Object Request Broker: Architecture and Specification. Revision 2.0", July 1995

[CPAJ01] Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, Anupam Joshi, "DReggie: Semantic Service Discovery for M-Commerce Applications", 1st Workshop on Reliable and Secure Applications in Mobile Environments, New Orleans, LA, USA, October 2001

[CPH07] Min-Xiou Chen, Chen-Jui Peng, Ren-Hung Hwang, "SSIP: Split a SIP Session over multiple Devices", Computer Standards Interfaces 29(5), pp. 531–545, 2007

[CSLB06] Jim Christensen, Jeremy Sussman, Stephen Levy, William E. Bennett, Tracee Vetting Wolf, Wendy A. Kellogg, "Too much Information", ACM Queue, Vol. 4, N°. 6, July-August 2006

[CuM03] Francisco Curbera, Nirmal Mukhi, "Metadata-Driven Middleware for Web Services", 4th International Conference on Web Information Systems Engineering (WISE'03), Rome, Italy, December 2003

[Cur07] Francisco Curbera, "Component Contracts in Service-Oriented Architectures", IEEE Computer Magazine, Volume 40, Issue 11, pp. 74-80, November 2007

[DaL06] Pierre-Charles David, Thomas Ledoux, "An aspect-oriented Approach for developing self-adaptive Fractal Components", 5th International Symposium on Software Composition (SC'06), Vienna, Austria, March 2006

[DaP02] Dan Davis, Manish Parashar, "Latency Performance of SOAP Implementations", 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'02), Berlin, Germany, May 2002

[DaZ02] Alexander Davis, Du Zhang, "A Comparative Study of DCOM and SOAP", 4th IEEE International Symposium on Multimedia Software Engineering (MSE'02), Newport Beach, CA, USA, December 2002

[DCOM98] Nat Brown, Charlie Kindel, "Distributed Component Object Model Protocol – DCOM/1.0", Internet Draft, Network working group, January 1998

[DFKM02] Pavlin Dobrev, David Famolari, Christian Kurzke, Brent A. Miller, "Device and Service Discovery in Home Networks with OSGi", IEEE Communications magazine, Volume 40, Issue 8, pp. 86-92, August 2002

[DHDS98] Bruno Dumant, François Horn, Frederic Dang Tran, Jean-Bernard Stefani, "Jonathan: an Open Distributed Processing Environment in Java", IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), Lake District, UK, September 1998

[Don07] Didier Donsez, "On-Demand Component Deployment in the UPnP Device Architecture", 4th IEEE Consumer Communications and Networking Conference (CCNC'07), Las Vegas, NV, USA, January 2007

[DMTF06] Distributed Management Task Force (DMTF), "CIM Schema: Application Model Specification V2.13", September 2006

[DPWS06] Microsoft Corporation, "Devices Profile for Web Services, February 2006", http://schemas.xmlsoap.org/ws/2006/02/devprof/

[DSA01] Anind K. Dey, Daniel Salber, Gregory D. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", International Journal of Human-Computer Interaction, Vol. 16, pp. 97-166, 2001

[DSTG07] Hylke W. van Dijk, Hans J. Scholten, Alvaro Tobalina, Victor Garcia Munoz, Stephane Milanini, Antonio Kung, "Open Home Networks: the TEAHA approach", 6th International Conference on Networking (ICN'07), Sainte-Luce, Martinique, France, April 2007

[EDH06] Clément Escoffier, Didier Donsez, Richard S. Hall, "Developing an OSGi-like Service Platform for .NET", 3rd IEEE Consumer Communications and Networking Conference (CCNC'05), Las Vegas, NV, USA, January 2006

[EHL07] Clément Escoffier, Richard S. Hall, Philippe Lalanda, "iPOJO: An extensible service-oriented component framework", 4th IEEE Service Computing Conference (SCC'07), Salt Lake City, UT, USA, July 2007

[FAYF04] Hiroshi Fujisawa, Katsunori Aoki, Makoto Yamamoto, Yoshihiro Fujita, "Estimation of multicast packet loss characteristic due to collision and loss recovery using FEC on distributed infrastructure wireless LANs", IEEE International Conference on Wireless Communications and Networking (WCNC'04), Atlanta, GA, USA, March 2004

[FEL07] Caroline Funk, Carolin Ehm, Claudia Linnhoff-Popien, "Support of stateful Services in Pervasive Environments", 5th IEEE International Conference on Pervasive Computing and Communications Workshops (PercomW'07), White Plains, NY, USA, March 2007.

[Fie00] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures", PhD thesis, University of California, Irvine, 2000

[FMK06] Caroline Funk, Jelena Mitic, Christoph Kuhmuench, "DSCL: A Language to support Dynamic Service Composition", 1st International Workshop on Service Integration in Pervasive Environments (SIPE'06), Lyon, France, June 2006

[FSLM02] Jean-Philippe Fassino, Jean-Bernard Stefani, Julia Lawall, Gilles Muller, "Think: A Software Framework for Component-based Operating System Kernels", 2002 USENIX Annual Technical Conference, Monterey, CA, USA, June 2002

[Fow04] Martin Fowler, "Inversion of Control Containers and the Dependency Injection pattern", 2004, http://martinfowler.com/articles/injection.html

[FRPD05] Cristina Feier, Dumitru Roman, Axel Polleres, John Domingue, Michael Stollberg, and Dieter Fensel, "Towards Intelligent Web Services: The Web Service Modeling Ontology (WSMO)", 1st International Conference on Intelligent Computing (ICIC'05), Hefei, China, August 2005

[FPV98] Alfonso Fuggetta, Gian Pietro Picco, Giovanni Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, Volume 24, Issue 5, pp. 342 – 361, May 1998

[FSLM07] Caroline Funk, Amelia Schultheis, Claudia Linnhoff-Popien, Jelena Mitic, Christoph Kuhmunch, "Adaptation of Composite Services in Pervasive Computing Environments", 4th IEEE International Conference on Pervasive Services (Pervasive'07), Istanbul, Turkey, July 2007

[GBS03] Paul Grace, Gordon S. Blair, Sam Samuel, "ReMMoC, A Reflective Middleware to Support Mobile Client Interoperability", 5th International Symposium on Distributed Objects, Middleware, and Applications (DOA'03), Catania, Sicily, Italy, November 2003

[GDSD04] 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

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", ed. Addison-Wesley, 1995

[GPVD99] Erik Guttman, Charles Perkins, John Veizades, Michael Day, "Service Location Protocol, Version 2", RFC 2608, June 1999

[GTA02] Rahul Gupta, Sumeet Talwar, Dharma P. Agrawal, "Jini home networking: a step toward pervasive computing", IEEE Computer Magazine, Volume 35, Issue 8, pp. 34-40, August 2002

[Hel02] Sumi Helal, "Standards for service discovery and delivery", IEEE Pervasive Computing Journal, Volume 1, Issue 3, pp. 95-100, July-Sept. 2002

[Hem06] Armijn Hemel, "Universal Plug and Play: Dead simple or simply deadly?", 5th System Administration and Network Engineering Conference, Delft, The Netherlands, May 2006

[HuS99] Galen C. Hunt, Michael L. Scott, "The Coign Automatic Distributed Partitioning System", 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI'99), New Orleans, LA, USA, February 1999

[HuS99-2] Galen C. Hunt, Michael L. Scott, "Intercepting and instrumenting COM applications", 5th USENIX Conference on Object-Oriented Technologies & Systems (COOTS'99), San Diego, CA, USA, May 1999

[IGRS06] IGRS Labs, "Intelligent Grouping and Resource Sharing Specification 1.0", 2006, http://www.igrs.org

[JaS05] F. Jammes, H. Smit. "Service-Oriented Architectures for Devices - the SIRENA View", 3rd IEEE Industrial Informatics Conference (INDIN'05), Perth, Australia, 2005

[JMX02] Sun Microsystems, "Java Management Extensions Instrumentation and Agent", Specification, v1.2, Maintenance Release, October 2002

[KeB03] Abdelmadjid Ketfi, Noureddine Belkhatir, "Dynamic Interface Adaptability in Service Oriented Software", 8th International Workshop on Component-Oriented Programming (WCOP'03), Darmstadt, Germany, July 2003

[Kei06] W. Keith Edwards, "Discovery systems in ubiquitous computing", IEEE Pervasive Computing, Volume 5, Issue 2, pp. 70 - 77, April-June 2006

[KeN75] Ralph L. Keeney, Keshavan Nair, "Decision Analysis for the Siting of Nuclear Power Plants-The Relevance of Multiattribute Utility Theory", IEEE, vol. 63, no. 3, March 1975

[Kep05] Jeffrey O. Kephart, "Research challenges of autonomic computing", 27th IEEE International Conference on Software engineering (ICSE'05), St. Louis, MO, USA, Mai 2005

[KGD00] Sumit Khurana, Provin Gurung, Ashutosh Dutta, "Device Message Protocol (DMP): An XML based format for Wide Area Communication with Networked Appliances", IETF Draft, draft-khurana-dmp-appliances-00.txt, November 2000

[KiN05] Robind kirk, Jan Newmarch, "A Location-Aware, Service-based Audio System", 2nd IEEE Consumer Communications and Networking Conference (CCNC'05), Las Vegas, NV, USA, January 2005

[KKS07] Swaroop Kalasapur, Mohan Kumar, Behrooz A. Shirazi, "Dynamic Service Composition in Pervasive Computing", IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 7, pp. 907-918, July 2007

[KNS01] W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy, "The Case for Recombinant Computing", Xerox Palo Alto Research Center Technical Report CSL-01-1, April 20, 2001

[Kra03] Sacha Krakowiac for ObjectWeb.org, "What is Middleware", http://middleware.objectweb.org/

[Kra08] Sacha Krakowiac, "Middleware Architecture with Patterns and Frameworks", June 2008, http://sardes.inrialpes.fr/%7Ekrakowia/MW-Book/main-onebib.pdf

[KrH04] Peter Kriens, BJ Hargrave for the OSGi Alliance, "Listeners considered harmful: The whiteboard pattern", Technical Whitepaper, August 2004

[KrS05] Sacha Krakowiak, Jean-Bernard Stefani, "export-bind, Un patron d'architecture pour la liaison adaptable, en Informatique Répartie", Sciences et Technologies, Chapter 9, ed. Hermès, 2005

[KuR06] Brijesh Kumar, Mahfuzur Rahman, "Mobility Support for UPnP devices using SIP", 3rd IEEE International Consumer Communications and Networking Conference (CCNC'06), Las Vegas, NV, USA, January 2006

[LAOS06] Steffen Lamparter, Anupriya Ankolekar, Daniel Oberle, Rudi Studer, Christof Weinhardt, "A Policy Framework for Trading Configurable Goods and Services in Open Electronic Markets", 8th International Conference on Electronic Commerce (ICEC'06), Fredericton, New Brunswick, Canada, August 2006

[Law00] Jamie Lawrence, "Ubiquitous annoyance", Communicate, Volume 5, Issue 2, pp. 54–59, December 2000

[LPR07] Marc Lacoste, Gilles Privat, Fano Ramparany, "Evaluating Confidence in Context for Context-Aware Security", 2nd European Conference on Ambient Intelligence, Darmstadt, Germany, November 2007

[MEX06] IBM, "Web Services Metadata Exchange (WS-MetadataExchange), Version 1.1", August 2006, http://download.boulder.ibm.com /ibmdl/pub/software/dw/specs/ws-mex/metadataexchange.pdf

[MHS05] Raluca Marin-Perianu, Pieter Hartel, Hans Scholten, "A Classification of Service Discovery Protocols" Technical Report TR-CTIT-05-25 Centre for Telematics and Information Technology, University of Twente, Enschede, ISSN 1381-3625, June 2005

[MIDP02] Java Community Process, "Mobile Information Device Profile for Java™ 2 Micro Edition Version 2.0", November 2002

[MiT04] Ludmil Mikhailov, Petco Tsvetinov, "Evaluation of services using a fuzzy analytic hierarchy process", Applied Soft Computing, Volume 5, Issue 1, pp. 23-33, December 2004

[MKF02] Nirmal Mukhi, Rania Khalaf, and Paul Fremantle, "Multi-protocol Web Services for Enterprises and the Grid", 2nd EuroWeb Conference, Oxford, UK, December 2002

[MMG02] Stan Moyer, Dave Marples, Abhrajit Ghosh, "Service Portability of Networked Appliances", IEEE Communications Magazine, February 2002

[MMT01] Stan Moyer, Dave Marples, and Simon Tsang, "A Protocol for Wide-Area Secure Networked Appliance Communication", IEEE Communications Magazine, October 2001

[MPR06] Amy L. Murphy, Gian Pietro Picco, Gruia-Catalin Roman, "LIME: A coordination model and middleware supporting, mobility of hosts and agents", ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 15, number 3, pp. 279–328, July 2006

[MSSG08] Abdelgadir E. Ibrahim, Liping Zhao, "Supporting the OSGi Service Platform with Mobility and Service Distribution in Ubiquitous Home Environments", IEEE Computer Magazine, 2008

[MUWS05] William Vambenepe for the Organization for the Advancement of Structured Information Standards (OASIS), "Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 2", March 2005

[Netc06] Internet Engineering Task Force (IETF), “Network Configuration Protocol”, RFC 4741, December 2006

[NPD07] Apostolos E. Nikolaidis, Serafeim Papastefanos, Gregory A. Doumenis, George I. Stassinopoulos, Marios Polichronis K. Drakos, "Local and Remote Management Integration for Flexible Service Provisioning to the Home", IEEE Communications Magazine, October 2007

[NTKR06] Jin Nakazawa, Hideyuki Tokuda, W. Keith Edwards, Umakishore Ramachandran, "A Bridging Framework for Universal Interoperability in Pervasive Systems", 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), Lisboa, Portugal, July 2006

[ODP95] ISO/IEC 10746-3:1996 - ITU-T Recommendation X.903, "Open Distributed Processing - Reference Model - Part 3: Architecture", January 1995

[OMAD05] Open Mobile Alliance (OMA), "OMA Device Management Protocol", Draft Version 1.2, April 2005

[OSGi05] OSGi Alliance, "OSGi R4 Core Specification and Service Compendium", October 2005

[OuP07] Laurent Ouakil, Guy Pujolle, "Téléphonie sur IP", Ed. EYROLLES, March 2007

[OWL04] World Wide Web Consortium, "OWL Web Ontology Language Reference", W3C Recommendation, February 2004, http://www.w3.org/TR/owl-ref/

[OzV99] Tamer Özsu, Patrick Valduriez, "Principles of Distributed Database Systems", 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1999

[PaF08] Pierre Parrend, Stephane Frenot, "Component-based Access Control: Secure Software Composition through Static Analysis", 7th International Symposium on Software Composition (SC'08), Budapest, Hungary, March 2008

[Pap03] Michael P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions", 4th International Conference on Web Information Systems Engineering (WISE'03), Rome, Italy, December 2003

[Pas01] Robert Pascoe, "Building Networks on the Fly", IEEE Spectrum, Volume 38, Issue 3, pp.61-65, March 2001

[PaT06] Pravin Pawar, Andrew Tomakoff, "Ontology-based Context-Aware Service Discovery for Pervasive Environments", 1st International Workshop on Service Integration in Pervasive Environments, Lyon, France, June 2006

[PoF03] Shankar R. Ponnekanti, Armando Fox. "Application-Service Interoperation without Standardized Interfaces", 1st IEEE International Conference on Pervasive Computing and Communications (Percom'03), Fort Worth, TX, USA, March 2003

[POS06] Juraj Polakovic, Ali Erdem Özcan, Jean-Bernard Stefani, "Building Reconfigurable Component-Based OS with Think", 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, Dubrovnik, Croatia, September 2006

[RAR07] Jan S. Rellermeyer, Gustavo Alonso, Timothy Roscoe, "R-OSGi: Distributed Applications through Software Modularization", 8th International Middleware Conference (Middleware'07), Newport Beach, CA, November 2007

[RCCI05] Pierre-Guillaume Raverdy, Rafik Chibout, Agnès de La Chapelle, Valérie Issarny, "The MSDA Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments", 6th International Middleware Conference (Middleware'07), Demonstration proposal, Grenoble, France, 2005

[Ric00] Golden G. Richard III, "Service Advertisement and Discovery: Enabling Universal Device Cooperation", IEEE Internet Computing Journal, Volume 4, Issue 5, pp. 18-26, September-October 2000

[RoF07] Yvan Royon, Stéphane Frénot, "A Survey of Unix Init Schemes", Technical Report, Inria RT-0338, June 2007

[Rot06] Arnon Rotem-Gal-Oz, "Fallacies of Distributed Computing Explained", 2006, http://www.rgoarchitects.com/Files/fallacies.pdf

[Roy07] Yvan Royon, "Environnements d'exécution pour passerelles domestiques", PhD Thesis, Institut National des Sciences Appliquées de Lyon, Lyon, France, December 2007

[SALU99] Salutation Consortium, "Salutation Architecture Specification Version 2.0c", June 1999

[Sat01] Mahadev Satyanarayanan, IEEE Personal Communications, Volume 8, Issue 4, pp. 10-17, August 2001

[SCOM08] Open Mobile Alliance (OMA), "Software Component Management Object", Draft Version 1.0, June 2008

[SIP02] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, Eve Schooler, "SIP: Session Initiation Protocol", IETF RFC 3261, June 2002

[SNMP02] David Harrington, Randy Presuhn, Bert Wijnen for the Internet Engineering Task Force (IETF), "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", RFC 3411, Standard 62, December 2002

[Sta02] Vince Stanford, "Using Pervasive Computing to Deliver Elder Care", IEEE Pervasive Computing, Volume 1, Issue 1, pp. 10-13, January-March 2002

[StC05] Daniel Steinberg, Stuart Cheshire, "Zero Configuration Networking: The Definitive Guide", O'Reilly, 1st Edition, December 2005

[Szy98] Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, Harlow, England 1998

[Tha94] Satish Thatté, "Automated synthesis of interface adapters for reusable classes", 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94), Portland, OR, USA, January 1994

[Tho06] Michael S. Thompson, "Service Discovery in Pervasive Environments", PhD thesis, University of Virginia, Blacksburg, Virginia, July 2006

[TiS02] Eli Tilevich, Yannis Smaragdakis, "J-Orchestra: Automatic Java Application Partitioning", 16th European Conference on Object-Oriented Programming (ECOOP'02), Melaga, Spain, June 2002

[TiS03] Eli Tilevich, Yannis Smaragdakis, "NRMI: Natural and Efficient Middleware", 23rd IEEE International Conference on Distributed Computer Systems (ICDCS'03), Providence, RI, USA, May 2003

[TMMS00] Simon Tsang, Stan Moyer, Dave Marples, Henning Schulzrinne, Arjun Roychowdhury, "SIP Extensions for Communicating with Networked Appliances," Internet Draft drafttsang-sip-appliances-do-00.txt, November 2000

[TMM01] Simon Tsang, Dave Marples, Stan Moyer, "Accessing Networked Appliances using the Session Initiation Protocol", 8th IEEE International Conference on Communications (ICC'01), Helsinki, Finland, June 2001

[TR6904] Broadband Forum, "CPE WAN Management Protocol (Technical Report 69)," May 2004

[UAV02] UPnP Forum, "UPnP AV Architecture:0.83", June 2002

[UDDI02] Organization for the Advancement of Structured Information Standards (OASIS), "UDDI Version 2 API Specification", July 2002

[UPnP06] UPnP Forum, "UPnP Device Architecture version 1.0", July 2006

[VaF02] Dimitar Valtchev, Ivailo Frankov, "Service gateway architecture for a smart home", IEEE Communications Magazine, Volume 40, Issue 4, pp. 126-132, April 2002

[Vay01] Julien Vayssiere, "Transparent Dissemination of Adapters in Jini", 3rd International Symposium on Distributed Objects, Middleware, and Applications (DOA'01), Rome, Italy, September 2001

[VCC07] Antonio Vilei, Gabriella Convertino, Fabrizio Crudo, "UPnP Architecture for Distributed Video Voice over IP Applications", 4th IEEE International Consumer Communications and Networking Conference, (CCNC'07), Las Vegas, NV, USA, January 2007

[VFL05] Mathieu Vallée, Fano Ramparany and Laurent Vercouter, "Flexible Composition of Smart Device Services", 1st International Conference on Pervasive Systems and Computing (PSC'05), Las Vegas, NV, USA, June 2005

[Wal98] Jim Waldo, "Remote procedure calls and Java Remote Method Invocation", IEEE Concurrency, Volume 6, Issue 3, pp. 5-7, July-September 1998

[Wal99] Jim Waldo, "The Jini architecture for network-centric computing", Communications of the ACM, Volume 42, Issue 7, pp. 76-82, July 1999

[Wei91] Mark Weiser, "The computer for the 21st century", Scientific American, 265(3):66-75, September 1991

[WHWC04] Steve R. White, James E. Hanson, Ian Whalley, David M. Chess, and Jeffrey O. Kephart, "An architectural approach to autonomic computing", 1st IEEE International Conference on Autonomic Computing (ICAC'04), New York, NY, May 2004

[WSMa06] Distributed Management Task Force, "Web Services for Management (WS-Management) v 1.0.0", February 2008

[WSMO06] WSMO Working Group, "Web Services Modeling Ontology, Final Draft", October 2006, http://www.wsmo.org/TR/d2/v1.3/

[WSP07] W3C, "Web Services Policy 1.5 – Framework", W3C Candidate Recommendation, June 2007, http://www.w3.org/TR/ws-policy/

[WWWK94] Jim Waldo, Geoff Wyant, Ann Wollrath, Samuel C. Kendall, "A Note on Distributed Computing", Sun Microsystems Technical Report, November 1994

[YuC97] Weimin Yu, and Alan Cox, "Java/DSM: A Platform for Heterogeneous Computing", Concurrency: Practice and Experience, 9(11):1213-1224, 1997

[YSHT07] Ming-Chien Yang, Norman Sheng, Brandon Huang, Jethro Tu, "Collaboration of Set-Top-Box and Residential Gateway plartforms", IEEE Transactions on Consumer Electronics, Volume 53, Issue n°3, pp. 905-910, August 2007.

[ZBBG07] Elmar Zeeb, Andreas Bobek, Hendrik Bohn, Frank Golatowski, “Lessons learned from implementing the Devices Profile for Web Services”, Digital EcoSystems and Technologies Conference (DEST'07), Inaugural IEEE-IES, Cairns, Australia, February 2007

[ZMN05] Feng Zhu, Matt W. Mutka, and Lionel M. Ni, "Service Discovery in Pervasive Computing Environments", IEEE Pervasive Computing, vol. 4, pp. 81-90, 2005

[ZZMN05] Feng Zhu, Wei Zhu, Matt W. Mutka, Lionel M. Ni, "Expose or Not? A Progressive Exposure Approach for Service Discovery in Pervasive Computing Environments", 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'05), Kauai Island, HI, USA, March 2005