HAL
open science

Politique de Liaison aux Services Intermittents dirigée par les Accords de Niveau de Service

Lionel Touseau

To cite this version:

Lionel Touseau. Politique de Liaison aux Services Intermittents dirigée par les Accords de Niveau de Service. Informatique [cs]. Université Joseph-Fourier - Grenoble I, 2010. Français. NNT : . tel-00520016

HAL Id: tel-00520016

https://theses.hal.science/tel-00520016

Submitted on 22 Sep 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université de Grenoble

Rapport scientifique présenté pour l'obtention du Doctorat

Spécialité : Informatique

Lionel Touseau

Sujet du mémoire

Politique de Liaison aux Services Intermittents dirigée par les Accords de Niveau de Service

Soutenue le 25 Mai 2010

devant le jury

Président	M. Jean-François Méhaut, Professeur des Universités à l'Université de Grenoble
Rapporteurs	M. Lionel Seinturier, Professeur des Universités à l'Université de Lille 1
	M. Stéphane Frénot, Maître de Conférences à l'INSA Lyon
Examinateurs	M. Vincent Olive, Chercheur, CEA
	M. Jean-Philippe Vasseur, Ingénieur, Cisco
	M. Didier Donsez, Professeur des Universités à l'Université de Grenoble (directeur de thèse)
Invité	M. David Menga, Ingénieur, EDF R&D

Remerciements

Tout d'abord je tiens à remercier les membres du jury: Jean-François Méhaut, qui a accepté de présider le jury, Lionel Seinturier et Stéphane Frénot qui ont rapporté ce manuscrit malgré des délais contraints, Vincent Olive et Jean-Phillipe Vasseur qui ont apporté un regard externe intéressant sur mon travail, ainsi que David Menga qui, même s'il n'a pas pu être présent lors de la soutenance, m'a fait part de son point de vue sur le thème du bâtiment intelligent.

Et je remercie bien entendu Didier Donsez qui a dirigé ma thèse. Même si suivre Didier a parfois été ardu du fait de son ouverture sur des sujets divers, cette même ouverture m'a permis de me sensibiliser à de nombreux domaines d'applications et ainsi élargir ma vision de l'informatique.

Je remercie également Jacky Estublier, Philippe Lalanda et Pierre-Yves Cunin qui m'ont permis d'effectuer ma thèse au sein de l'équipe Adèle, ainsi que tous les collègues de l'équipe pour les discussions scientifiques aussi bien que les bons moments partagés. En particulier Walter et Kiev de la D.D.Team, et bien sûr Mikael qui fut en quelque sorte mon prédécesseur.

D'ailleurs, je tiens aussi à remercier Humberto dont les travaux ont ouvert la voie aux composants orientés service, et qui m'a offert un accueil chaleureux au Mexique (une petite pensée pour la famille Cervantes qui s'est agrandie avec Julian et Alexis). Cette thèse a également bénéficié des travaux de Clément qui a continué sur la lancée d'Humberto et que je remercie.

Et merci à Antonin, Johann, Cristina, Jianqi, Stéphanie, Thomas avec qui j'ai partagé mon bureau, ainsi qu'à tous les autres thésards et ingénieurs.

En dehors du cadre professionnel, je tiens à m'excuser auprès de ma famille et de mes amis franciliens pour qui je n'ai malheureusement pas été très disponible, et les remercie de leur soutien. Un grand merci à la famille Blanes, et aux Dékazé grenoblois avec qui j'ai pris plaisir à me changer les idées et à gagner le GBBLPT'2009.

Enfin, la meilleure pour la fin, la cerise sur l' ありがとう, un gros merci à Laureline qui m'a soutenu au quotidien, et elle-même sait que ça n'a pas toujours été facile. MERCI !

Résumé

L'informatique s'est récemment développée autour de deux axes : l'informatique ambiante d'une part avec la multiplication des objets communicants, et l'internet des services d'autre part suite à l'essor parallèle des centres de traitement de données et d'Internet. Dans ces domaines, la disponibilité fluctuante des ressources, qui entraîne une intermittence des services fournis, représente désormais une préoccupation majeure dans la conception d'applications.

La programmation orientée composants appliquée aux architectures orientées service simplifie la gestion des liaisons de service via des politiques. Néanmoins les politiques de liaison existantes suivent soit une approche statique interdisant alors toute reconfiguration d'architecture en cours d'exécution, soit une approche dynamique ne garantissant pas une stabilité minimale de l'architecture dans le cas de services intermittents.

Cette thèse propose un compromis entre stabilité architecturale et dynamisme en plaçant l'interruption de service au centre des préoccupations du concepteur. La politique de liaison résultante offre ainsi une tolérance aux interruptions de service jusqu'à une certaine limite au-delà de laquelle l'architecture est reconfigurée dynamiquement. Afin de situer cette limite, notre proposition se base sur l'utilisation d'accords de niveau de service, une forme enrichie des contrats de service permettant, entre autres, l'expression de contraintes sur la disponibilité des services.

L'approche a été expérimentée sur la plate-forme à services OSGi en étendant les mécanismes de gestion des liaisons du modèle à composants iPOJO, puis validée dans le contexte de l'informatique ambiante ainsi que sur le serveur d'applications JOnAS.

Mots clé : architecture dynamique orientée service, disponibilité, service intermittent, composant orienté service, politique de liaison, accord de niveau de service

Abstract

Lately, the evolution of information technologies has been following two trends. On the one hand the proliferation of communicating devices contributes to the creation of an ambient intelligence. On the other hand, the booming of Internet associated with the rapid growth of data centres capabilities results in the emergence of an internet of services. In both domains, application design is challenged by the dynamic availability of computing resources and data.

The combination of component-based software engineering and service-oriented computing techniques allows service bindings to be driven by policies. However, for the time being, policies either follow a dynamic approach which does not suit the needs of architectural stability when dealing with intermittent services, or a static approach which does not allow dynamic reconfiguration.

The work presented in this thesis proposes a trade-off between the two approaches by considering service disruptions as a major concern. The proposed binding policy relies on service level agreements to be disruption-tolerant, since service-level agreements allow expressing and enforcing obligations regarding availability and quantified disruptions.

This approach has been implemented on the OSGi service platform and iPOJO, a service-oriented component model for OSGi. iPOJO service dependency management has been extended in order to support our policy. The latter was validated both in the context of ambient intelligence, and on open-source and OSGi-based JOnAS application server.

Keywords : dynamic service-oriented architecture, availability, intermittent service, service-oriented component, binding policy, service-level agreement

Sommaire

CHAPITRE I INTRODUCTION **10**
I.1 Contexte *11*
I.2 Problématique *21*
I.3 Proposition *25*
I.4 Structure du document *27*

PREMIERE PARTIE : ETAT DE L'ART **29**

CHAPITRE II PLATES-FORMES DYNAMIQUES A SERVICES **30**
II.1 Préambule sur l'approche à services *31*
II.2 Architectures orientées service *36*
II.3 Architectures dynamiques orientées service *44*
II.4 Composants orientés services *53*
II.5 Synthèse *65*
CHAPITRE III ACCORDS DE NIVEAU DE SERVICE **68**
III.1 Fondements des accords de niveau de service *69*
III.2 Principes généraux des accords de niveau de service *74*
III.3 Etat de l'art *85*
III.4 Synthèse *95*

DEUXIEME PARTIE : PROPOSITION **101**

CHAPITRE IV INTERRUPTIONS ET SERVICES INTERMITTENTS **102**
IV.1 Interruptions de service *103*
IV.2 Service intermittent *114*
CHAPITRE V POLITIQUE DE LIAISON DIRIGEE PAR LES ACCORDS DE NIVEAU DE SERVICE **119**
V.1 Vers une politique de liaison consciente des interruptions *120*
V.2 Caractérisation des interruptions par les accords de niveau de service *126*
CHAPITRE VI REALISATION **132**
VI.1 Architecture générale *133*
VI.2 Supervision des interruptions de services *134*
VI.3 Support des accords de niveau de service *135*
VI.4 Gestion des liaisons par le conteneur *139*

TROISIEME PARTIE : EXPERIMENTATION ET RESULTATS **141**

CHAPITRE VII IMPLEMENTATION **142**
VII.1 Choix de la plate-forme cible *143*
VII.2 Gestion des accords de niveau de service *145*
VII.3 Gestion des liaisons *148*

VII.4 Outil de supervision *153*
CHAPITRE VIII VALIDATION **154**
VIII.1 Méthodologie de migration *155*
VIII.2 Serveur d'application JOnAS *157*
VIII.3 AspireRFID ~ Chaine d'approvisionnement à température maitrisée ... *160*
VIII.4 Passerelle domestique H-Omega *166*
CHAPITRE IX PERSPECTIVES ET CONCLUSION **169**
IX.1 Perspectives *170*
IX.2 Conclusion *177*
CHAPITRE X BIBLIOGRAPHIE **180**
ANNEXES **198**
X.1 Annexe A Bundles et composants iPOJO *199*
X.2 Annexe B Méta-données iPOJO DSLA pour l'application de contrôle du froid dans AspireRFID *201*
X.3 Annexe C Descripteur de WireApp pour AspireRFID *204*

Table des Figures

FIGURE 1. DES OBJETS COMMUNICANTS AUX SYSTEMES D'INFORMATION 11
FIGURE 2. TENDANCE A LA DIFFUSION DE L'INFORMATIQUE SUR LES 50 DERNIERES ANNEES (ADAPTE DE [WALDNER2007]) 13
FIGURE 3. ROBOTS AUTONOMES: ROVIO ET SPYKEE (A GAUCHE), WALL-E (AU CENTRE) ET UN ROVER MARTIEN (A DROITE) 16
FIGURE 4. CONFIGURATION DE GRILLE « VERTE » ADAPTABLE A LA DEMANDE 19
FIGURE 5. LE CONSOMMATEUR C DU SERVICE S EST LIE AU PRESTATAIRE P. 30
FIGURE 6. VAGUES ET TENDANCES DU GENIE LOGICIEL [DONSEZ2006] 33
FIGURE 7. MOTIF D'INTERACTION D'UNE ARCHITECTURE ORIENTEE SERVICE 36
FIGURE 8. DIAGRAMME DE SEQUENCE UML DU PATRON SOC 37
FIGURE 9. SCHEMA PYRAMIDAL REPRESENTANT UNE SOA ETENDUE 38
FIGURE 10. ORCHESTRATION (A GAUCHE) ET CHOREGRAPHIE (A DROITE) [PEDRAZA2009A] .. 39
FIGURE 11. PILE TECHNOLOGIQUE DES SERVICES WEB 39
FIGURE 12. UDDI : DE LA THEORIE (A GAUCHE) A LA PRATIQUE (A DROITE) 41
FIGURE 13. INTERACTIONS GUIDANT LES ARCHITECTURES DYNAMIQUES ORIENTEES SERVICE . 45
FIGURE 14. RESEAU DOMESTIQUE UPNP 48
FIGURE 15 A) ET B). PILE DES PROTOCOLES UPNP (A GAUCHE) ET DPWS (A DROITE) 49
FIGURE 16. LA PLATE-FORME A SERVICES OSGI, SURCOUCHE MODULAIRE A JAVA (D'APRES PETER KRIENS) 52
FIGURE 17. REPRESENTATION DES VUES INTERNES ET EXTERNES D'UN COMPOSANT 54
FIGURE 18. STRUCTURE DE COMPOSANTS IPOJO ET INTERACTIONS ENTRE CONSOMMATEUR ET FOURNISSEUR DE SERVICE 60
FIGURE 19. IPOJO SUPPORTE LA COMPOSITION FONCTIONNELLE (EN-HAUT) ET STRUCTURELLE (EN-BAS) 61
FIGURE 20. MODELE D'ASSEMBLAGE ET D'IMPLEMENTATION DE SCA 63
FIGURE 21. LES QUATRES NIVEAUX DE CONTRAT 70
FIGURE 22. TENDANCE DES DOMAINES D'APPLICATION DES SLA DEPUIS PLUS DE 30 ANS 71
FIGURE 23. REPRESENTATION UML DU MODELE DE CONTENU D'UN ACCORD DE NIVEAU DE SERVICE 75
FIGURE 24. CYCLE DE VIE D'UN ACCORD DE NIVEAU DE SERVICE 78
FIGURE 25. NIVEAUX DE NEGOCIATION 80
FIGURE 26. GESTIONNAIRE DE NIVEAU DE SERVICE ET SONDES 82
FIGURE 27. BOUCLE AUTONOMIQUE MAPE-K 83
FIGURE 28. REPRESENTATION UML DES ELEMENTS DE WSLA [LUDWIG ET AL 2003] 86
FIGURE 29. SCHEMA D'INTERACTION DU COMPLIANCE MONITOR POUR WSLA 87
FIGURE 30. CONTENU D'UN ACCORD WS-AGREEMENT 88
FIGURE 31. ARCHITECTURE EN COUCHES DE L'APPROCHE RULE BASED SLA [PASCHKE2006] .91
FIGURE 32. NOTIONS DE TIME TO REPAIR (TTR), TIME TO FAILURE (TTF) ET TIME BETWEEN FAILURE (TBF) 104
FIGURE 33. PERTE DU FLOT D'EXECUTION SUITE A UNE RECONFIGURATION DYNAMIQUE 112
FIGURE 34. CHAINE DE COMPOSITION DE SERVICES DE RANG N 113
FIGURE 35. SCHEMA UML DE CARACTERISATION DES SERVICES INTERMITTENTS 115
FIGURE 36. CYCLE DE VIE D'UN SERVICE INTERMITTENT 116
FIGURE 37. CONFIGURATION SIMPLE D'APPLICATION MULTI-FOURNISSEURS 122

FIGURE 38. AUTOMATE DES RECONFIGURATIONS D'ARCHITECTURE SUIVANT LA POLITIQUE DSLA 122
FIGURE 39. AUTOMATES RESULTANTS DES DIFFERENTES POLITIQUES DE LIAISON 124
FIGURE 40. POSITIONNEMENT DES DIFFERENTES POLITIQUES DE LIAISON 125
FIGURE 41. META-MODELE DES SLO PORTANT SUR LA DISPONIBILITE DES SERVICES INTERMITTENTS 127
FIGURE 42. ETAPES DE NEGOCIATION D'UN SLA 129
FIGURE 43. REPRESENTATION DU GESTIONNAIRE DSLM SOUS FORME D'UN GESTIONNAIRE AUTONOMIQUE 130
FIGURE 44. ARCHITECTURE GENERALE 133
FIGURE 45. UTILISATION DU DISRUPTIONSLOGGER 134
FIGURE 46. DSLA MANAGER 135
FIGURE 47. DIAGRAMME DE SEQUENCE UML DU PROCESSUS DE NEGOCIATION 137
FIGURE 48. UTILISATION DU DSLA MANAGER PAR LE CONTENEUR POUR LA GESTION DES LIAISONS DE SERVICE 140
FIGURE 49. OSGI WIREADMIN 150
FIGURE 50. LANGAGE WADL POUR LE WIREADMINBINDER 151
FIGURE 51. CAPTURES D'ECRAN DU GREFFON DSLA POUR LA JVISUALVM 153
FIGURE 52. DEUX SERVICES TECHNIQUES DE JONAS : DATABASE MANAGER ET TRANSACTION MANAGER 158
FIGURE 53. ARCHITECTURE DE L'INTERGICIEL ASPIRERFID POUR UNE CHAINE D'APPROVISIONNEMENT [DONSEZ2009B] 161
FIGURE 54. COMPOSANTS D'UN EDGE ASPIRERFID DEPLOYE SUR UNE PASSERELLE OSGI 161
FIGURE 55. APPLICATION DE CONTROLE DU FROID 163
FIGURE 56. COURBES DE SUIVI DE LA TEMPERATURE 165
FIGURE 57. H-OMEGA AGREGE DES SERVICES DISTRIBUES GRACE AU REMOTE SERVICE MANAGER. 166
FIGURE 58. APPLICATION MEDICALE UTILISANT LA PERSISTANCE D'H-OMEGA 168
FIGURE 59. SELECTION DE SERVICE SELON L'HISTORIQUE DES INTERRUPTIONS ET LA DISPONIBILITE GARANTIE. 172

Chapitre I

Introduction

> "*Software is invisible to most of the world. Although individuals, organizations, and nations rely on a multitude of software-intensive systems every day, most software lives in the interstitial spaces of society, hidden from view except insofar as it does something tangible or useful.*"
> Grady Booch

Les travaux présentés dans cette thèse se placent dans deux tendances récentes des technologies de l'information : l'internet des choses, et l'internet des services dont l'informatique en nuage est le principal représentant. Ces deux pans de l'informatique ont évolué en parallèle avec le développement d'internet d'une part et l'essor des objets communicants d'autre part. Dans ces deux contextes la garantie d'une disponibilité des ressources est devenue une préoccupation importante dans la mise en œuvre des applications. L'approche orientée service présente des avantages permettant d'aborder la complexité de ces domaines où les applications doivent être capables de s'adapter dynamiquement, c'est-à-dire de se reconfigurer automatiquement à l'exécution.

Cette thèse s'intéresse donc aux solutions existantes de gestion des liaisons de service et à leurs limites lorsqu'elles sont appliquées à des systèmes où l'interruption de service est un phénomène courant. Elle propose une politique de liaison s'appuyant sur des accords de niveau de service ciblant la disponibilité et les interruptions de services. Cette politique permet ainsi la conception d'applications à la fois réactives, mais faisant néanmoins preuve d'une certaine stabilité architecturale face à des services intermittents.

L'utilisation des accords de niveau de service est aussi un moyen pour les applications d'avoir un certain niveau de garantie sur la disponibilité des services utilisés.

Ces travaux ont été réalisés au sein de l'équipe Adèle, spécialisée dans le génie logiciel et les intergiciels pour les plateformes de déploiement et d'exécution d'applications à composants et à services.

I.1 Contexte

Avant d'aborder le contexte de cette thèse, il est d'abord nécessaire de situer l'informatique de 2010, près d'un demi-siècle après ses débuts, en dégageant les tendances de son évolution. A partir de cette analyse nous pourrons positionner ce travail par rapport aux besoins émergents.

D'un côté l'informatique se fait de plus en plus présente au quotidien à travers les objets courants [Weiser1991] et on commence à parler d'intelligence ambiante. De l'autre, la croissance d'Internet, la facilité d'accès au haut débit, le développement des applications web riches entraînent la virtualisation des ressources informatiques (infrastructures, applications, ...) et on semble se diriger vers une informatique « à la demande » organisée en nuages (*Cloud computing* en référence à la représentation d'internet).

Ces deux branches d'évolution ne sont pas totalement disjointes, puisque si la taille des équipements informatiques accompagnant les usagers a évolué vers la mobilité, les machines sont de plus en plus connectées, à la fois entre elles et à l'Internet. La Figure 1 schématise une vision d'ensemble de cette informatique où des réseaux de capteurs, des réseaux domestiques et d'entreprise collaborent avec des systèmes d'information.

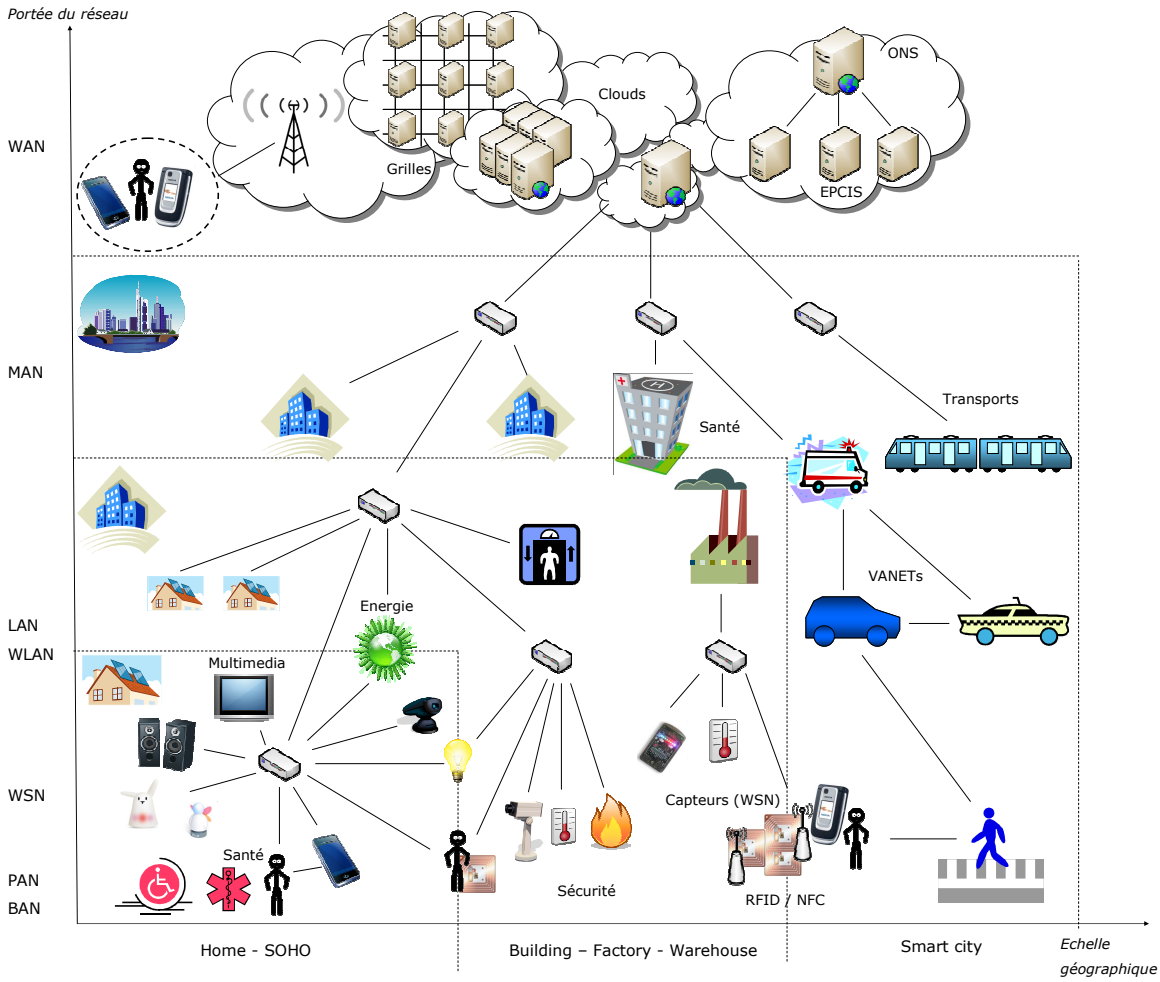

Figure 1. Des objets communicants aux systèmes d'information

Dernièrement, les approches orientées service ont été appliquées à ces différents domaines d'application. En effet, lorsque les interconnexions entre systèmes sont nombreuses il est intéressant de tirer profit du découplage offert par l'approche à service qui permet l'évolution indépendante des différents systèmes.

Le service peut être en effet un moyen de représenter les fonctionnalités d'un objet physique, alors utilisable par d'autres objets ou par des systèmes informatiques. Mais il permet également de mettre à disposition des ressources ou des applications distantes à travers une interface d'utilisation Les services web sont typiquement utilisés pour l'accès aux fonctionnalités d'une application sur internet. On parle désormais d'*Internet des services* en plus de l'*Internet des choses* [ITU2005]. Les notions de spécifications et de contrats de service sont développées dans l'état de l'art consacré aux services.

Enfin, une dernière préoccupation actuelle concerne l'empreinte énergétique des systèmes informatiques. L' « informatique verte » concerne à la fois les systèmes d'information et centres de données où l'informatique à la demande permet d'ajuster l'offre à la demande; ainsi que les réseaux pervasifs qui représentent un moyen de diminuer la consommation d'énergie via l'utilisation de capteurs.

I.1.1 Vers une intelligence ambiante

Ces dernières années ont été marquées par l'accroissement et la dissémination des ordinateurs et logiciels dans notre environnement. Du simple lecteur de musique numérique portable au système de navigation d'un avion de ligne, ces objets intelligents parsèment le monde physique dans lequel nous vivons et souvent de manière transparente, par opposition à l'ordinateur personnel (PC) fixe. On retrouve les notions de transparence et de dissémination dans les termes d'*informatique ubiquitaire, enfouie ou pervasive* [Weiser1993]. Parmi les éléments qui composent l'informatique ubiquitaire nous nous intéresserons plus particulièrement aux objets communicants, base des interactions machine-à-machine (M2M) et de l'Internet des Choses qui contribuent à créer une intelligence ambiante [Williams2008, Rellermeyer et al 2008]. Un objet communicant (contrairement au simple lecteur de musique portable du début des années 2000 évoqué plus haut) a la particularité de pouvoir interagir avec d'autres objets ou logiciels, et de pouvoir ainsi être intégré à diverses applications, éventuellement de façon opportuniste.

La Figure 2 présentée ci-dessous adaptée de [Waldner2007] illustre ces propos en montrant l'évolution des systèmes informatiques sur les 50 dernières années. La réduction de la taille physique des composants de stockage et de calcul, couplée aux progrès dans la gestion de l'énergie (batteries au lithium, processeurs faible consommation, …) et à l'essor des modes de communication sans fil, ont ouvert la voie à une informatique nomade de plus en plus mobile. L'ordinateur personnel est alors relégué au rôle d'objet communicant parmi tant d'autres.

Le développement des communications sans-fil d'une part et les progrès en micro-électronique poussant à la miniaturisation des processeurs et supports de stockage d'autre part, ont permis de créer des objets de taille réduite pouvant communiquer entre eux et embarquant des capacités de calcul et de stockage dédiés à des tâches spécifiques ou bien

généralistes. Les protocoles de communication basés sur les ondes radio utilisant des antennes émettrices et réceptrices se font de plus en plus nombreux : BlueTooth (IEEE 802.15.1), Zigbee et 6LoWPAN[1] (IEEE 802.15.4), WiFi (IEEE 802.11), WiMax (IEEE 802.16), UWB (IEEE 802.15.3a), FlashOFDM (IEEE 802.20), RFID, NFC, 3G, … Ces communications dites « par les airs » permettent non seulement d'intégrer des objets communicants à des systèmes plus vastes, mais aussi de les relier à l'Internet, d'où l'appellation d'Internet des Choses. Ainsi un système pervasif, malgré les capacités de calcul et de stockage réduites des objets qui le composent, peut profiter des avancées d'autres domaines telles que les grilles de calcul (*Grid computing*), l'informatique utilitaire (*Utility computing*) et l'informatique en nuages (*Cloud computing*) qui offrent des capacités bien plus importantes.

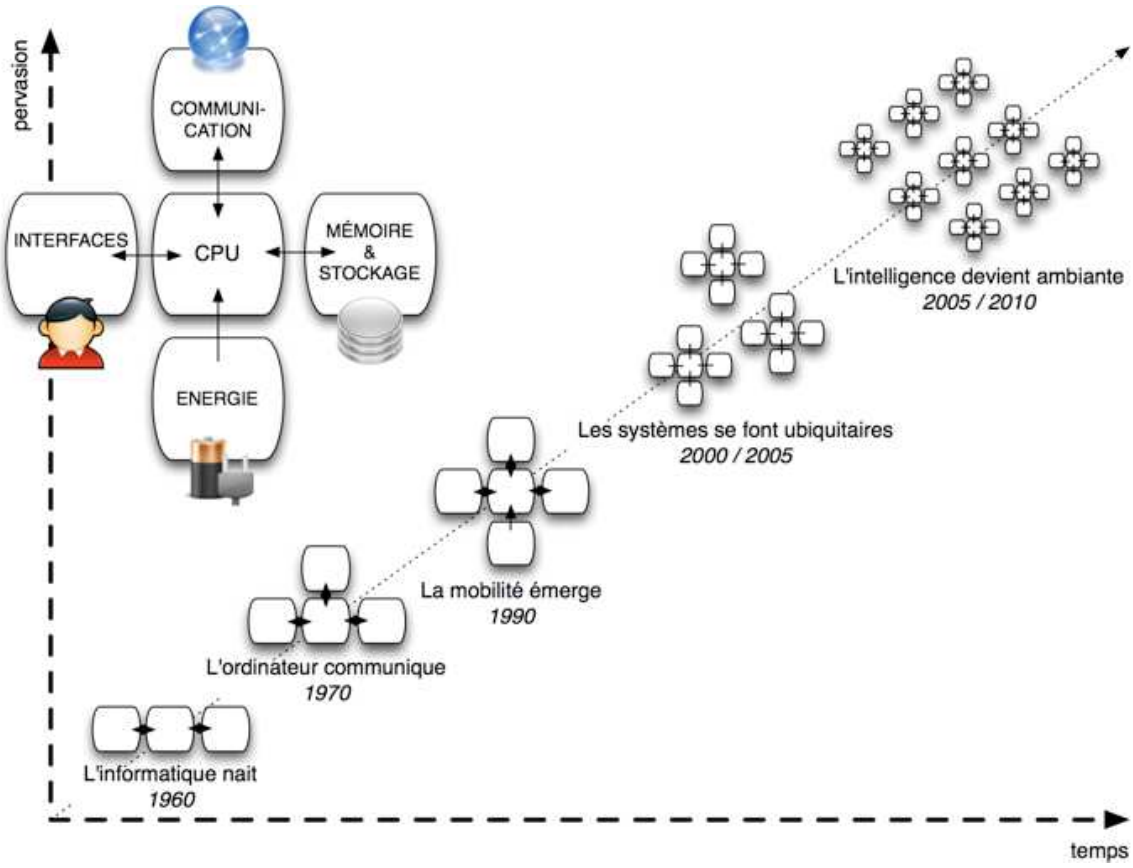

Figure 2. Tendance à la diffusion de l'informatique sur les 50 dernières années (adapté de [Waldner2007])

Cette tendance à l'informatisation de l'environnement par la miniaturisation et le développement des antennes radio est déjà visible. La RFID (*Radio Frequency IDentification*) fait partie des prémices de l'Internet des Choses [Yan et al 2008]. La technologie RFID [Sheng2008] permet d'identifier chaque objet de manière unique grâce à un système d'étiquettes électroniques pouvant être alimentées pour lire (et écrire) dans la mémoire à distance grâce à un champ électromagnétique. Au delà des problèmes d'éthiques concernant la sécurité et le respect de la vie privée, les possibilités offertes par cette technologie sont vastes et le marché de la RFID est en pleine expansion [Harrop2009]. En effet, le coût de fabrication de telles étiquettes est devenu suffisamment

[1] 6LoWPAN : IPv6 over LoW Power Wireless Area Networks. Protocol IP pour les réseaux IEEE802.15.4

attractif pour intéresser les entreprises désirant améliorer leur processus interne ou bien personnaliser des services associés aux produits physiques placés, loués ou vendus auprès de leurs clients. La lecture d'étiquettes (par lots) à distance permet d'automatiser certains processus comme la chaîne d'approvisionnement et d'assurer la traçabilité de marchandises. Parallèlement, le marché des téléphones mobiles intelligents (*smartphones*) s'est développé et ces derniers présentent désormais des caractéristiques jusqu'alors propres aux micro-ordinateurs (capacité de stockage de plusieurs giga-octets, système d'exploitation, écrans de plus en plus lisibles et utilisables pour d'autres applications que la téléphonie…). Certains intègrent également des capteurs (caméra, accéléromètre et GPS) et disposent de possibilités de communication accrues grâce à l'intégration de plusieurs antennes (BlueTooth, WiFi, 3G, NFC [Michahelles et al 2007]). En outre, les offres d'objets communicants pour le grand public se multiplient, même s'il s'agit de jouets (Lego Mindstorm, consoles portables Nintendo DSi et Sony PSP) ou de gadgets la plupart du temps. En témoignent les sites marchands comme Mageekstore et ThinkGeek entre-autres qui se sont spécialisés sur ce secteur, ou les constructeurs tels que Violet (NabazTags et Mir:ror).

Les objets communicants peuvent être classés en deux catégories : capteurs et actionneurs, certains pouvant même jouer les deux rôles. Par exemple la brique intelligente du Lego NXT permet de contrôler des servomoteurs et de relever les mesures fournies par divers capteurs (température, luminosité, son, etc). Les capteurs fournissent des données sur le monde physique qui peuvent ensuite être traitées par un système d'information. Les actionneurs permettent à des systèmes d'agir sur le monde physique.

Ci-dessous, le Tableau 1 dresse une liste d'exemples de capteurs, actionneurs et applications exploitant ces objets dans différents domaines. Ce tableau n'offre qu'un aperçu des possibilités de l'intelligence ambiante (nous n'abordons pas les domaines spatial ou militaire par exemple). La majorité de ces exemples sont déjà en application, notamment dans le contexte du bâtiment intelligent (domotique et immotique) [Snoonian2003, Marples2004].

Cependant l'internet des objets soulève plusieurs problèmes [ITU2005] dont l'autonomie des équipements alimentés sur batterie ou l'organisation des nœuds-objets en réseaux. Plusieurs solutions permettant de palier aux problèmes d'autonomie ont été envisagées.

- **Mise en veille** : Certains objets comme les *smartphones* ou les ordinateurs portables sont capable de diminuer leur consommation énergétique adaptant la puissance de rétro-éclairage, des antennes WiFi ou BlueTooth, ou du processeur au niveau de batterie, ou bien en se mettant en veille. C'est aussi le cas de petits capteurs à l'autonomie réduite qui n'ont pas besoin d'être actifs entre l'envoi de deux mesures tels que ceux utilisant le protocole 6loWPAN. Les brèves phases de réveil sont programmées à intervalles réguliers et suivies de longues phases de sommeil.
- **Diminution de la fréquence de communication** : En déplaçant l'intelligence sur les objets et en leur délégant certains traitements tels que le filtrage de données ou l'historisation des mesures, il est aussi possible de minimiser la fréquence de communications plus gourmandes en énergie. Par exemple, un capteur de température intelligent transmettra une nouvelle mesure seulement lorsque la température aura changé de manière significative ou

lorsqu'un seuil est franchi et non à intervalles réguliers afin d'économiser des communications superflues.

- **Alimentation autonome** : Enfin, des machines plus intelligentes comme les robots-jouets de dernière génération ont la capacité de s'autoalimenter. Soit en allant se recharger sur une borne (Rovio de Wowee, ou Spykee de mecano, Figure 3.a), soit en utilisant une source externe telle que l'énergie solaire à la façon de Wall-E dans le film d'animation du même nom (Figure 3.b) [Pixar2008], ou plus concrètement comme les astromobiles *Rovers* envoyés explorer la planète Mars (Figure 3.c). Robots mis à part, des équipements plus « communs » peuvent également gagner en autonomie grâce à des sources externes d'alimentation. De nombreux parcmètres sont équipés de cellules photovoltaïques, et les Bornes d'Informations Voyageurs (BIV) [2] utilisent l'énergie de l'éclairage public pendant la nuit pour se recharger.

Domaine		Exemples
Environnement		• Capteurs : Sondes sismiques, stations météo (thermomètre, baromètre, pluviomètre anémomètre), niveau des eaux, GPS ... • Applications : Météorologie, prévention des risques (sismologie, crues, tsunami), pollution de l'air
Etat d'un Système	**Bâtiment intelligent**	• Capteurs : Thermomètres, détecteurs de fumée (CO/CO2), pression (SAS), cameras de surveillance, consommation électrique, état des portes, luminosité, lecteurs RFID • Actionneurs : Eclairage, chauffage, ventilation, climatisation, portes, volets, caméras de vidéosurveillance, alarmes, ... • Applications : Détection de présence, systèmes anti-incendie, ...
	Véhicules	• Capteurs : jauges de niveau, vitesse, radar de recul, GPS, témoins des ceintures de sécurité, altimètre (avion) • Actionneurs : régulateur de vitesse, pilotage automatique • Applications : parking automatique (avec radar de recul, direction, pilotage)
	Téléphones mobiles	• Capteurs : Accéléromètres, GPS, camera, écran tactile multi-touch
Etat physique d'une personne		• Capteurs : accéléromètre, glucomètre et autres équipements médicaux • Applications : médecine (HAD, MAD, détection de chutes), sport (suivi de performances)

Tableau 1. Exemples d'applications à base de capteurs dans divers domaines

[2] Les BIV sont déployées par la SEMITAG sur leur réseau de tramway et reçoivent l'information par GPRS. Plus de 70 BIV sont en service sur l'agglomération grenobloise.

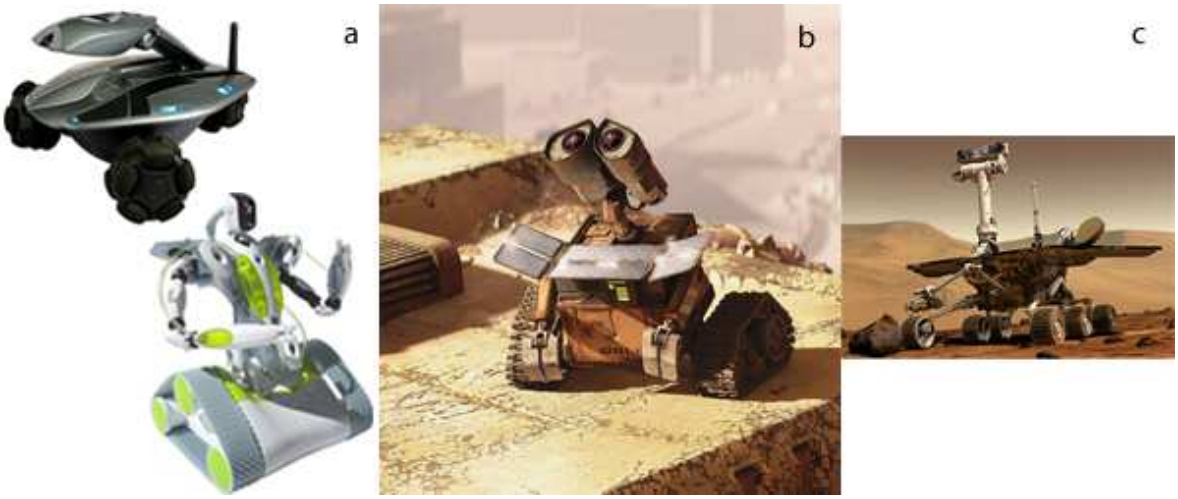

Figure 3. Robots autonomes: Rovio et Spykee (à gauche), Wall-E (au centre) et un Rover martien (à droite)

Pour dialoguer, les objets communicants ont besoin de s'organiser en réseaux via des modes de communication hétérogènes, filaires ou par les airs. Le terme de réseaux capillaires a été introduit pour désigner ces nouvelles classes de réseaux pervasifs. Les réseaux d'objets communicants couvrent en général une zone limitée : du BAN (*Body Area Network*) ayant une portée de quelques centimètres au LAN (réseau local) en passant par le PAN (*Personal Area Network*) ayant une portée de quelques mètres (cf. Figure 1). Dans la littérature on retrouve les réseaux d'objets communicants regroupés sous plusieurs appellations. Le terme WSN (*Wireless Sensor Netwotks*) s'applique à tout réseau de capteurs sans-fil tandis que les SANETs (*Sensor-Actuator NETworks*) concernent plus généralement tout réseau de capteurs et d'actionneurs [Girolami et al 2008], les MANETs (*Mobile Ad-hoc NETwork*) désignent les réseaux d'équipements mobiles et les VANETs (*Vehicular Ad-hoc NETwork*) sont une forme de MANETs spécifiques à l'interaction entre véhicules et équipements routiers.

Concernant le problème de l'organisation des internets des choses en réseaux hétérogènes et fortement dynamiques, la principale caractéristique qui est souhaitée est d'offrir aux réseaux pervasifs un équivalent du *Plug'n'Play* : connecter et fonctionner.

De plus pour construire des applications au dessus de capteurs ou d'actionneurs possédant des capacités de communication, il est préférable que ces objets possèdent une intelligence, c'est-à-dire une capacité de calcul. Un système centralisé communicant avec des équipements primaires (simple transmission de commandes ou émission de signaux électriques) sera plus coûteux en calculs et en communications. En portant une intelligence et une capacité de calcul « au plus près », donc sur les objets, il est possible de filtrer les données et de ne les transmettre que lorsque c'est nécessaire. Ceci permet de réduire la fréquence des communications, comme expliqué plus haut.

Lorsque l'intelligence ne peut-être portée sur les objets, une brique intelligente de type Box ou miniPC durci peut se charger de collecter et de filtrer les données issues de capteurs moins sophistiqués. Si cette brique agit comme une passerelle, elle peut relayer les informations à des systèmes de plus haut niveau. Dans ce type de configuration, une approche consiste en la représentation des fonctionnalités d'objets communicants par des services. Ainsi le service expose ce que peut fournir un équipement : des mesures ou

données pour les capteurs (envoi de températures) ou des fonctionnalités propres aux actionneurs (réglage de la luminosité, orientation d'une caméra). On parle alors de *plate-forme à services*. L'application de l'approche orientée service à l'informatique ubiquitaire est détaillée dans le second chapitre traitant des plates-formes dynamiques à services.

Toutefois l'Internet des Choses par sa nature hétérogène, fortement dynamique et parfois critique pose plusieurs problèmes. La problématique exposée en section I.2 présente plusieurs verrous que nous avons essayé de lever dans cette thèse.

I.1.2 Informatique à la demande

Les dernières années ont été fortement marquées par l'expansion d'Internet. Selon une étude réalisée par l'OECD (Organization for Economic Co-operation and Development) [OECD2009], il y aurait environ 300 millions d'abonnés à l'Internet haut débit dans le monde, soit près d'un foyer sur cinq (pour 19 millions en France). De plus en plus d'activités se font désormais en ligne : procédures administratives telles que la déclaration d'impôts, banque en ligne, e-commerce ou encore le travail nomade.

En effet, parallèlement à l'essor d'Internet, une nouvelle tendance voit le déclin de l'installation d'applications sur les terminaux des utilisateurs et de l'achat de logiciels, au profit de l'utilisation d'applications web riches utilisables à distance, et le plus souvent de manière forfaitaire. L'avantage pour un particulier est la disponibilité de ses applications quelque soit l'ordinateur utilisé pour y accéder. Ce qui est d'autant plus intéressant si l'on considère la multiplication et le polymorphisme des ordinateurs actuels contribuant à l'intelligence ambiante décrite dans la section précédente (ordinateurs portables, netbooks, smartphones, consoles de jeu, ...). Les applications *GoogleApps* de Google ou le lecteur de musique en ligne *Deezer* en sont des exemples.

Ces applications sont déployées sur des infrastructures non contrôlées par l'utilisateur telles que des serveurs virtuels, des grilles de calcul ou autres systèmes d'information reliés à l'internet, et désignés par le terme d'informatique en nuage (*cloud computing*).

I.1.2.1 Informatique en nuage

La notion de nuage fait référence à Internet où la complexité de l'infrastructure est masquée et représentée schématiquement par un nuage. Le but de ce nouveau paradigme est de proposer une informatique dématérialisée, redimensionnable et accessible à la demande [Cloud2010]. Ainsi, en utilisant des ressources externes potentiellement extensibles, une entreprise peut adapter l'utilisation de ces ressources à sa charge et par conséquent faire face aux pics d'activité et à l'inverse consommer moins de ressources durant les périodes peu actives. Avec une infrastructure classique propriétaire, un fournisseur d'applications n'est pas assuré d'avoir un parc informatique permettant de pouvoir tenir la charge et ce dernier risque de tourner la plupart du temps en sous-régime ou en surrégime.

L'externalisation du calcul et du stockage permet donc de mettre en œuvre une informatique à la demande s'adaptant dynamiquement à la charge d'utilisation grâce à la virtualisation de ressources.

Dans le cadre de cette informatique à la demande émergeante, les applications ou ressources sont accessibles et utilisables via des services. Le modèle économique SaaS (*Software as a Service*) s'est développé avec l'avènement des services web. Dans ce modèle l'usager n'est pas propriétaire (par achat de licences) des applications qui sont consommées et payées à la demande. De manière plus générale on parle aussi de XaaS pour désigner la mise à disposition de plates-formes ou d'infrastructures en tant que services (respectivement PaaS et IaaS).

Il faut toutefois émettre des réserves. Si ce modèle est intéressant sur le court terme les utilisateurs n'ont aucun contrôle sur les applications utilisées, et ce mode de fonctionnement peut poser des problèmes de confidentialité et de sécurité. Le succès du cloud computing n'est donc pas encore établi et sa notoriété est peut-être partiellement due à un effet de mode.

I.1.2.2 Serveurs d'applications modulaires orientés service

Sur les mêmes principes d'informatique à la demande, les serveurs d'application proposent eux aussi des offres « à la carte ». Considérant la complexité et la taille croissante des logiciels, les bonnes pratiques dictées par les règles de génie logiciel poussent les systèmes informatiques à la modularité.

Un système modulaire présente plusieurs avantages par rapport à un système monolithique. Il est en effet beaucoup plus facile d'appréhender la complexité d'un système brique par brique plutôt que dans son ensemble. Ce qui améliore, entre autres, sa maintenabilité. De plus les modules composants de tels logiciels peuvent idéalement être réutilisés dans d'autres applications. Découper un logiciel de taille conséquente en briques plus petites c'est aussi gagner en souplesse puisqu'il est possible que certains composants s'avèrent dispensables dans certains cas, ce qui permet de répondre à des besoins différents. Il est alors possible de proposer des solutions « à la carte » ou « à la demande ». Ainsi un serveur d'application comme JOnAS peut être embarqué dans un environnement contraint ou déployé sur une grille de calcul [Desertot2007].

L'émergence de l'approche orientée service mettant en avant le découplage des composants a peu à peu introduit le service comme brique de base de tels systèmes.

Ainsi le socle même du serveur (ou du cœur d'une architecture à plugins comme l'IDE Eclipse) est une composition de services dits « techniques ». Ces services correspondent par exemple à la couche de persistance, de sécurité, ou aux moteurs transactionnels. Ils se distinguent des services métiers fournis par les applications -spécifiques propres à un domaine- déployées sur le serveur.

Un des principes de la programmation orientée service est l'indépendance des cycles de vie des composants d'une application. Ce qui pose des problèmes concernant la disponibilité de chaque composant qui évolue de manière dynamique. Problème qui ne se pose pas avec une application monolithique où tout est contrôlé. Par exemple, les services techniques proposés sur ce type de serveurs peuvent être ajoutés, mis à jour, ou retirés dynamiquement pendant l'exécution du serveur et de ses applications.

Ce qui nous amène au même problème que celui soulevé par le contexte précédent et de manière plus générale par les architectures dynamiques orientées services, et que nous exposons dans la section suivante.

I.1.3 Informatique verte

"La contrainte stimule l'imagination"

Georges Pérec

De manière transversale aux deux contextes précédents, l'informatique fait également face à des problèmes d'ordre écologique. « Green IT » est le terme utilisé ces dernières années pour désigner l'informatique « verte » [Murugesan2008]. Au-delà de son effet de mode, le Green IT s'annonce comme une des préoccupations majeures de la prochaine décennie. La gestion intelligente de l'énergie s'inscrit dans la lignée des projets de développement durable (e.g., Smart Grid). La Figure 4 représente une configuration de grille gérée façon « GreenIT » à la fois en utilisant des serveurs virtualisés pour adapter la consommation de ressources à la charge, et en adaptant également l'apport en électricité provenant aussi bien d'énergies renouvelables que de *Smart Grid* (réseau de distribution intelligente de l'électricité).

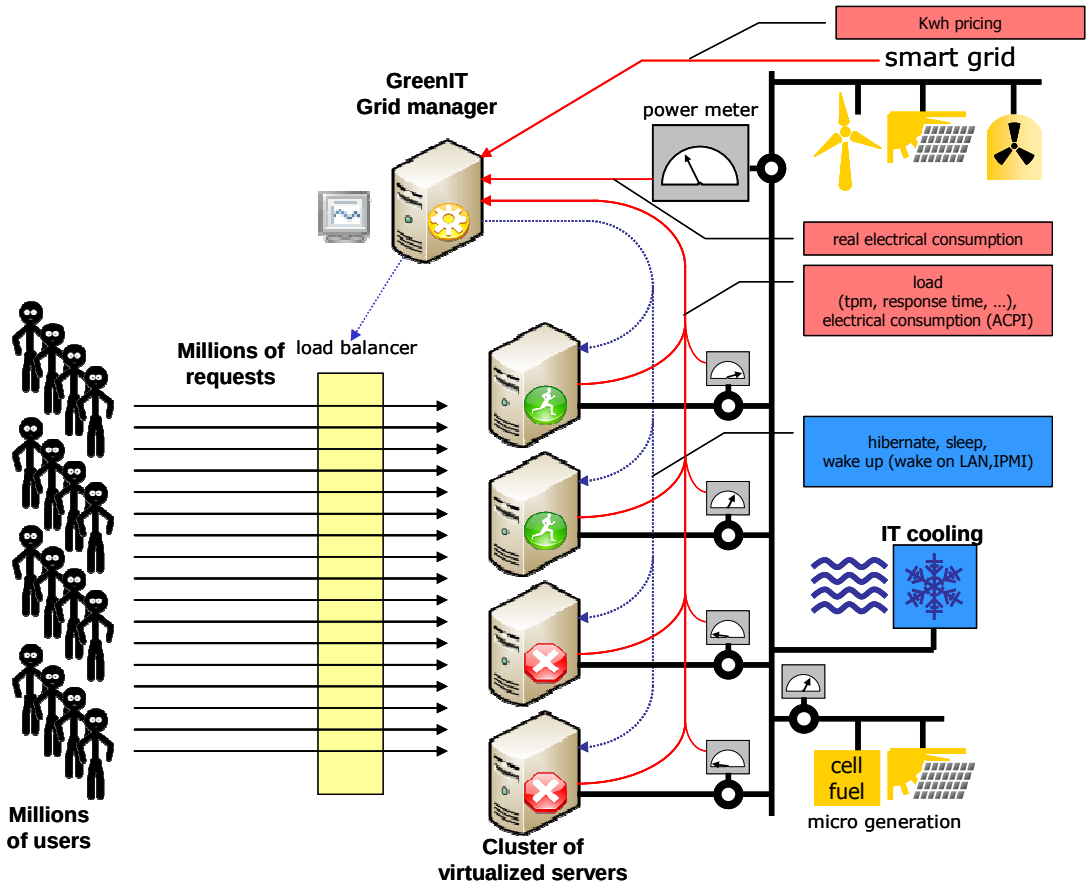

Figure 4. Configuration de grille « verte » adaptable à la demande

La consommation et l'approvisionnement en électricité devient une préoccupation de plus en plus importante pour les administrateurs de centres de données (*data centers*) quelque soit la taille de ces derniers [Fouquet2010]. La consommation des centres se répartit entre l'alimentation des composants des machines et leurs refroidissements. Des études présentent le total de l'électricité consommée en 2007 par les data centers comme étant de 200 milliards de kWh pour une valeur de 29 milliards de $. Cette quantité représente 1 % de l'électricité produite dans le monde quelque soit son origine (nucléaire, charbon, pétrole, solaire, éolien, ...). Elle correspond aussi à 10% du budget des Direction des Systèmes d'Information (DSI). Des centres sont aussi obligés de déménager car un manque d'approvisionnement en électricité existe dans certains pôles industriels où la demande est plus forte que l'offre. D'autre part, ces data centers ont un impact non négligeable sur l'écologie de la planète car ils produisent 2% du CO2 mondial qui sera tôt ou tard taxé. Un autre constat est que ces centres ne sont en moyenne chargés qu'à 20% de leur puissance maximale la plupart du temps, ce qui justifie l'intérêt de serveurs capables de s'adapter à la demande.

Le Green Computing prône une approche globale pour améliorer la performance énergétique des équipements informatiques que ce soit pour les data centers comme pour des équipements Small Office - Home Office (SOHO) [Saito2010]. Des mesures comme DCiE/PUE (pour *Power Usage Effectiveness*), SPECPower, 80+, ... sont désormais utilisées pour calculer et pour publier la performance énergétique des data centers. Le Green Computing prône aussi bien des solutions comme l'optimisation de la climatisation et la récupération d'énergie pour le chauffage de locaux, que la généralisation des clients légers ou bien la virtualisation des serveurs. Une solution consiste aussi à démarrer et arrêter des unités de calcul en fonction de la charge de calcul demandée au data center. Dans ce cas, l'informatique autonomique a son rôle à jouer dans la gestion intelligente des ressources de calcul. Le gestionnaire gère notamment le départ et le déplacement des images virtuelles des serveurs ou des instances de serveurs (par exemple, instances d'un même service JavaEE en mode *fail-over*). Les mesures collectées sur les serveurs sont non seulement des informations de performance (charge de la CPU et des disques, temps de réponse, ...) mais des informations de consommation électrique. Les actions sont le démarrage (WakeOnLAN) et l'hibernation des unités de calcul, la migration de sessions utilisateurs, la migration d'images de serveurs virtuels. Les rapports d'activité comportent désormais des métriques énergétiques et écologiques telles que la consommation électrique détaillée et globale, les coûts instantanés et cumulés en euro, le coût en euro par transaction, en gramme de CO2 produit par transaction, le PUE ... en plus des métriques traditionnelles sur la performance ou la disponibilité.

Par ailleurs l'informatique pervasive peut-être un moyen de mettre en œuvre l'informatique verte. En s'appuyant sur la construction d'applications à base de capteurs, tels que les *Smart Grid*, il est possible de concevoir des systèmes intelligents capables de minimiser la consommation d'énergies polluantes. Ou encore, l'utilisation de capteurs de luminosité peut servir à orienter des panneaux solaires et déterminer si un système doit utiliser cette source d'alimentation ou une autre plus conventionnelle en cas de conditions climatiques défavorables.

I.2 Problématique

L'évolution des technologies de l'information a donné naissance à des applications présentant de nouvelles contraintes et de nouveaux besoins. L'approche orientée service offre un ensemble de mécanismes permettant de satisfaire ces besoins [Bieber2002]. La problématique à laquelle tente de répondre cette thèse concerne essentiellement les préoccupations liées à l'adaptation dynamique des architectures orientées service. Nous faisons l'hypothèse que dorénavant le contrôle sur les différentes parties d'une application n'est pas assuré et que chaque partie est soumise à des conditions de disponibilité qui lui sont propres. C'est pourquoi nous abordons la question des architectures dynamiques en plaçant les interruptions de service au centre de notre réflexion.

I.2.1 Défis des nouvelles applications

Les nouvelles applications émergeantes bâties sur l'internet des choses ou des services ont des besoins bien spécifiques. Entre-autres, hétérogénéité, évolutivité et adaptivité. Ces contraintes sont qualifiées de non-fonctionnelles car elles ne concernent pas directement le code applicatif. Elles entrent en considération lors de la phase de développement d'une application, et accroissent ainsi fortement la complexité et le coût des tâches de développement et de maintenance qui sont généralement conduites par des experts d'un domaine particulier (automobile, immotique, société de télésurveillance, ...).

L'informatique ubiquitaire est **hétérogène** par nature. Elle s'appuie sur des infrastructures non-homogènes, tant au niveau matériel que logiciel avec des protocoles pouvant varier d'un objet communicant à l'autre.

Ces nouvelles applications doivent également pouvoir **évoluer** pour supporter l'introduction de nouvelles fonctionnalités, de nouveau matériel sans devoir pour autant les redévelopper dans leur intégralité. La modularité associée à un couplage faible entre chaque module composant une application permet de répondre efficacement à ce besoin.

Enfin ces nouveaux types d'applications doivent pouvoir être reconfigurées dynamiquement, à la demande, lors de leur exécution; d'où un besoin d'**adaptivité** et de réactivité. Aussi bien dans un contexte d'intelligence ambiante où les applications doivent gérer le dynamisme ambiant et s'adapter à de nouvelles configurations, du fait de la mobilité et de la multitude des objets communicants, que dans un contexte d'informatique à la demande où l'architecture des applications et infrastructures doit pouvoir évoluer dynamiquement en fonction des besoins des utilisateurs.

L'approche à service permet dans un premier temps de répondre aux besoins tels que l'hétérogénéité et l'évolutivité grâce à un couplage faible entre consommateurs et fournisseurs de services [Escoffier2008]. Les architectures dynamiques orientées service présentées dans le chapitre II ont en outre été conçues pour répondre aux besoins d'adaptivité. Ce type de plate-forme à services permet aux applications de réagir aux arrivées et départs de services. Ainsi, les fournisseurs de service peuvent être substitués à l'exécution s'ils fournissent le même service.

Néanmoins pour que ce dynamisme ne soit pas géré directement par le code métier des applications, une nouvelle approche a fait son apparition ces dernières années. L'ingénierie logicielle à base de composants appliquée aux architectures dynamiques orientées service [Cervantes2004b] permet de mettre en œuvre le principe de séparation des préoccupations. De ce fait, des modèles à composants orientés service permettent de piloter les liaisons de service d'un composant à travers l'usage de politiques.

Ces politiques se limitent principalement à une approche statique qui correspond à la simple liaison tardive, par opposition à une approche dynamique. Dans le premier cas, une fois la liaison créée entre un consommateur et un fournisseur de service, celle-ci est fixe et le consommateur ne peut se lier à un autre fournisseur. Si la liaison est rompue le consommateur est alors arrêté. Dans le second cas, le consommateur peut s'adapter aux arrivées et départs de services. La liaison est détruite et recréée selon la disponibilité dynamique d'autres fournisseurs du service requis.

Enfin, lorsque ces nouvelles technologies sont appliquées à un domaine critique (santé, énergie, transport, finances, militaire) la mise en œuvre d'applications est soumise à des contraintes supplémentaires. Notamment vis-à-vis de la stabilité des architectures logicielles et de la disponibilité des composants de ces architectures afin que le fonctionnement de telles applications ne soit pas perturbé. Une approche trop dynamique peut poser problème par rapport à ces contraintes, tandis que l'approche statique ne permet pas de répondre aux besoins d'adaptabilité à l'exécution.

I.2.2 Interruptions de service

Du fait de la nature mobile des objets communicants, le contexte de l'intelligence ambiante est propice aux interruptions de service. La perte de connexion, l'autonomie limitée, les opérations de maintenance sont des exemples de phénomènes responsables de ces interruptions. Toutefois, ces interruptions sont généralement de durée limitée n'étant pas nécessairement causées par des erreurs ou des pannes sérieuses.

De même, les infrastructures mettant en œuvre une informatique à la demande (ou en nuage), qui idéalement devraient pouvoir répondre à toute demande, doivent désormais se plier à des contraintes d'ordre écologique. Les grilles de calculs et centres de données sont en effet très gourmands en énergie. L'informatique verte implique la mise en veille de ressources ou bien leur ajustement. Pour s'adapter à la demande, les serveurs entrent et sortent d'un état d'hibernation plus ou moins profond, ce qui implique une disponibilité non-immédiate du fait des temps de démarrage nécessaires. Ces reconfigurations d'infrastructures matérielles et/ou logicielles conduisent alors à l'interruption des services qu'elles fournissent.

Ces services que nous qualifierons par la suite d' « intermittents » ont la particularité d'avoir une disponibilité fluctuante, mais néanmoins normale et souvent prévisible. C'est-à-dire que les interruptions qui les caractérisent font partie intégrante de leur cycle de vie.

I.2.2.1 Garanties sur la disponibilité de service

L'intermittence de service d'une part et la criticité de certaines applications d'autre part induisent tous les deux un besoin de garanties sur la disponibilité des services utilisés. En

effet, dans l'approche à service les utilisateurs d'un service n'ont théoriquement aucun contrôle sur le cycle de vie de ce service.

Un consommateur ne connaît pas a priori le fournisseur du service qu'il utilise, ce dernier étant masqué derrière le concept de service, et surtout ne contrôle pas son cycle de vie et donc sa disponibilité qui peut dépendre de conditions externes. Or il arrive que l'application consommatrice du service ait besoin d'être sûre que quand elle aura besoin de l'utiliser il ne lui fera pas défaut. Ce qui est par exemple le cas des applications critiques du domaine de la santé.

En outre, le faible couplage des architectures orientées service ne présente pas que des avantages. Il est parfois nécessaire d'identifier le responsable légal en cas d'erreur ou d'incident (pouvant être causés par l'interruption d'un service), ce qui n'est pas possible si ce dernier est complètement masqué derrière la notion de service. Dans le contexte des applications inter-organisations impliquant différents acteurs, il est donc primordial que le découplage propre à l'approche à service soit modéré afin que les acteurs puissent être identifiés. C'est d'ailleurs ce découplage jugé trop fort et le peu de contrôle qu'il implique qui freine l'adoption des plateformes à services dynamiques dans des domaines critiques où la disponibilité et la stabilité sont cruciales.

Il existe plusieurs façons de répondre à ce problème d'interruptions de service. Une approche classique consiste à prévenir les pannes et les interruptions [Avizienis2001]. La recherche dans le domaine de la tolérance aux pannes explore cette voie et propose des solutions basées sur la réplication, matérielle ou logicielle. Le but étant d'améliorer la disponibilité d'un système en le répliquant pour anticiper les possibles pannes. Dans le cas de services fournis par des équipements physiques disséminés (qu'il s'agisse de capteurs ou d'actionneurs), une approche basée sur la réplication signifierait la multiplication d'objets communicants fournissant les mêmes fonctionnalités. La réplication systématique peut alors vite devenir couteuse et polluer visuellement l'environnement.

A priori cette solution pourrait être envisagée pour des infrastructures telles que les grappes et grilles de calcul ou des serveurs nécessitant une forte disponibilité, ce qui est historiquement le cas, mais elle ne correspond plus à la philosophie d'informatique à la demande où seules les ressources strictement nécessaires doivent être utilisées. De plus la redondance matérielle est contraire aux principes de l'informatique « verte » n'est pas applicable dans des contextes où elle n'est pas économiquement viable.

Une autre manière d'aborder le problème de la disponibilité de services dynamiques consiste à admettre que l'interruption est un phénomène normal du cycle de vie d'un service et qu'il suffit de contractualiser la disponibilité des services pour couvrir les cas où l'indisponibilité d'un service entrave le bon fonctionnement d'une application. Les accords de niveau de service (abrégés en SLA pour *Service Level Agreement*) représentent un moyen efficace d'obtenir des garanties tout en prévoyant les cas où elles ne seraient pas respectées. Toutefois, bien que le domaine des accords de niveau de service se soit naturellement penché sur les problèmes de disponibilité, la plupart des efforts ont été portés du côté des serveurs d'applications ou d'entreprise qui sont majoritairement hautement disponibles. Or par définition, l'indisponibilité est une caractéristique des services intermittents et les interruptions de service sont beaucoup plus courantes dans les contextes auxquels nous nous intéressons, pour de multiples raisons évoquées précédemment. De plus les accords de niveau de service ont plutôt été conçus pour couvrir l'utilisation de services entre organisations (réseaux, hébergement, qualité de service d'une application pour ses utilisateurs) et doivent donc être adaptés pour pouvoir régir

des interactions entre briques logicielles (c'est-à-dire entre une application consommatrice de service et le fournisseur du service).

I.2.2.2 Continuité de service

Une seconde conséquence des politiques de liaison dynamique concerne la continuité de service. Lorsqu'un service est interrompu, l'exécution du consommateur est arrêtée puisque, se plaçant dans le scénario du pire cas, l'éventualité du retour du fournisseur n'est pas considérée. Cette approche privilégiant la réactivité ne permet pas de conserver le contexte de la liaison le temps d'une interruption. La liaison est détruite et recréée lorsque le service, peu importe son fournisseur, est à nouveau disponible.

Une destruction de liaison implique l'arrêt de l'application, sauf dans les cas où la dépendance de service est optionnelle. Par précaution le consommateur de service est en effet désactivé si un des services qu'il utilise n'est pas ou plus disponible puisqu'il n'est plus considéré comme en état de fonctionner. Or en pratique, un service peut par exemple être utilisé une unique fois par l'application et, dans ce cas, une interruption du service n'est plus gênante s'il a déjà été utilisé, et la désactivation du consommateur est alors inutile.

Cette désactivation entraine une rupture du flot d'exécution et possiblement une perte du contexte d'exécution si ce dernier n'est pas sauvegardé puis restauré à la réactivation, ce qui est rarement le cas. Les interruptions de service se ressentent alors directement sur l'architecture et le cycle de vie des applications consommatrices. De surcroît, il est courant qu'une application fasse appel à des services pour en fournir un ; il s'agit de services composites construits à partir d'autres services. Dans ce cas de figure, l'interruption d'un des services nécessaires à l'application entraînera en cascade l'interruption du service composite.

La désactivation due à une interruption s'avère également handicapante pour des raisons liées à la perte du contexte. Si un service propose une suite logique d'opérations nécessitant par exemple d'exécuter les opérations A et B avant une opération C, l'arrêt et la réactivation de l'application consommatrice lui feront reprendre son activité à l'opération A, ignorant le travail fait en amont. Plus généralement il existe des services qui ont un état interne ou qu'on qualifie de services sessionnels. Pour ces services, il serait intéressant de pouvoir assurer une continuité de service en reprenant l'exécution au retour du fournisseur de service, de façon à rendre transparentes les interruptions des services intermittents.

De plus si deux fournisseurs du même service sont disponibles et que l'un deux est interrompu, une politique de liaison adaptive reconfigurera dynamiquement la liaison de manière à utiliser l'autre fournisseur sans désactiver le consommateur. Pourtant dans certains cas, cette reconfiguration de l'architecture appelée « substitution » n'est pas une décision optimale.

Dans le cas d'une interruption de service éphémère ou de durée limitée n'ayant pas d'incidence sérieuse sur l'application, il est également préférable d'attendre le retour du fournisseur de service et de conserver la liaison plutôt que de la détruire pour en créer une autre. Ce qui est d'autant plus justifié si le fournisseur interrompu est plus intéressant d'un point de vue non-fonctionnel (meilleure qualité de service etc), ou si l'utilisation du service est facturée de manière forfaitaire, auquel cas la substitution n'est économiquement pas intéressante.

I.3 Proposition

Cette thèse s'intéresse donc à un moyen d'optimiser la gestion des liaisons de service pour fournir une base à la conception d'applications à la fois adaptables, réactives, mais faisant néanmoins preuve d'une certaine stabilité architecturale lorsque les services sont intermittents; et ceci tout en permettant aux applications d'avoir un certain niveau de garantie sur la disponibilité des services utilisés.

Il s'agit de trouver un équilibre entre la réactivité et le dynamisme des approches dites « adaptatives », et la stabilité de l'approche statique. La politique de liaison que nous proposons vise ainsi à assurer une continuité dans l'utilisation des services intermittents, ceci afin de rendre les interruptions de service les plus transparentes possible.

Les approches classiques ont tendance à faire l'amalgame entre interruption et panne, et par conséquent le retour d'un service interrompu n'est pas envisagé. Dans un premier temps notre contribution consiste à définir la notion de service intermittent, c'est-à-dire que nous distinguons une classe de services dont les interruptions font partie de leur cycle de vie normal. Partant de cette hypothèse, selon la durée de ses interruptions il peut-être plus intéressant d'attendre le retour d'un fournisseur service plutôt que de supposer qu'il a définitivement disparu et, soit le remplacer par un autre, soit arrêter l'application.

Le second point de notre contribution vise à traiter les services intermittents de cette manière en introduisant une nouvelle politique de liaison sensible aux interruptions de service. Une fois lié à un fournisseur de service, lorsque ce dernier devient indisponible, le consommateur de service s'engage à attendre un certain temps le retour du fournisseur avant de le considérer comme réellement indisponible. Cette liaison dirigée par les interruptions offre un compromis entre l'approche trop dynamique des politiques de liaison dites « adaptives », et l'approche statique privilégiant la stabilité nécessaire à certaines applications (mission-critiques par exemple).

Dans ce but, nous proposons de contractualiser les liaisons de service par des accords de niveau de service (SLA pour *Service Level Agreements*) portant sur la disponibilité des services, ou plus exactement sur leurs interruptions. En effet, la disponibilité telle qu'elle est traitée habituellement dans les accords de niveau de service vise plutôt des systèmes hautement disponibles et il nous faut donc raffiner son expression pour l'adapter à nos besoins.

L'utilisation de SLA permet d'une part d'offrir des garanties à une application sur la disponibilité des services qu'elle utilise, et dans le cas où le niveau de service par rapport à la disponibilité ne serait pas respecté, de définir le comportement à adopter. D'autre part les contraintes de disponibilité fixées par un accord entre le consommateur et le fournisseur d'un service, appelées objectifs de niveau de service (SLO pour *Service Level Objectives* en anglais), fournissent les informations nécessaires à la mise en œuvre de notre politique de liaison. Elle se base en effet sur les seuils de tolérance aux temps d'interruptions pour décider de la conservation ou de la destruction d'une liaison lorsqu'un service est interrompu.

Notre proposition couvre toutes les étapes du cycle de vie des accords de niveau de service, depuis la négociation accompagnant la phase de sélection du service à la terminaison du contrat, en passant par la supervision des obligations concernant la

disponibilité. En revanche, dans cette thèse nous ne nous intéressons pas à la manière dont un fournisseur de service peut atteindre ses objectifs de niveau de service. Cette préoccupation tient plus du domaine de l'informatique autonomique qui se trouve être un autre domaine du génie logiciel qui est également traité dans l'équipe Adèle.

Notons que cette réflexion sur le conditionnement du cycle de vie des liaisons entre consommateurs et fournisseurs de service par la disponibilité contractualisée sous forme de contraintes temporelles, peut également s'appliquer à la disponibilité des consommateurs de service.

La réalisation des travaux présentés dans cette thèse cible la plate-forme dynamique à services OSGi et un de ses modèles à composants, iPOJO, pour lequel nous proposons une extension. Notre approche a été expérimentée dans les contextes présentés dans cette introduction, à savoir sur des applications autour des objets communicants et de l'intelligence ambiante, mais aussi sur des serveurs d'applications construits sur l'approche à service pour être en mesure de proposer une offre « à la demande ».

I.4 Structure du document

La première partie de cette thèse dresse un état de l'art en deux chapitres sur le thème des services, traitant de l'approche orientée services puis des accords de niveau de service.

Après un préambule sur la programmation orientée services, le Chapitre II présente les principes de base des architectures orientées service (SOA), puis s'intéresse successivement aux architectures dynamiques orientées service et aux approches à composants pour ces architectures. Les plates-formes dynamiques permettent de concevoir des applications réactives exploitant les propriétés de la programmation orientée service telles que le couplage lâche entre consommateurs et fournisseurs. Néanmoins la gestion du dynamisme accroît la complexité de ces applications. D'où l'intérêt d'utiliser les principes de la programmation orientée composants afin de traiter les préoccupations propres à la gestion des services (découverte, sélection, liaison, publication, retrait) de manière transversale. Dans cette dernière section l'accent est mis sur les différentes politiques de liaisons proposées par les modèles à composants orientés service. A travers ce chapitre, diverses technologies du domaine des SOA sont étudiées, parmi lesquelles les services web, OSGi, Service Component Architecture (SCA) et iPOJO.

Le chapitre suivant s'intéresse aux accords de niveau de service (SLA), en commençant par remonter à ses origines. Il expose les principes de base tels que les éléments constitutifs d'un SLA avant de présenter ensuite les activités liées à la gestion des SLAs telles que la négociation ou les techniques de supervision du niveau de service.

Suite à cette présentation des principes théoriques, l'état de l'art passe en revue divers formalismes de représentation des accords de niveau de service ainsi que des canevas et intergiciels conçus pour la supervision et l'administration de SLAs. On y retrouve des standards tels que WSLA ou WS-Agreement.

La deuxième partie, composée de trois chapitres, décrit notre proposition et l'approche que nous avons suivie pour répondre à la problématique de gestion des liaisons face aux interruptions de service. Le premier chapitre de cette partie (Chapitre IV) est un chapitre de transition puisqu'il présente tout d'abord une étude des interruptions de service à travers un bref état de l'art, avant de s'intéresser aux services intermittents. La section sur l'état de l'art a pour but de montrer les limites des modèles à composants orientés service existants et de leurs politiques de liaison en matière de gestion des interruptions de service ; ce qui permet d'introduire par la suite le concept de service intermittent.

Le Chapitre V concernant notre proposition explique les principes de la politique de liaison tolérante aux interruptions, et pourquoi, dans le contexte des services intermittents elle est préférable à une approche fortement dynamique. Il décrit également comment cette politique peut s'appuyer sur les principes des accords de niveau de service, et comment nous envisageons de représenter l'intermittence de service à travers les accords de niveau de service.

Le troisième et dernier chapitre de cette partie porte sur la mise en œuvre des concepts présentés dans le Chapitre V. C'est-à-dire qu'il y est décrit l'architecture logicielle retenue pour la gestion des accords de niveau de service et l'application de notre politique de liaison.

La troisième partie de cette thèse porte sur l'expérimentation et la validation de notre approche. Elle détaille les choix techniques de conception de notre solution, comment nous l'avons expérimentée, et analyse les résultats obtenus avant de conclure. Le Chapitre VII explique notamment comment ont été motivés les choix d'OSGi et d'iPOJO pour l'implémentation de notre politique et du gestionnaire d'accords de niveau de service.

Le Chapitre VIII présente trois scénarios différents dans lesquels nous avons expérimenté notre approche. Nous avons testé notre politique basée sur les SLAs sur les services techniques du serveur d'application JOnAS sur lequel est exécutée une application d'e-commerce. Notre approche a également été validée dans le cadre du projet Aspire sur l'internet des choses. L'application ciblée s'intéresse à la chaîne d'approvisionnement à température maîtrisée. Le dernier scénario porte sur une passerelle domotique sur laquelle est déployée une application médicale de Maintien à Domicile (MAD).

Ce document se clôt sur des perspectives de travaux futurs qui pourraient donner suite à cette thèse. Il s'achève ensuite par une conclusion, le Chapitre X présentant la bibliographie des œuvres citées dans ce document.

Première partie : Etat de l’art

Chapitre II Plates-formes dynamiques à services

“ Nous apprécions les services que quelqu'un nous rend d'après la valeur qu'il y attache, non d'après celle qu'ils ont pour nous. ”

Friedrich Nietzsche, extrait d' *Humain, trop humain*.

Ce chapitre dresse un état de l’art sur l’approche à services, également connue sous l’acronyme SOC pour *Service-Oriented Computing* et présente ses principes, ainsi que les pratiques du génie logiciel qui se sont construites autour de cette approche : les architectures orientées service, les plates-formes dynamiques à services et les modèles à composants orientés service.

Un service est une prestation rendue par un prestataire ou fournisseur de service à un client qui est le consommateur bénéficiaire du service. Par définition un service a pour vocation de répondre à un besoin du consommateur. Ce dernier est donc l’instigateur de la relation consommateur-fournisseur, il ne subit pas un service mais en bénéficie car comme le dit l’adage bien connu « le client est roi ».

Dans la suite de cette thèse nous utiliserons la notation présentée sur la Figure 5 pour représenter le consommateur et le fournisseur d’un service.

La première section de ce chapitre est un préambule sur l’approche à services, à la suite duquel une seconde section décrit les architectures orientées service (SOA), les principes qui les soutiennent et quelques technologies suivant ces principes (principalement les services Web et les ESB). Ensuite la troisième section introduit la notion de dynamisme avec les plates-formes dynamiques à services, ou comment le SOA peut-être étendu pour servir de base à la création d’applications dynamiques. Plusieurs technologies y sont présentées. Enfin la quatrième partie de ce chapitre est consacrée à l’étude des composants orientés service, une approche récente tirant partie des avantages des approches à service et à composants. Nous y expliquons comment le dynamisme peut-être géré au niveau des composants, et y présentons quelques modèles à composants orientés service. Ce chapitre s’achève par une synthèse globale sur l’approche à service.

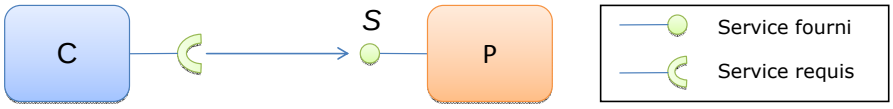

Figure 5. Le consommateur C du service S est lié au prestataire P.

II.1 Préambule sur l'approche à services

La littérature offre un large éventail de définitions du service logiciel, toutes dépendantes d'un point de vue particulier et il n'y a toujours pas de consensus autour d'une définition unique [Jones2005].

> "*Un service est un comportement défini par contrat qui peut être réalisé et fourni par n'importe quel composant afin d'être utilisé par n'importe quel composant, sur la seule base du contrat.* "
>
> Bieber and Carpenter [Bieber2002].

> "*Service : Les moyens par lesquels les besoins d'un consommateur sont liés aux capacités d'un fournisseur.*"
>
> [MacKenzie et al 2006].

> "Un service, au sens SOA, est une fonctionnalité exposée ayant trois propriétés *: (1) le contrat d'interface vers le service est indépendant de la plate-forme, (2) le service localisé et invoqué dynamiquement, (3) le service s'auto-suffit, c'est-à-dire que le service maintient son propre état.*"
>
> Sayed Hashimi [Hashimi2003].

Nous proposons une définition qui se veut plus générale et qui fait intervenir à la fois le point de vue du consommateur, et celui du fournisseur de service :

> "*Un service est une spécification de fonctionnalités pouvant être rendues par des fournisseurs s'y conformant. Le cadre d'utilisation d'un service par des consommateurs est défini par un contrat stipulant en outre un ensemble de propriétés extra-fonctionnelles.*"

Cette définition distingue la spécification fonctionnelle d'un service, pivot permettant à deux entités d'interagir et utilisé comme base commune lors de la conception logicielle de consommateurs et fournisseurs, du contrat qui est propre à chaque fournisseur et qui n'a de sens qu'une fois les entités liées.

II.1.1 Objectifs

Aujourd'hui les entreprises proposant des solutions dans le domaine des technologies de l'information font face à une augmentation importante de la complexité des systèmes informatiques. Il est courant que des infrastructures ou des ressources historiquement indépendantes qui ont été acquises grâce à des fusions ou suite au rachat d'autres entreprises, doivent être gérées par la même entreprise. Ces systèmes devraient pouvoir être intégrés aisément, et interconnectés sans générer un problème de multiplication des interfaces dû à des liaisons point à point [Channabasavaiah2003]. Ce type d'entreprise a acquis au fil du temps un large éventail d'applications patrimoniales (*legacy software*)

aussi bien que des applications développées indépendamment qu'elle souhaiterait pouvoir réutiliser facilement. L'intégration de tels systèmes est une tâche complexe et coûteuse [Lee et al 2003] qui pourtant devient de plus en plus fréquente. L'approche à services (SOC) s'est rapidement imposée durant ces dernières années comme une solution à ce problème d'**hétérogénéité**.

La notion de service offre une granularité « **à gros grain** ». La logique des applications est **encapsulée** et masquée derrière le service qui rend cette logique accessible à travers des opérations dites « **métier** » (liées à un domaine particulier) de plus haut niveau, exposées par le fournisseur de service aux consommateurs de ce service. En outre l'approche à services permet de découpler les systèmes qui deviennent des consommateurs et fournisseurs de service s'articulant autour de l'interface d'échange qu'est le service. Ce **couplage faible** lié au masquage de l'hétérogénéité, en plus de faciliter l'intégration et l'interconnexion de systèmes, offre de nouvelles possibilités et permet de répondre aux besoins de nombreux autres domaines, tels que les applications émergentes décrites dans l'introduction de cette thèse. Construire des applications participant à une intelligence ambiante implique l'intégration d'objets communicants hétérogènes. De plus ce genre d'applications présente des besoins importants en ce qui concerne l'évolutivité et la flexibilité. Les objets de l'internet des choses sont en constante évolution, que ce soit d'un point de vue matériel -le remplacement d'un objet physique- ou logiciel -la mise à jour du logiciel pilotant ou utilisant l'objet-. C'est pour cela qu'un système ciblant de telles infrastructures doit être suffisamment flexible pour supporter l'évolution. Il en va de même pour les applications de taille importante comme les serveurs d'applications qui, grâce à l'approche à services peuvent devenir modulaires et proposer des offres « *à la carte* » [Desertot2006].

A la différence des systèmes monolithiques robustes mais peu flexibles et difficilement maintenables ou évolutifs, la modularité [Parnas1975, Fabry1976] permet de gagner en flexibilité tout en gérant efficacement les problèmes liés à l'évolution des logiciels. Un système monolithique, c'est-à-dire d'un seul bloc, ne peut évoluer qu'en un tout tandis que chaque module composant une application orientée services peut évoluer de manière indépendante. Du fait du couplage faible entre ces modules orientés service, les applications construites suivant l'approche à services peuvent ainsi évoluer brique par brique.

La granularité « à gros grain » améliore significativement la **maintenabilité** et par conséquent diminue les coûts de maintenance des applications construites selon l'approche à services. L'approche à services s'est également révélée être un moyen efficace pour réduire les coûts de développements et accélérer la production d'applications et ainsi le retour sur investissement grâce aux principes de réutilisation et de composition. Dans la lignée de l'approche à composants, l'approche à services favorise la réutilisation de briques logicielles et prône la construction d'applications par composition de briques existantes.

II.1.2 Des objets aux services

L'approche à services peut être vue comme le successeur et le prolongement des approches à objets et composants, comme le montre la Figure 6 inspirée de l'analyse de Racoon [Racoon 1997]. Cette analyse a montré que les tendances du génie logiciel suivaient une évolution en forme de vagues successives. Chaque vague est représentative d'une hausse de l'intérêt pour une approche particulière avant que cette dernière devienne

mature et laisse la place à la prochaine vague. Dans cette interprétation [Donsez2006], l'approche à services s'est développée avec l'essor du modèle B2B (*Business-to-business*) et des EAI (*Intégration des applications d'entreprise*) avant que ses concepts ne se voient appliqués aux interactions M2M.

Les approches à objets et composants précédant l'approche orientée service prônent la réutilisation de briques logicielles par le principe d'encapsulation. Les systèmes devenant de plus en plus complexes et coûteux à développer il est préférable de réutiliser des composants existants dits « sur l'étagère » (COTS pour *Component/Commercial Off-The-Shelf* en anglais).

La réutilisabilité de briques logicielles déjà développées constitue un enjeu majeur du génie logiciel. Il y a plus d'une dizaine d'années, l'approche objet [Gamma et al 1995] donnait un premier élément de réponse à cette problématique. Les objets ont apporté un ensemble de principes de conception[3] parmi lesquels l'encapsulation : l'interface est indépendante et séparée de ses mises en œuvre.

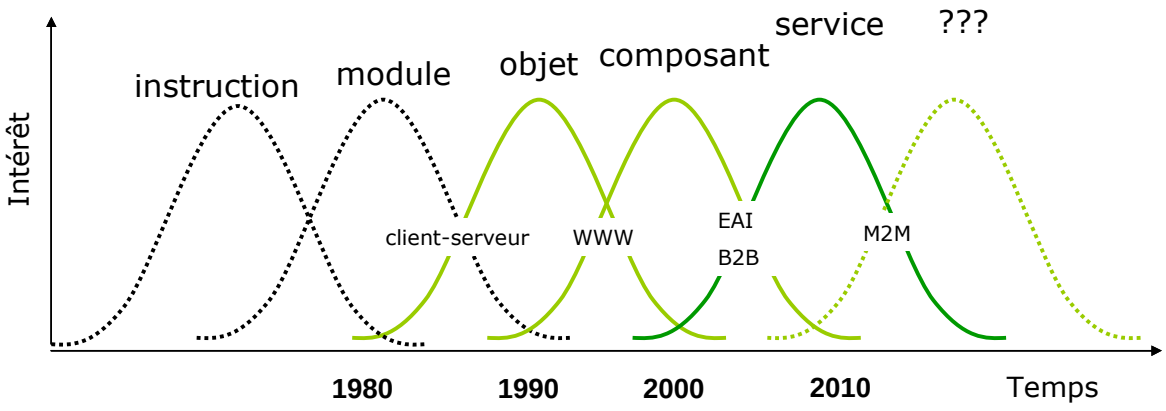

Figure 6. Vagues et tendances du génie logiciel [Donsez2006]

L'approche à composants est ensuite apparue définissant à travers le concept de composant des regroupements cohérents et réutilisables d'objets [Szyperski1998]. Objets et composants partageaient alors des motivations communes : l'encapsulation de la logique et de propriétés accessibles par des interfaces bien spécifiées dans le but de faciliter la réutilisation de logiciels.

Comme l'était le composant pour l'objet, le service est une montée en abstraction par rapport au concept de composant [Dodani2004, FAROS2006]. Un service peut-être rendu par un ou plusieurs objets ou composants dont les fonctionnalités sont encapsulées et accessibles à travers une interface. De la même manière que dans l'approche à composants, un service se présente alors comme une boîte opaque. Les interfaces et les données exhibées par le service sont exprimées en termes métiers et permettent d'accéder au contenu de ces boîtes par des opérations de haut niveau. Ce sont ces interfaces également appelées *spécifications de service* qui sont au centre de l'approche à services et qui permettent la réutilisation de services existants.

[3] Parmi les principes de conception orientés objet, nous pouvons aussi citer l'abstraction (un objet possède un type de données et un comportement bien spécifié), le polymorphisme (capacité de prendre des formes différentes à l'exécution), l'héritage (possibilité d'enrichir un objet avec des extensions), l'identité (capacité de distinguer un objet parmi les autres).

II.1.3 De l'interface au contrat de service

Un service est défini par sa spécification, à laquelle il se conforme. Cette spécification se résume souvent à une interface dite *fonctionnelle*, c'est-à-dire une interface exposant les fonctionnalités rendues par le service. Néanmoins, même si la pratique pourrait le laisser penser, un service (tout comme un composant), ne se définit pas du seul point de vue des fonctionnalités. Un service peut spécifier en plus de ses propriétés fonctionnelles des propriétés *extra-fonctionnelles* qui ne concernent pas directement ses fonctionnalités. C'est le cas par exemple de la sécurité ou de la qualité de service : elles n'affectent pas directement les fonctionnalités du service mais font tout de même partie de la définition du service. Ce sont ces propriétés extra-fonctionnelles qui permettent, entre-autres, à un consommateur de différencier deux prestataires de services similaires d'un point de vue fonctionnel. Prenons l'exemple du tirage de photographies numériques en ligne. Plusieurs prestataires proposent ce service. Toutefois, même si les fonctionnalités (qui dans cet exemple sont le téléchargement de photos, leur traitement, leur impression et la livraison à domicile) sont les mêmes, les offres se différencient autrement. Elles ne seront pas forcément identiques concernant plusieurs aspects extra-fonctionnels : le tarif, la qualité du papier photo, les délais de traitement et d'impression, la possibilité de suivi de la livraison, l'ergonomie du site web, etc.

A partir de ce constat, nous pouvons déduire la règle d'équivalence suivante :

> "*Deux prestataires fournissent un service équivalent si et seulement si leurs propriétés fonctionnelles et extra-fonctionnelles sont équivalentes.*"

Le contrat de service regroupe les propriétés fonctionnelles et extra-fonctionnelles d'un service. Il définit également le service et le cadre de son utilisation. A partir du moment où un consommateur utilise un service, il accepte de manière tacite ou explicite ce contrat. Le Chapitre III relatif aux accords de niveau de service décrit plus en détail le contrat de service.

II.1.4 Du serveur aux services

Telle qu'elle a été popularisée par les Services Web [Vinoski2003], l'approche à services est souvent perçue comme étant dérivée du concept de serveur. Afin de servir ses clients un serveur suit un mode d'interaction de type client-serveur [Mehta2000] consistant en des appels de procédures synchrones selon un mécanisme de type requête-réponse. A cause de cet amalgame, l'approche à services est souvent présentée comme reposant sur ce mode d'interaction client-serveur. Dans cette thèse nous ne souhaitons pas nous limiter à cette vision restrictive du service et garder à l'esprit l'existence d'autres formes de service [Krakowiak2009] qui ne trahissent pas, du moins d'un point de vue lexical, les définitions du service que l'on peut trouver dans la littérature. Ces autres formes reposent sur les motifs d'interactions suivants:

- appel de procédure ou de méthode synchrone avec paramètres et valeur de retour;
- accès à des attributs tels que des structures de données (ce qui peut se rapporter à la forme précédente avec l'utilisation de fonctions jouant le rôle d'accesseurs et de mutateurs);
- appel de procédure asynchrone [Ananda1992];
- source ou puits d'événements;
- flux de données.

Nous nous intéresserons en plus de la première forme (aussi appelée requête-réponse) au cas des deux dernières également connues sous le nom de publication-souscription (*publish-subscribe*) et producteur-consommateur de données. Dans le cas de ces deux connecteurs [Garlan1993, Eskelin1999], le fournisseur de service (respectivement la source d'événements et le producteur de données) a un rôle plus actif que s'il ne faisait que répondre à des requêtes. Le contrat de service ne se résume alors plus à une interface décrivant les opérations et procédures proposées par le fournisseur de service mais sera spécifique au mode d'interaction. Pour les interactions producteur-consommateur de données, le contrat portera sur le type des données échangées tandis que pour les interactions publication-souscription, la partie fonctionnelle du contrat concernera le sujet ou bien le contenu des événements (*topic-based* ou *content-based publish-subscribe* [Eugster2003]). La partie extra-fonctionnelle, quant à elle, dépend moins du type d'interaction. Par exemple un contrat de service engageant un producteur de données ou un rédacteur d'événements pourrait porter sur la fréquence d'émission des données ou des événements.

II.2 Architectures orientées service

L'architecture orientée services [Huhns2005] (SOA pour *Service-Oriented Architecture*) est un style architectural construit sur les paradigmes de l'approche à services. Le terme SOA, en plus de désigner l'architecture basée sur un motif d'interaction particulier à la base des applications orientées service, fait aussi référence aux technologies mettant en œuvre l'approche à services.

II.2.1 Principes de l'architecture orientée service

Le découplage entre consommateurs et fournisseurs est le principe fondamental des SOA. En découplant les éléments d'une application il devient alors possible de les remplacer à tout moment et de composer des applications à partir de services existants. Le contrat de service également appelé spécification de service est le seul lien entre le fournisseur et les consommateurs d'un service. Outre le principe d'encapsulation hérité des approches à objets et à composants, les SOA utilisent un mécanisme de liaison tardive s'appuyant sur un motif d'interactions tripartites pour atteindre ce niveau de découplage.

La liaison proprement dite n'est effectuée qu'au moment de l'exécution, juste avant que le service requis ne soit utilisé. L'utilisateur du service ne possède qu'une dépendance vers l'interface fonctionnelle du service, et c'est à l'exécution que les appels sont redirigés vers le fournisseur de service. Ainsi consommateurs et fournisseurs peuvent évoluer indépendamment tant que la spécification fonctionnelle du service ne change pas. Ce mécanisme de liaison tardive est rendu possible grâce à l'annuaire de services.

Parfois appelé registre, répertoire ou courtier de services, l'annuaire est l'entité centrale des architectures orientées service. C'est son rôle d'intermédiaire entre consommateurs et fournisseurs de service qui permet le découplage et la liaison retardée propre aux SOA. Les schémas ci-dessous décrivent de manière synthétique (Figure 7) et détaillée (Figure 8) le motif d'interaction entre les 3 acteurs des SOAs. Ce motif comporte trois phases :

- la publication;
- la découverte et la sélection;
- la liaison.

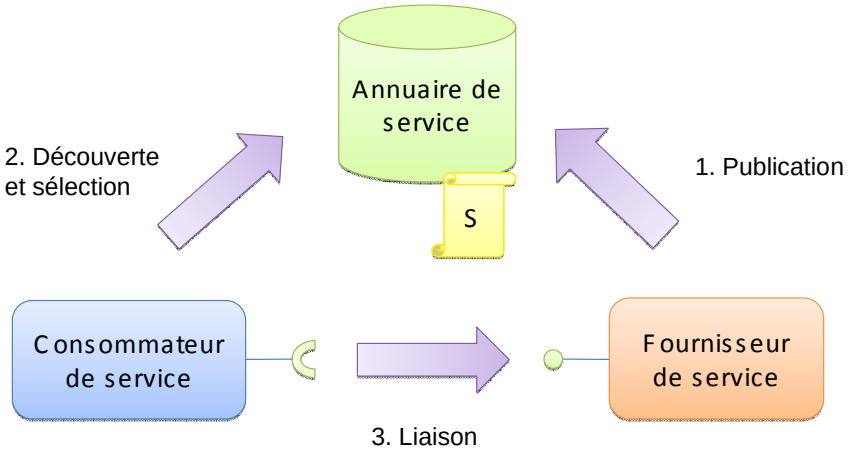

Figure 7. Motif d'interaction d'une architecture orientée service

Avant qu'un service ne puisse être utilisé un fournisseur de ce service doit être localisé par l'annuaire. Pour cela, les fournisseurs de service publient leur service auprès de l'annuaire tandis que les consommateurs interrogent l'annuaire afin de découvrir et sélectionner un ou des prestataires du service recherché. Seulement ensuite le consommateur peut se lier à un fournisseur et utiliser son service. Afin de proposer une sélection plus fine des services, il est parfois possible pour les prestataires de publier en plus de l'interface fonctionnelle du service, ses propriétés extra-fonctionnelles. Ainsi les consommateurs peuvent, par l'utilisation de filtres par exemple (cf. Figure 8), choisir parmi plusieurs prestataires du même service. L'annuaire quant à lui peut être unique et centralisé, distribué entre plusieurs sous-annuaires dispersés sur le réseau, dupliqué pour répondre à des problèmes de tolérance aux pannes, etc. Enfin, un consommateur de service peut aussi jouer le rôle de fournisseur. En utilisant un ou plusieurs services il peut créer de nouvelles fonctionnalités et proposer ainsi de nouveaux services. Il s'agit d'une forme de composition de service.

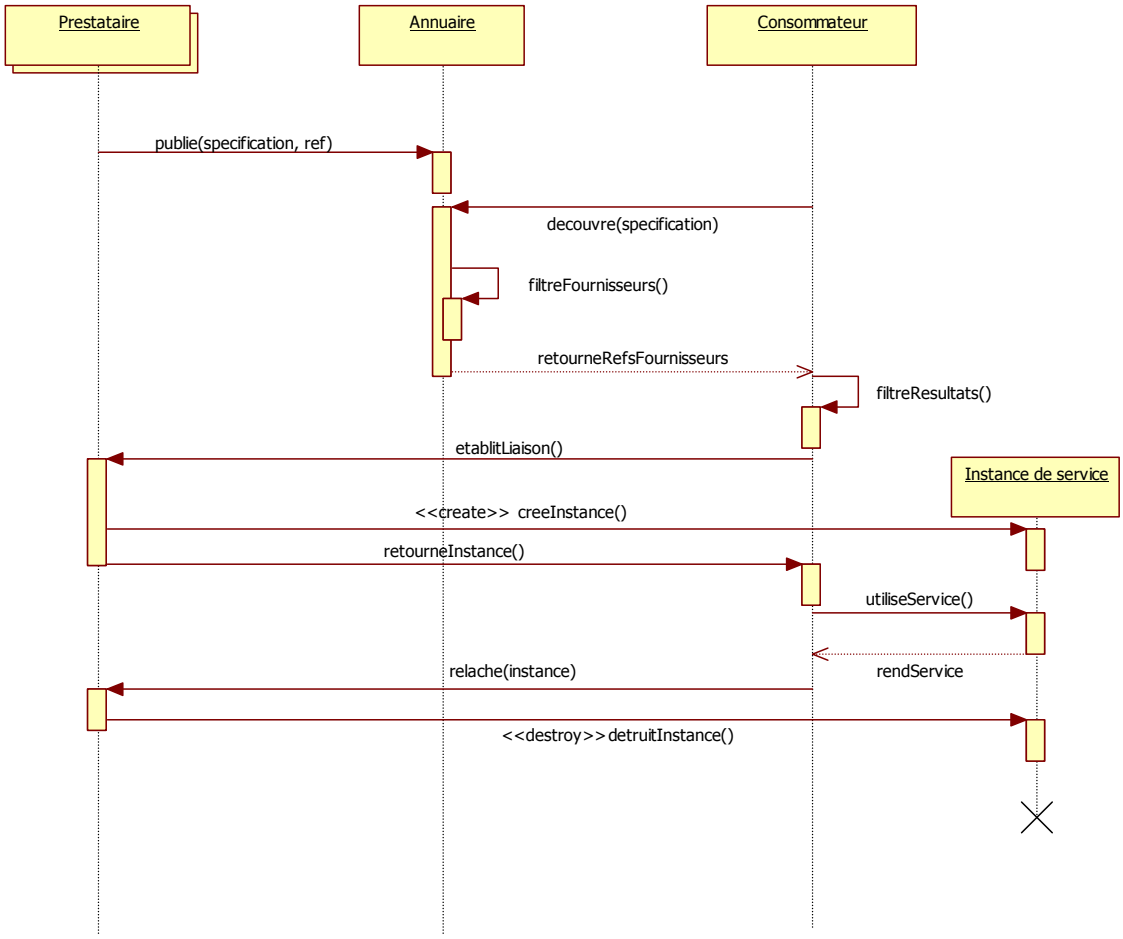

Figure 8. Diagramme de séquence UML du patron SOC

On trouve également dans la littérature certaines règles de conception de SOA [FAROS2006] : l'indépendance aux aspects technologiques, l'absence d'état, la cohésion

et l'idempotence. On notera cependant que, si certaines règles comme l'encapsulation et l'indépendance aux aspects technologiques sont bien maîtrisées, d'autres sont plus contraignantes (l'idempotence ou l'absence d'état) et sont à ce titre moins appliquées.

II.2.2 Architectures orientées service étendues

L'architecture orientée service étendue est une extension à l'architecture orientée service telle qu'elle a été présentée ci-dessus. Cette notion de SOA étendue définie par Papazoglou [Papazoglou et al 2007] s'attaque à des problématiques relatives à l'approche à services. Le SOA ne traite pas, de base, des problématiques de composition et d'administration qui apparaissent dès que l'on commence à réaliser des applications bâties sur l'approche à services.

La Figure 9 présente ces préoccupations sous forme d'une pyramide reposant sur les primitives du SOA de base (la publication, la découverte, la sélection et la liaison) gravitant autour de la spécification de service.

Au-dessus du SOA basique sont proposées des fonctionnalités pour le support de la composition de services dont la coordination de services qui correspond à une composition comportementale : ce sont les procédés métiers qui sont composés. La chorégraphie et l'orchestration présentées sur la Figure 10 sont deux formes de composition comportementale [Peltz2003]. L'orchestration consiste en une série d'activités appelant chacune un service. La logique de composition se situe au niveau de l'orchestrateur. A l'inverse, dans une chorégraphie de services, la logique de composition est embarquée dans le service.

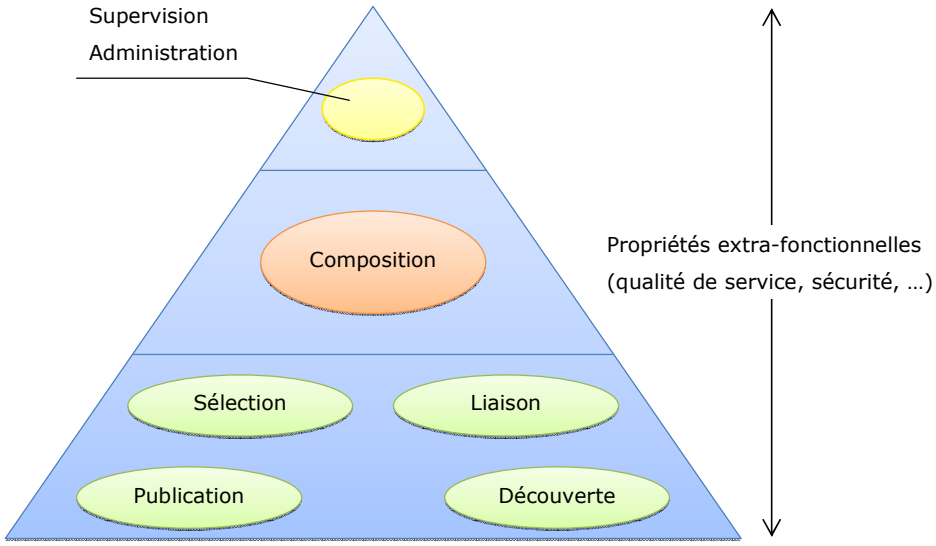

Figure 9. Schéma pyramidal représentant une SOA étendue

L'autre type de composition qui permet de créer des applications à services est la composition structurelle qui permet de construire des services composites. Sans cet étage relatif à la composition de services, il ne pourrait y avoir de notion d'applications à services. Le dernier étage de la pyramide des SOAs étendues concerne ce qui a trait à

l'administration des applications à services (construites par composition de services). Cette couche traite des fonctionnalités de gestion telles que le déploiement, la supervision et l'évolution.

Enfin à tous ces étages, une architecture orientée service étendue doit être capable de gérer des propriétés extra-fonctionnelles telles que la sécurité ou la qualité de service.

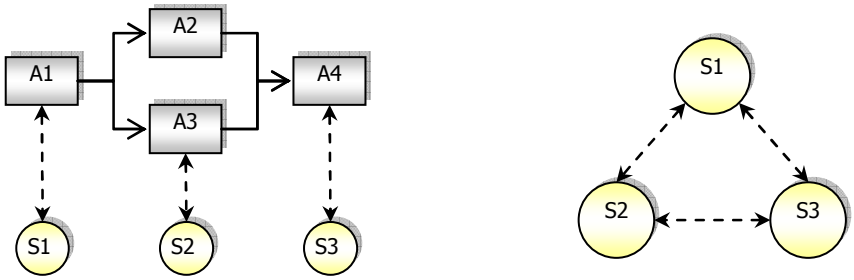

Figure 10. Orchestration (à gauche) et chorégraphie (à droite) [Pedraza2009a]

II.2.3 Services Web

Les Services Web sont l'archétype des architectures orientées services, ce qu'on peut qualifier de standard de fait. C'est l'ensemble des technologies que l'on regroupe sous le terme de *Services Web* qui a contribué au succès et à une adoption grandissante de l'approche à services [Malloy2006]. L'objectif premier des services Web visait à faciliter l'intégration de systèmes hétérogènes. En effet les services Web permettent à des applications hétérogènes s'exécutant au sein de différentes entreprises de pouvoir interopérer à travers des services exposés par les applications. L'utilisation de spécifications construites sur le standard XML (*eXtensible Markup Language*) facilite l'encapsulation et par conséquent l'interopérabilité [Motahari2006].

La Figure 11 représente la pile technologique de base (non étendue) sur laquelle s'appuient les services Web. Un service est spécifié par son interface WSDL. La communication passe par des messages encapsulés dans une enveloppe SOAP, véhiculés par une couche de transport (généralement le protocole HTTP, mais aussi SMTP, IIOP, ...). UDDI est l'annuaire de service permettant la découverte et la sélection de services Web.

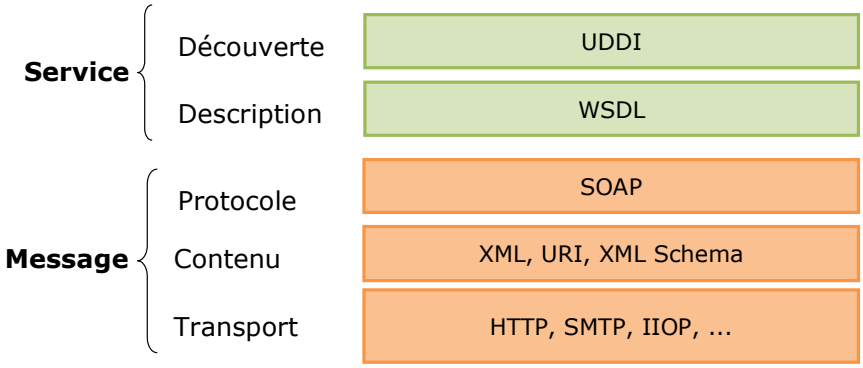

Figure 11. Pile technologique des services web

II.2.3.1 Spécification de service : WSDL et ses extensions

La description des services web est réalisée dans un langage spécifique appelé WSDL (*Web Service Description Language*) [WSDL2001, WSDL2007] recommandée par W3C (*World Wide Web Consortium*) depuis la version 2.0. Le langage WSDL est basé sur une grammaire XML et permet de décrire l'interface du service, les types de données employés dans les messages, le protocole de transport employé dans la communication et même des informations permettant de localiser le service spécifié.

De nombreuses spécifications souvent nommées WS-* étendent WSDL afin de transformer les services Web en architecture orientée services étendue. Ainsi certaines spécifications fournissent des informations complémentaires, comme WS-CDL (*Web Services Choreography Description Language*) qui contient des informations sur l'ordre des opérations à effectuer, ou WSDL-S (*Web Services Semantics*) qui décrit la sémantique du service.

II.2.3.2 Communication : SOAP

La communication avec un service Web utilise le protocole standard SOAP. Ce protocole bâti sur XML permet la transmission de messages entre les services Web et leurs clients. SOAP fournit une enveloppe contenant des informations sur le message lui-même afin de permettre son acheminement et son traitement. SOAP fournit également un modèle de données définissant le format du message, c'est-à-dire les informations à transmettre tels que les paramètres d'entrées et de sorties.

Les services Web reposent sur des interactions de type client-serveur et SOAP est un protocole de RPC (*Remote Procedure Call*), c'est-à-dire qu'il offre un mécanisme de requête-réponse permettant à un objet d'invoquer à distance des méthodes d'un autre objet. La transmission de messages est généralement assurée par le protocole http qui passe plus facilement les pare-feux, mais peut également se faire à l'aide d'autres protocoles tels que SMTP.

II.2.3.3 Découverte : UDDI

L'annuaire de services Web, appelé UDDI (*Universal Description, Discovery and Integration*) supporte l'enregistrement de descriptions de services appelés types de services ainsi que de fournisseurs de services (des entreprises). UDDI fournit des moyens de publier un fournisseur de service, typiquement une entreprise, à travers des pages blanches, jaunes et vertes. Les pages blanches contiennent le nom du fournisseur et sa description, les pages jaunes catégorisent un fournisseur et les pages vertes, plus techniques, décrivent les moyens d'interaction avec un fournisseur: descripteurs WSDL de services fournis ou processus métiers.

Il est courant que l'annuaire UDDI se publie lui-même comme un service Web pouvant être interrogé en utilisant le protocole SOAP.

UDDI est un annuaire distribué dans lequel l'information est répliquée sur plusieurs sites, en conséquence un fournisseur de services ne doit publier sa description de service que vers un seul nœud du réseau de registres. En plus de pouvoir se publier lui-même ainsi que son type de service, un fournisseur de service peut aussi se retirer de l'annuaire.

Toutefois, ceci reste essentiellement théorique (cf. Figure 12). En pratique, les descripteurs WSDL contiennent déjà des informations permettant de localiser le service, ainsi la découverte et la sélection du service se font au développement et non à l'exécution. Le consommateur de service connait la localisation du fournisseur de service et n'utilise que très rarement l'annuaire UDDI. Ce mode d'interaction où consommateurs et fournisseurs sont directement liés ne tire pas profit de la propriété de couplage « lâche » apportée par l'approche à services. Les rares implémentations d'UDDI se heurtent en effet à un problème de performances en tentant de passer à l'échelle du web, et se limitent donc aux réseaux d'entreprise.

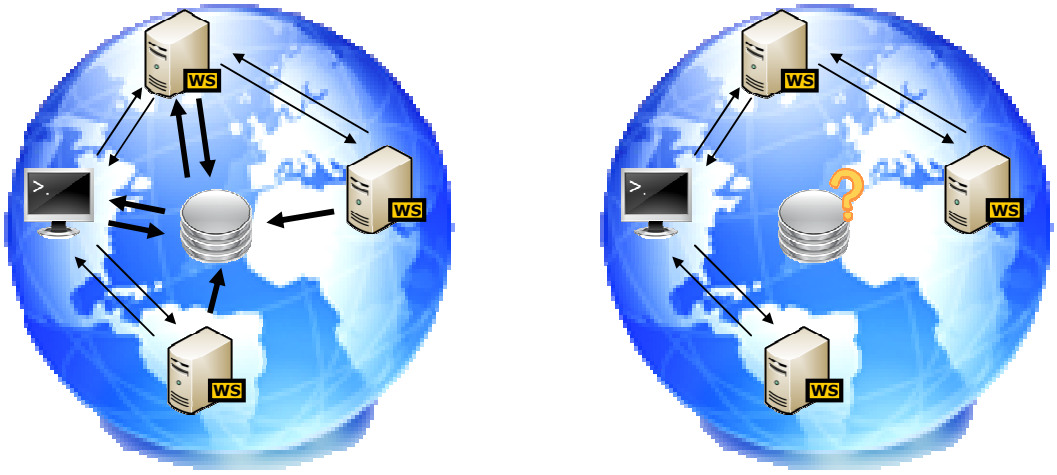

Figure 12. UDDI : de la théorie (à gauche) à la pratique (à droite)

Par conséquent, bien que les services Web soient fondés sur les principes de l'approche à services, paradoxalement, ils ne suivent pas de manière systématique le patron d'interaction propre aux architectures orientées services tel que nous l'avons décrit dans la section précédente.

II.2.3.4 Composition : WS-BPEL, WS-CDL et SCA

Si la composition comportementale a fait l'objet de nombreuses études dans le domaine des services Web, la composition structurelle est beaucoup moins étudiée. Cette dernière présente une vision similaire à la notion de *composite* et de *composition hiérarchique* issue de l'approche à composants. SCA (*Service Component Architecture*), spécifié par de grands groupes industriels comme IBM, Oracle, BEA et SAP, est le représentant principal de ce type de composition [Chappell2007]. SCA définit une approche générale pour créer des composants et un mécanisme pour décrire les interactions entre ces composants, et ce tout en restant agnostique de toute technologie à l'instar des services Web. On peut ainsi qualifier de composite les services Web exposés par des composants SCA. La section II.4.3 sur les composants orientés services fournit plus de détails sur SCA.

Du côté de la composition comportementale ou « par procédés », les services Web proposent deux spécifications : WS-BPEL (*Business Process Execution Language*, anciennement BPEL4WS) [Jordan2007] pour l'orchestration et WS-CDL pour la chorégraphie [Qiu et al 2007].

WS-BPEL repose sur une approche par *workflow*, c'est-à-dire par graphes de processus métiers. BPEL est un langage de programmation basé sur XML destiné à l'exécution de processus métiers. Il inclut les concepts d'activité de base et d'activité structurée. Une activité de base consiste en une interaction avec un service Web. Une activité structurée représente l'orchestration d'un ensemble d'activités de base organisées en un flot de processus (cf. Figure 10 plus haut). L'activité structurée peut inclure des boucles ainsi que des ramifications, et permet de définir des variables pouvant contenir des valeurs de retour de services Web et pouvant ensuite être passées en paramètres dans l'appel d'un autre service Web. WS-BPEL fournit en plus des mécanismes permettant de gérer les transactions et ainsi supporter l'indisponibilité d'un service Web au moment de son invocation.

WS-CDL est un langage standard, également basé sur XML, qui décrit sous forme de chorégraphie les collaborations entre services Web. En définissant d'un point de vue général leur comportement observable, c'est-à-dire l'ordre à suivre dans l'échange de messages, WS-CDL permet aux services Web de collaborer pour atteindre un but commun.

Enfin, toujours dans une perspective de composition orientée procédés, il est également possible de combiner orchestration et chorégraphie dans une approche hybride telle que FOCAS (*Framework for Orchestration, Composition and Aggregation of Services*) [Pedraza2009b].

II.2.4 Services Web RESTful

REST (*REpresentational State Transfer*) est un concept inventé par Roy Fielding en 2000 [Fielding2000]. REST, qui a été popularisé par les interfaces Web 2.0, désigne un style architectural pour la conception d'applications Web suivant plusieurs principes :

- l'URI (*Uniform Resource Identifier*) est l'élément permettant d'accéder à une ressource ;
- toutes les opérations nécessaires sont fournies par HTTP (GET, POST, PUT et DELETE);
- chaque opération est sans état;
- les liens vers d'autres ressources reposent sur les standards hypermédias HTML, XML, JSON, …

L'approche REST peut-être utilisée pour simplifier la mise en œuvre des architectures orientées services basées sur les services Web, on parle alors de services Web *RESTful* [Richardson2007]. Cette approche offre une alternative « légère » à SOAP, WSDL et UDDI : les ressources sont exposées à travers des URI et accessibles par les primitives du protocole HTTP. De plus, contrairement à SOAP, les requêtes ne sont pas encapsulées dans une enveloppe, ce qui réduit le coût de traitement, et elles ne sont pas limitées au XML. Elles peuvent contenir des réponses au format JSON (*JavaScript Object Notation*) ou même des objets Java sérialisés.

En fait REST est plus orienté *données* qu'*opérations*, à la différence des services Web. D'ailleurs les quatre méthodes HTTP suivent le paradigme CRUD [Kilov1990] des systèmes de gestion des bases de données (Create, Read, Update et Delete correspondent respectivement à Put, Get, Post et Delete). De ce fait, l'approche RESTful est également utilisé dans le cadre de réseaux de capteurs (WSN pour *Wireless Sensor Networks*) avec le protocole CoAP (Constrained Application Protocol) conçu pour les environnements d'exécution contraints [Shelby2010].

II.2.5 Enterprise Service Bus et Event-driven SOA

Les *Enterprise Service Bus* (ESB) [Chapell2004, Schmidt et al 2005] ciblent initialement les réseaux d'entreprises (intra et inter entreprises). En effet, le but premier d'un ESB était de faciliter l'intégration d'applications et services de manière agile et dynamique dans des environnements distribués hétérogènes. A l'instar des services Web utilisés également pour l'intégration de logiciels patrimoniaux hétérogènes, les ESBs sont parfois considérés comme la nouvelle génération d'intégration d'applications d'entreprise (EAI).

Les ESBs sont une forme de SOA dirigée par les événements (*Event-driven SOA* parfois qualifié de SOA 2.0) [TIBCO2007]. Il s'agit en fait de bus de messages utilisant les technologies des services Web (SOAP, WSDL, ...). Ces bus servent d'intergiciel orienté messages pour la mise en œuvre d'appels de procédures asynchrones.

Toutefois, du point de vue du développeur, l'asynchrone est plus complexe à appréhender puisqu'il requiert entre autres la gestion des synchronisations. De plus l'utilisation d'événements n'est pas toujours optimale. Selon le contexte, des appels directs de service peuvent se révéler plus appropriés car plus performants et nécessitant moins d'indirections. En fait tout dépend du domaine d'application. Par exemple dans le cas d'une application RFID, le choix d'une solution à base d'événements se fera naturellement car la technologie RFID est événementielle par nature. Au contraire dans un serveur d'application modulaire, les communications entre composants auront plutôt tendance à passer par des appels synchrones que par un bus à événements.

Comme dans toutes les architectures logicielles dirigées par les événements, les applications construites au-dessus d'un ESB écoutent et réagissent aux événements. Ce type d'architecture pose alors des problèmes inhérents à la nature des événements. Par exemple le bus-médiateur peut rapidement se retrouver surchargé suite à des phénomènes de tempêtes d'événements (*event storm*). Les nuages d'événements (*event clouds*) demandent également des techniques de traitement avancées avec l'utilisation de solutions telles que le *Complex Event Processiong* (CEP) [Luckham2002].

II.3 Architectures dynamiques orientées service

Nous avons vu dans l'introduction de ce document que les nouvelles applications telles que celles qui émergent autour de l'intelligence ambiante ont des besoins importants concernant le dynamisme. Lorsque l'on parle de dynamisme ou d'applications dynamiques, nous faisons référence à *la capacité des applications à **s'adapter** (ou se **reconfigurer**) durant leur exécution* [Ketfi2002]. Le mécanisme de liaison tardive introduit par les SOA implique un certain niveau de dynamisme en permettant la composition de services et la création d'applications à l'exécution. Toutefois pour supporter des applications fortement dynamiques, le seul mécanisme de liaison tardive ne suffit pas. Une fois que le consommateur est lié au fournisseur de service, la liaison est statique et n'évoluera plus (à moins de rechercher le service avant chaque utilisation, ce qui est loin d'être efficace et n'est donc pas mis en pratique). Alors que les prestataires de service ont des cycles de vie indépendants des entités logicielles utilisant leurs services, ces dernières ne disposent d'aucun moyen pour être tenues informées des changements au niveau de la disponibilité des fournisseurs de services. Ce qui est d'autant plus gênant lorsque ces prestataires ont une disponibilité dynamique [Cervantes2004a] susceptible d'évoluer régulièrement au cours du temps.

II.3.1 Le dynamisme dans les architectures orientées service

Si le couplage faible inhérent à l'approche à services apporte de la flexibilité aux architectures orientées service, le consommateur de service n'a en principe aucun contrôle sur le cycle de vie du fournisseur auquel il est lié et n'a donc pas de garanties sur sa disponibilité. En allant plus loin, nous pouvons établir qu'un consommateur de service ne peut savoir si le service qu'il utilise sera disponible au moment où il l'utilisera. C'est pourquoi les architectures orientées service telles que présentées dans la section précédente ne permettent pas de construire des applications orientées service dynamiques.

Pour qu'une architecture orientée service puisse servir de base à la construction d'applications dynamiques, les seules fonctions de publication, découverte, sélection et liaison ne suffisent pas. Néanmoins il est possible d'étendre le motif d'interactions des SOA de base pour rendre ces architectures dynamiques avec les primitives suivantes:

- le retrait;
- la notification.

Plutôt que de devoir interroger constamment l'annuaire pour savoir si le service auquel le consommateur est lié est toujours disponible, il est plus logique que ce soit l'annuaire qui tienne le consommateur informé de la disponibilité des fournisseurs auxquels il est, ou souhaite être, lié. Il y a entre ces deux modes de pensée la même différence qu'entre les approches de type *push* et *pull* [Eugster2003].

Dans une SOA classique le consommateur de service est supposé chercher activement (approche pull) un prestataire via un annuaire passif qui se limite à enregistrer les publications de service et à répondre aux requêtes des consommateurs. En ajoutant une fonction autorisant un fournisseur à se désenregistrer ainsi qu'un mécanisme de notification des consommateurs à chaque publication ou retrait de service, l'annuaire se voit attribuer un rôle plus actif et à l'inverse la découverte de service peut se faire passivement du côté du consommateur. Le consommateur en signalant son intérêt à l'annuaire réalise en quelque sorte un « appel d'offre ».

Ce mécanisme de notification, présenté sur la Figure 13 ci-dessous, permet aux consommateurs de réagir aux changements de disponibilité des fournisseurs et ouvre la voie aux applications adaptables (ou reconfigurables) capables, à l'exécution, d'évoluer grâce à des liaisons dynamiques.

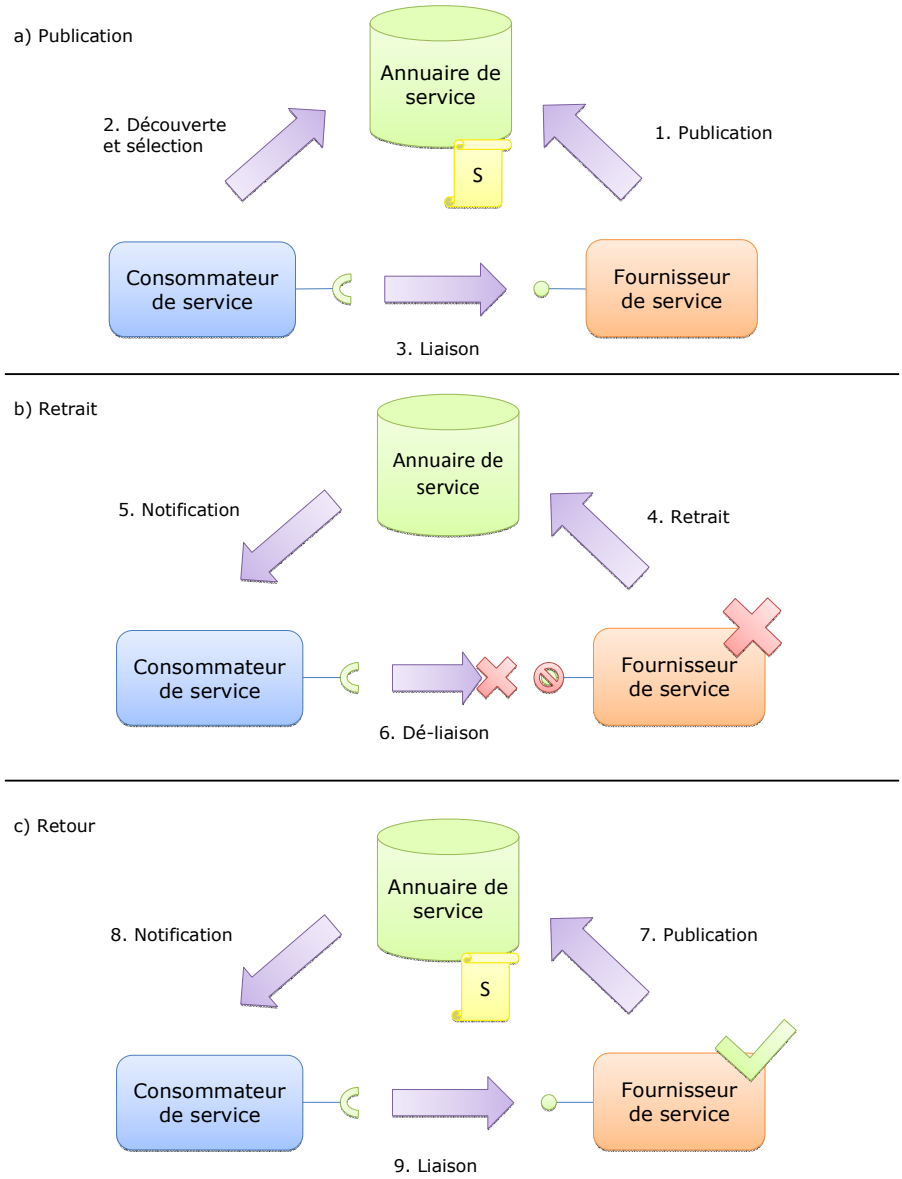

Figure 13. Interactions guidant les architectures dynamiques orientées service

II.3.2 Etude de plates-formes dynamiques à services

La plupart des technologies réalisant des SOA dynamiques sont issues d'un contexte où la disponibilité des services évolue rapidement. Du moins, plus rapidement que les services Web exposés par un serveur d'entreprise qui sont en théorie moins sujets aux interruptions. Ce qui explique en partie pourquoi les services Web ont adopté un modèle relativement statique.

Par exemple, UPnP et DPWS ciblent explicitement les réseaux d'objets communicants sur LAN ou WLAN. De même pour OSGi qui à la base visait les passerelles domotiques, et le protocole Bonjour d'Apple qui n'est pas présenté ici. Cette section se limite à l'étude de Jini, UPnP, DPWS et OSGi. TORBA [Leblanc et al 2002], l'infrastructure orientée courtage de CORBA, n'est pas présentée dans cette section.

II.3.2.1 Jini

Jini est une spécification de plate-forme à services basée sur le langage Java qui fut proposée par Sun en 1999 [Waldo1999] et est aujourd'hui maintenue au sein du projet Apache River. Jini cible les réseaux locaux dynamiques où l'approche à services apporte de la flexibilité : de nouveaux équipements communicants (nœuds du réseau) utilisant Jini peuvent être découverts dynamiquement. La plate-forme Jini permet ainsi la construction d'applications dynamiques suivant l'approche à services en masquant la couche réseau sous-jacente.

D'un point de vue technique, Jini repose sur l'utilisation de Java RMI comme protocole de communication et les spécifications de service se présentent sous la forme d'interfaces Java, donc le contrat de service est purement syntaxique. Les appels de méthodes Java passent par des proxys (appelés *stubs* et *skeletons*). Jini permet en outre le téléchargement de code et la sérialisation.

Registre

Consommateurs et fournisseurs de services doivent trouver un registre avant de commencer à interagir. Un registre, qui correspond en pratique à un service de recherche, peut être connu par une adresse fixe ou bien il peut être découvert. Dans Jini, le registre est lui-même un service appelé *lookup service*. Les différents acteurs (usagers et fournisseurs de service) diffusent une annonce de présence. Cette annonce est écoutée par les services de recherche qui répondent aux annonces et se font connaître. Par la suite, ce sont ces services de recherche qui seront utilisés comme registres. Bien que Jini supporte l'existence de registres multiples, il n'offre pas de mécanismes permettant aux registres de déléguer une demande de service. Au lieu de cela, les fournisseurs de services enregistrent leur service dans plusieurs registres.

Publication

Lorsqu'un fournisseur souhaite publier un service, il envoie au service de recherche un objet instance d'une classe qui implémente l'interface du service. Cet objet permet d'utiliser le service proposé, notamment à travers les méthodes déclarées dans la spécification. Le fournisseur peut aussi publier de manière optionnelle des propriétés appelées *attributs*. Un attribut qui est un objet du sous-type de la classe *Entry*. Les attributs servent ensuite lors de la phase de sélection du service.

Lors de la publication de son service, un fournisseur indique aussi la durée du bail. Le bail (*lease*) est une notion propre à Jini qu'on ne retrouve pas dans les autres SOA. Une fois le bail expiré, si le fournisseur n'a pas renouvelé le bail, l'offre de service est retirée du registre. Le fournisseur peut toutefois annuler lui-même le bail s'il souhaite se désenregistrer. Il est aussi libre de modifier les propriétés qu'il a publié avec le service.

Enfin les services peuvent être organisés par groupes appelés *fédérations* et un registre peut héberger un groupe de services particulier.

Découverte

La découverte de services se fait par interrogation des registres qui auront répondu à l'annonce. Jini ne propose cependant pas de mécanisme flexible du côté du registre pour réaliser un filtrage et une sélection fine des fournisseurs. Les critères servant à déterminer si un service correspond à une demande sont d'une part l'équivalence syntaxique des interfaces du service (l'interface du service fournie et celle demandée doivent coïncider), et d'autre part que les valeurs d'attributs requises soient identiques à celles publiées avec la description du service.

Jini supporte également la découverte passive, ce qui en fait une plate-forme à services dynamique. Un consommateur peut être averti des arrivées ou des départs de fournisseurs de service. Pour cela un consommateur s'enregistre auprès du service de recherche. Il fournit le type du service qu'il souhaite utiliser afin d'être informé de la disponibilité des fournisseurs de ce service. Cette demande de notification est elle aussi régie par un système de baux. Le consommateur doit renouveler son bail afin de continuer à être notifié tout comme il peut le résilier pour indiquer qu'il n'est plus intéressé.

En conclusion, le dynamisme est bien supporté par Jini grâce à la recherche à la fois active et passive, et au principe de bail qui permet d'avoir une garantie relative sur la disponibilité d'un fournisseur de service. Cependant, bien qu'antérieur aux services Web, et novateur pour son époque, Jini n'a pas eu le succès escompté. De plus Jini ne propose pas de modèle de composition, cet aspect étant laissé à la charge du développeur. La gestion « manuelle » du dynamisme et de la composition complexifie la tâche du développeur, qui n'a pas cette préoccupation avec les services Web par exemple. Cette complexité peut expliquer en partie la faible adoption de la plate-forme à services Jini.

II.3.2.2 UPnP et DPWS

UPnP

UPnP (*Universal Plug and Play*) est un ensemble de spécifications établies par l'UPnP Forum [UPnP1999]. UPnP a été créé pour fournir une infrastructure réseau transparente pour le SOHO (*Small Office / Home Office*), c'est-à-dire les réseaux, domestiques ou de petite entreprise, tels que celui représenté sur la Figure 14. Les équipements d'un tel réseau doivent pouvoir être changés, branchés ou débranchés et ainsi joindre et quitter le réseau à volonté, et ceci de manière dynamique. Ce dynamisme oblige ce type de réseau à viser le zéro-administration. Pour cela, les équipements-nœuds d'un réseau UPnP sont auto-configurables et auto-descriptifs. Basé sur IP et autres standards d'Internet (TCP, UDP, HTTP et XML), UPnP est indépendant de la technologie utilisée, que ce soit le mode de communication (Ethernet, WiFi, etc), le système d'exploitation, ou encore le langage de programmation utilisé pour l'implémentation.

Les nœuds sont soit des équipements (*devices* dans le jargon UPnP), soit des points de contrôle (*control point*). Par exemple, un appareil d'éclairage est un équipement tandis qu'une télécommande ou un assistant personnel (PDA) sont des points de contrôle. Les équipements fournissent des services utilisables par les points de contrôle et notifient ces derniers lorsqu'ils changent d'état. Il n'y a donc pas d'entité jouant le rôle d'annuaire, chaque consommateur de service doit maintenir à jour son propre annuaire selon les événements transitant sur le réseau.

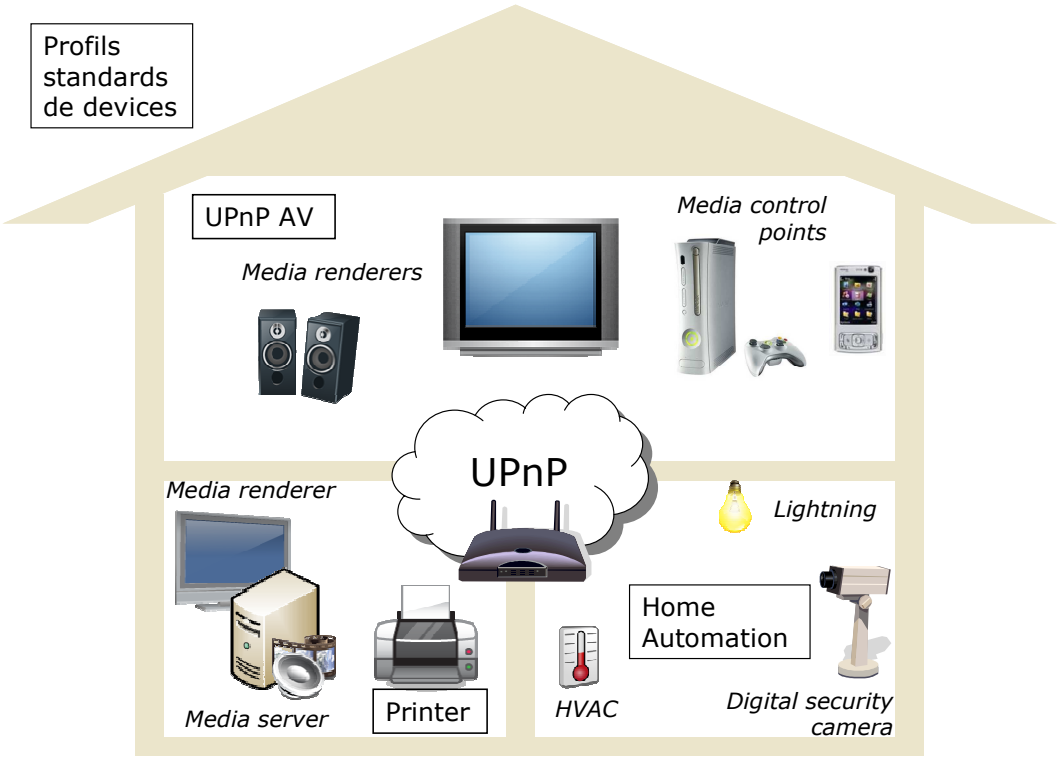

Figure 14. Réseau domestique UPnP

La découverte est basée sur le protocole SSDP (*Simple Service Discovery Protocol* [Goland et al 1999]). Pour découvrir un service, un usager (c'est-à-dire un point de contrôle) recherche les équipements disponibles sur le réseau UPnP. Les messages de découverte sont envoyés selon le protocole HTTP au-dessus d'UDP, en unicast (HTTPU) ou en multicast (HTTPMU) bien que ces protocoles n'aient pas été standardisés. La Figure 15.a décrit la pile de protocole d'UPnP.

L'équipement qui est auto-descriptif transmet alors au point de contrôle le service qu'il propose via son descripteur de service, un fichier écrit dans un langage XML propre à UPnP. Le descripteur fournit l'URL (adresse IP et port) d'accès à l'équipement et un lien vers un descripteur plus précis spécifiant l'interface syntaxique du service (variables d'état, actions disponibles et leurs paramètres). Il peut en outre fournir optionnellement des icones utilisables par les points de contrôle graphiques, des informations spécifiques au fournisseur du dispositif comme le nom du modèle, le numéro de série ou le nom du fournisseur, des URL vers les sites web des fournisseurs, etc. Une fois découvert, le service peut être invoqué, à l'instar des services Web, en utilisant le protocole RPC SOAP.

L'UPnP Forum spécifie également des profils spécifiques à divers domaines tels que ceux représentés sur le schéma ci-dessus. Par exemple le profil UPnP AV[4] [UPnPAV2008] fournit les descriptions d'un *media renderer* (téléviseur UPnP) et d'un *media server* (serveur de fichiers multimédias, comme le logiciel Azureus) tandis que le profil domotique (*Home Automation*) fournit des descripteurs pour l'éclairage, les systèmes de ventilation, de chauffage, de télésurveillance, etc.

En plus de la découverte passive de service grâce aux notifications par SSDP, UPnP propose un mécanisme de notification des changements d'état d'un équipement. Ce mécanisme utilise le protocole General Event Notification Architecture (GENA) [Cohen et al 1999]. Lorsqu'une variable d'état est modifiée, l'équipement émet une notification sur le réseau contenant l'identifiant du device, la variable d'état et la nouvelle valeur. Il s'agit d'ailleurs du seul moyen pour interagir de manière asynchrone avec UPnP.

Bien que le Forum soit toujours actif et que la technologie UPnP continue à équiper divers équipements ciblant les réseaux domestiques (Freebox HD v5, XBox 360, PS3 pour les derniers équipements multimédias), UPnP utilise des protocoles hétérogènes qui ne facilitent pas son intégration avec d'autres technologies.

De plus, du point de vue de l'approche à services, les consommateurs de service ne peuvent découvrir que les équipements, et non les services. Ils doivent donc dans un premier temps découvrir tous les équipements, puis les interroger pour trouver les services requis.

Le Forum UPnP définit également un profil pour la qualité de service [UPnPQoS2008]. Ce profil se concentre surtout sur la gestion du trafic réseau et la qualité de transmission des applications multimédia.

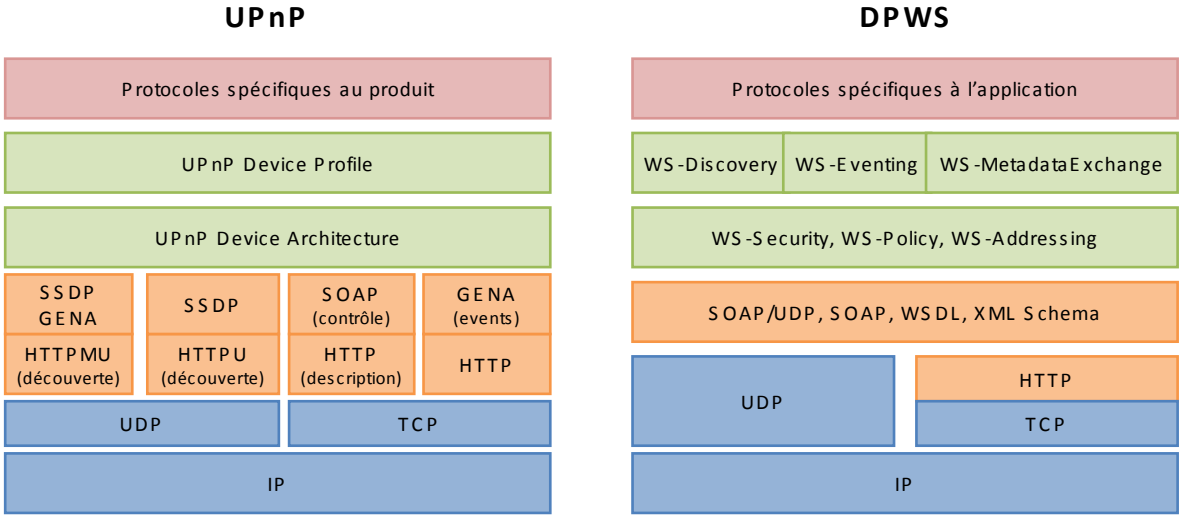

Figure 15 a) et b). Pile des protocoles UPnP (à gauche) et DPWS (à droite)

[4] Les standards UPnP AV (pour Audio Video) sont supervisés par DLNA, un groupement d'industriels du divertissement délivrant des certifications aux équipements qui respectent leurs guides d'interopérabilité.

DPWS

DPWS (*Devices Profile for Web Services*), une spécification ciblant les mêmes réseaux qu'UPnP, est considéré comme le successeur de ce dernier, à la différence près que DPWS s'appuie sur un ensemble homogène de technologies standards issues des services Web ; ce qui lui donne un plus grand potentiel d'intégration. En effet, DPWS suit la même approche qu'UPnP mais est en plus compatible avec les services Web.

DPWS est bâti sur un sous-ensemble de la spécification des services Web qui permet à des équipements d'utiliser ce mécanisme standard de l'industrie pour communiquer avec d'autres équipements, ordinateurs, et services Web sur Internet.

La spécification DPWS définit deux types de services: les services d'hébergement (*hosting services*) et les services hébergés (*hosted services*). Un service d'hébergement est lié à un équipement et joue un rôle primordial dans la phase de découverte, tandis qu'un service hébergé dépend de l'équipement qui l'héberge. En plus de ces services, DPWS intègre un ensemble de services:

- les services de découverte qui permettent de diffuser une annonce et de découvrir d'autres équipements,
- les services d'échange de méta-données qui fournissent un accès dynamique aux services hébergés par un équipement et à ses méta-données,
- et les services de publication / souscription qui permettent à des équipements de s'abonner aux messages produits par un service en particulier.

La Figure 15.b présente la pile de protocoles utilisés pour DPWS [Jammes et al 2005]. En plus des protocoles classiques utilisés dans les services Web que nous ne détaillerons pas ici (WSDL, SOAP, WS-Addressing, WSMetadataExchange, WS-Policy et WS-Security), DPWS ajoute des protocoles pour la découverte et les notifications. WS-Discovery, protocole multicast pour la découverte plug-and-play et ad-hoc d'équipements connectés au réseau, permet de chercher, localiser des équipements puis d'exposer les services fournis par ces équipements. WS-Eventing, quant à lui, définit un protocole pour la prise en charge des événements de type publish-subscribe, ce qui permet à des équipements d'être informés des changements au niveau d'autres devices.

Toutefois, DPWS souffre du même problème qu'UPnP : les consommateurs de services doivent tenir eux-mêmes leur propre annuaire de services. De plus les protocoles de découvertes ne permettent la découverte immédiate des services, ce sont les équipements qui sont découverts en premiers lieu. Soit activement par envoie de messages sur le réseau selon la spécification WS-Discovery, ou de manière passive en écoutant les annonces des équipements joignant le réseau. Enfin, bien que DPWS spécifie précisément les équipements et la façon d'interagir avec eux, de nombreuses capacités sont optionnelles [Zeeb et al 2007]. Cet ensemble d'options risque à terme de poser des problèmes de compatibilité entre des clients et des dispositifs.

WSD (*Web Services for Devices*) [WSDAPI2008] est l'implémentation par Microsoft de la spécification DPWS qui est incluse dans le système d'exploitation Windows Vista, et qui permet à des équipements tels que des imprimantes ou des scanners d'interagir avec Windows sur un réseau IP. Néanmoins Windows Vista fournit un annuaire de services interrogeables par les applications qui s'y exécutent afin de palier à l'absence d'entité centralisant les services du réseau dans DPWS.

II.3.2.3 OSGi

La particularité de la plate-forme OSGi réside dans le fait que contrairement aux autres plates-formes à service, elle est centralisée, c'est-à-dire non distribuée. Les services qui s'y exécutent sont co-localisés au sein d'une même machine virtuelle (Java). La spécification OSGi fut établie par l'OSGi Alliance, un consortium réunissant de grands acteurs industriels tels qu'IBM, Nokia, Motorola et Oracle [OSGi2005]. A l'origine, la spécification OSGi ciblait essentiellement les applications et services techniques pour des passerelles domestiques. Bien que ce domaine constitue encore une préoccupation pour les communautés de chercheurs et d'industriels travaillant sur cette spécification [Royon2007a, Frénot2008], le spectre des cibles d'OSGi a été étendu par la suite. D'une part aux systèmes embarqués véhiculaires, puis plus récemment à la téléphonie mobile [Sprint2008] et à d'autres petits objets dont la mémoire est limitée, et d'autre part aux applications modulaires de grande taille telles que les serveurs d'applications Java Enterprise Edition (OW2 JOnAS [JOnAS2007, Desertot2007] et Sun Glassfish [Glassfish2009]) ou les architectures à plugins comme l'environnement de développement Eclipse [Gruber et al 2005]. La dernière version de la spécification, la 4.2, a été approuvée en Septembre 2009.

OSGi est une surcouche à la machine virtuelle Java, la plate-forme hérite ainsi du modèle de sécurité apportée par Java 2 et des défauts de celui-ci [Parrend2008]. De plus la plate-forme OSGi se compose de 4 couches : l'environnement d'exécution, la couche de modularité, la couche de cycle de vie, et la couche de services. OSGi spécifie à la fois des mécanismes pour le déploiement, et pour la livraison et la consommation de services.

Le *bundle* est l'unité de déploiement. Un bundle est une archive Java (jar) contenant du code et des ressources ainsi qu'un ensemble de méta-données permettant son déploiement et son exécution sur la plate-forme OSGi. Comme le montre la Figure 16, les bundles en plus de pouvoir s'appuyer à la fois sur le canevas OSGi, la machine virtuelle Java et le système d'exploitation, peuvent interagir entre eux par l'intermédiaire de la plate-forme en fournissant et requérant des services et des paquetages. Plate-forme sur laquelle ces unités peuvent être déployées et administrés à l'exécution : un bundle peut-être installé, désinstallé, démarré, arrêté, mis à jour « à chaud », c'est-à-dire sans nécessiter l'interruption totale de la plate-forme. Chaque bundle décrit les paquetages Java qu'il fournit et requiert, ainsi que leurs numéros de version car la plate-forme est capable d'exécuter des versions différentes d'un même paquetage.

En plus de cette vue liée au déploiement, la plate-forme offre une vue « service » puisqu'elle met en œuvre les mécanismes de l'approche à service et fournit une architecture orientée services dynamique. Chaque bundle peut en effet consommer et fournir des services, et ceci de manière dynamique puisqu'OSGi permet l'introduction et le retrait de bundle à l'exécution, et donc les arrivées et départs dynamiques de service.

La plate-forme OSGi définit en effet un annuaire de service appelé *service registry*. Un service OSGi est constitué d'une interface Java et d'un ensemble de propriétés (ensemble de <clé, valeur>) pouvant donner des informations sur le fournisseur de service ou sur le service fourni (qualité de service, …). Un fournisseur peut publier dans l'annuaire le ou les services qu'il fournit, mais aussi retirer une offre et modifier les propriétés liées à la publication d'un service. Le fournisseur enregistre dans l'annuaire une référence vers l'objet réalisant le service (implémentant l'interface de service).

L'annuaire permet à la fois une découverte active et passive des services. Un consommateur peut interroger l'annuaire pour récupérer une liste de fournisseurs

réalisant le service. Cette liste peut être filtrée par un filtre LDAP (*Lightweight Directory Access Protocol*) portant sur les propriétés du service. Ce filtrage permet une sélection plus fine des services qu'avec Jini. Un consommateur peut également enregistrer un écouteur (*listener*) auprès de l'annuaire, en précisant l'interface du service voulu et un filtre LDAP, pour être notifié des arrivées et départs de services l'intéressant. A partir de cette liste des fournisseurs, un consommateur récupère une ou plusieurs références Java des objets servants publiés par les fournisseurs. Il peut ensuite invoquer directement les méthodes spécifiées par le service. Néanmoins, afin de bien gérer le dynamisme, un consommateur de service doit relâcher la référence vers le fournisseur de service lorsque ce dernier se désenregistre. S'il ne le fait pas le bundle du fournisseur ne peut être déchargé de la mémoire et il ne pourra pas être désinstallé correctement ni mis à jour. Ce problème de *stale references* est mal connu de la plupart des développeurs OSGi [Gama2008].

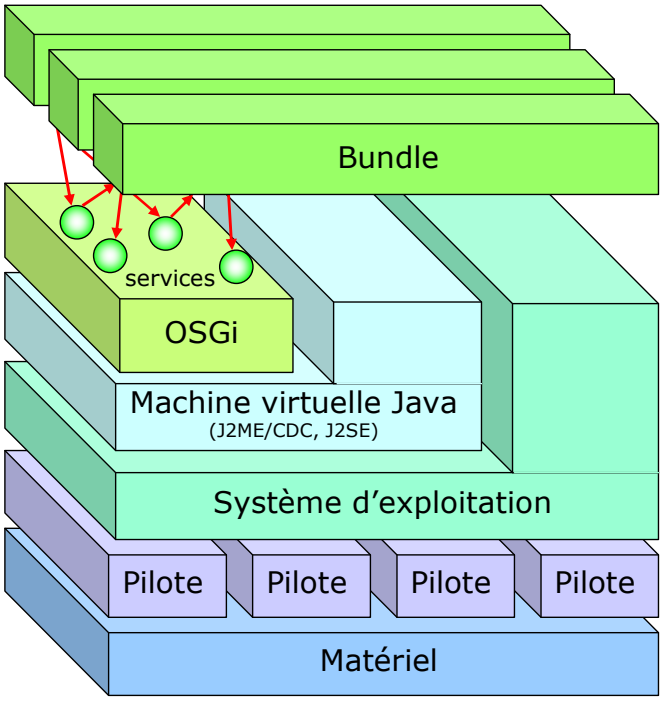

Figure 16. La plate-forme à services OSGi, surcouche modulaire à Java (d'après Peter Kriens)

OSGi ne définit pas de modèle de composition. Toute composition de services est à la charge du développeur ainsi que le dynamisme influençant cette composition, comme dans le cas de Jini. Cependant, afin de faire face à cette complexité, en plus d'un respect des bonnes pratiques de programmation, il existe pour la plate-forme OSGi des modèles à composants décrits dans la section II.4.3 qui déchargent le développeur en externalisant la gestion du dynamisme et la composition de composants orientés service.

Enfin OSGi possède des fonctionnalités de déploiement et d'administration qui expliquent en partie son succès grandissant. Il existe d'ailleurs de nombreux utilisateurs qui se servent d'OSGi comme plate-forme de déploiement et occultent l'approche à services et les fonctionnalités qui permettent pourtant d'établir des architectures orientées service fortement dynamiques.

Eclipse Equinox, Apache Felix, Knopflerfish 2 et Concierge sont les implémentations open-source les plus connues de la spécification OSGi.

II.4 Composants orientés services

"*Never send a human to do a machine's job.*"

The Matrix, Agent Smith.

Les architectures dynamiques orientées service impliquent une gestion fine du dynamisme des services. Or le code gérant les liaisons entre consommateurs et fournisseurs de service se retrouve alors mêlé au code logique de ces entités logicielles, augmentant en conséquence la complexité des applications capables de réagir à la disponibilité dynamique des services.

Les modèles à composants orientés services combinent l'approche à services et l'approche à composants : les services y sont fournis et consommés par des briques logicielles conçues selon l'approche à composants. Cette approche permet, entre-autres, de traiter de manière orthogonale, c'est-à-dire par le conteneur, le dynamisme des architectures orientées services dynamiques.

La définition du composant la plus souvent citée dans la littérature est celle que donne Szyperski [Szyperski1998] :

> *" Un composant logiciel est une unité binaire de composition ayant des interfaces spécifiées de façon contractuelle et possédant uniquement des dépendances de contexte explicites. Un composant peut être déployé de manière indépendante et est sujet à composition par des tierces. "*
>
> Clemens Szyperski

II.4.1 Ingénierie logicielle des composants

Le terme « *composant* » sous-entend l'assemblage de briques logicielles dans le but de créer un logiciel plus gros, ainsi que la réutilisation de composants logiciels composites ou primitifs. Ce qui nous intéresse plus particulièrement ici, est le fait que cette approche s'appuie sur le principe de séparation des préoccupations.

L'ingénierie logicielle à base de composants (CBSE en anglais), plus couramment appelée approche à composants, vise en effet à séparer les fonctionnalités d'un composant de ses propriétés et besoins extra-fonctionnels. Ainsi le code du composant est essentiellement constitué du code métier et il n'est pas parasité par la gestion de ses propriétés extra-fonctionnelles telles que la sécurité, la persistance, les transactions, la qualité de service, ou encore la gestion de son cycle de vie, qui sont traitées de manière orthogonale. La gestion de ces propriétés est à la charge du conteneur dans lequel s'exécute un composant. Le conteneur est parfois appelé membrane en référence à la biologie cellulaire où le noyau contenant l'ADN (i.e., les fonctionnalités du composant) est entouré d'une membrane qui constitue une surface d'échanges.

Un composant se constitue donc d'un contenu, son cœur fonctionnel, et d'un conteneur, son environnement d'exécution. La Figure 17 représente ces deux entités ainsi que la vue interne du composant, le point de vue de son concepteur, et la vue externe, le point de vue de l'utilisateur du composant.

D'un point de vue externe, le composant est une boîte noire qui fournit des fonctionnalités accessibles par des **interfaces fonctionnelles.** Ces fonctionnalités peuvent être rendues par un ensemble d'objets qui peuvent aussi requérir d'autres fonctionnalités issues d'autres composants. Toujours du point de vue externe, les propriétés extra-fonctionnelles du composant peuvent être vues ou configurées via des interfaces dites de **configuration** et le composant peut-être administré si son conteneur fournit une **interface d'administration**. Enfin, le conteneur peut éventuellement utiliser les fonctionnalités de la plate-forme sous-jacente sur laquelle il s'exécute via des **interfaces techniques**. Ces interfaces techniques peuvent fournir au conteneur une couche de persistance, la gestion des transactions ou d'autres propriétés extra-fonctionnelles.

D'un point de vue interne, un composant peut interagir avec son conteneur et inversement grâce à des **interfaces de contrôle**. A travers une interface de contrôle, le conteneur peut par exemple fournir une méthode permettant au composant de récupérer un point d'entrée (parfois appelé *contexte*) vers son environnement d'exécution.

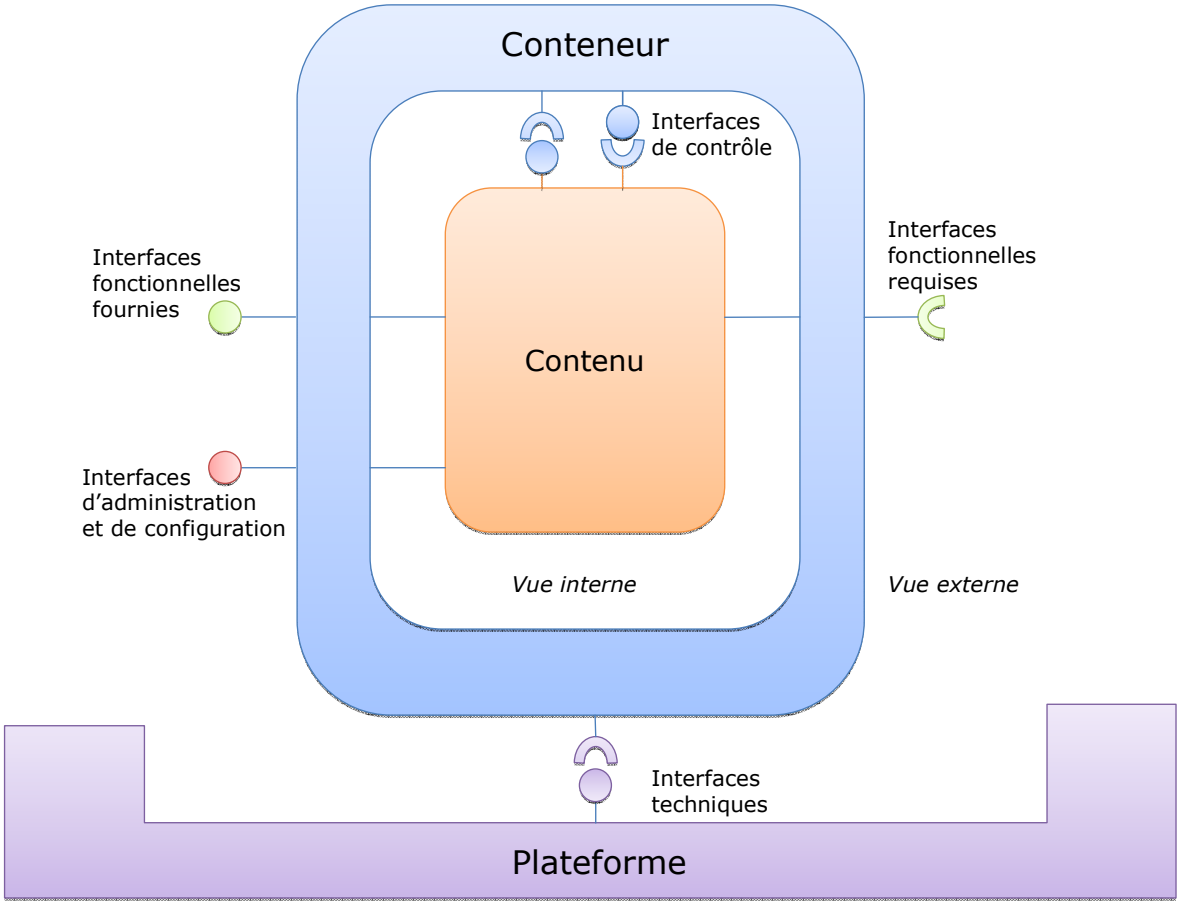

Figure 17. Représentation des vues internes et externes d'un composant

L'approche à composants repose sur l'utilisation des interfaces fonctionnelles pour l'assemblage de composants. La composition structurelle permet de créer à la fois des composants composites (composant construit à partir d'autres composants, on parle de composition verticale ou hiérarchique) et des applications à composants (composition horizontale ou fonctionnelle). Ces compositions et applications sont généralement

décrites dans des langages de description d'architectures (ADL pour *Architecture Description Language*) où sont à la fois décrits composants, connecteurs, et la manière dont ils sont assemblés [Medvidovic2000]. Les ADLs fournissent une vue de l'application; ce qui manque dans l'approche à services.

Un composant peut-être configuré de deux façons différentes : de manière programmatique ou bien de manière déclarative. Dans le premier cas le code du composant utilise explicitement l'interface de programmation du conteneur pour interagir directement avec lui et se configurer. Dans le second cas, le développeur de composant fournit une description déclarative de ses composants qui est interprétée par le conteneur, lequel configure le composant grâce à ces informations. La voie déclarative utilise généralement des méta-données décrites dans une grammaire XML, mais il est de plus en plus courant de décrire la configuration d'un composant à travers des annotations du langage utilisé pour le contenu (attributs C#, annotations Java 5). Les annotations permettent en effet de palier aux problèmes de synchronisation entre le code fonctionnel et les descripteurs non fonctionnels dans les ateliers de développement.

La manière programmatique a l'avantage de permettre la reconfiguration dynamique à l'exécution tandis que la manière déclarative, qui a l'avantage d'alléger le code et de laisser le développeur se focaliser sur la logique fonctionnelle, constitue une approche plus statique (une fois que le composant est déployé, la configuration n'est en principe pas révisée).

Il existe ensuite plusieurs techniques pour que le conteneur puisse appliquer ces configurations au composant. Les plus courantes sont l'interception, le tissage d'aspects, ou encore l'inversion de contrôle.

L'état de l'art propose un vaste panel de modèles à composants. Certains comme Enterprise Java Bean [EJB2010], Corba Component Model [OMG2001] ou Common Component Architecture [CCA2004] sont liés à un domaine ou une plate-forme particulière (respectivement les serveurs d'applications d'entreprise, le temps réel, et le calcul intensif), tandis que d'autres comme Fractal [Bruneton et al 2004] restent intentionnellement génériques. D'autres travaux se sont par exemple intéressés à la séparation du modèle à composants et du langage de définition des aspects non-fonctionnels afin de proposer des modèles à composants extensibles [Duclos2002].

Les deux sections suivantes présentent l'application de l'ingénierie des composants à la mise en œuvre de l'approche à services et à la gestion de la disponibilité dynamique des services, puis étudient quelques modèles à composants orientés service.

II.4.2 Composants orientés services

Les architectures orientées service spécifient essentiellement les mécanismes de publication, découverte et liaison, et ceci le plus souvent à travers le point de vue du consommateur. En particulier dans les services Web où l'accent est mis sur l'interface de service, pas la manière de le fournir. En général rien n'est précisé quant à la manière de rendre le service et la manière de concevoir un fournisseur de service est souvent laissée à l'initiative du développeur. L'approche à composants appliquée aux architectures orientées service est une initiative relativement récente dont ServiceBinder (une évolution

de Beanome) [Cervantes2002], puis plus tard Service Component Architecture (SCA) en sont les pionniers.

Elle consiste dans un premier temps à externaliser le code spécifique lié à la plate-forme à services mais orthogonal à la logique « métier » réalisant le service. Les tâches de publication, découverte et sélection de services sont donc effectuées par le conteneur du composant

- soit de manière programmatique : le composant utilise des interfaces de contrôle fournies par le conteneur pour se lier à un fournisseur de service ou publier un service.
- soit de manière déclarative : le conteneur interprète des méta-données associées au code du composant et se charge alors de résoudre les dépendances de services en cherchant les fournisseurs de service adéquats.

Dans un second temps, le conteneur peut également prendre en charge la gestion du dynamisme des architectures orientées service qualifiées de dynamiques, c'est-à-dire les architectures décrites dans la section II.3 qui fournissent aux consommateurs de service des mécanismes pour se tenir informés des arrivées et départs de services. L'intérêt de l'approche à composants réside ici dans le fait que le conteneur pourra automatiquement publier ou désenregistrer de l'annuaire un service selon le cycle de vie du composant qui le fournit. Lorsque le composant est désactivé ou stoppé, le service est retiré de l'annuaire. Quand il est activé ou démarré, le service est publié.

De même, si un composant consommateur de service confie la gestion de ses dépendances de service au conteneur d'un modèle à composant orienté service, les créations et de destructions de liaisons, lorsqu'un fournisseur publie ou retire son service, se feront de manière automatique.

La section suivante présente plusieurs modèles à composants orientés services. Parmi ceux-ci, certains permettent au dynamisme d'être pris en charge dans la membrane du composant.

II.4.3 Etude de modèles de composants orientés services

Cette section présente une étude de quelques modèles à composants orientés services (SOCM pour *Service-Oriented Component Model*). La plupart des modèles présentés ici cible la plate-forme à services OSGi. Les travaux sur les modèles à composants spécialisés dans la mise en œuvre de SOAs sont rares voire inexistants sur d'autres plates-formes. Seul SCA propose un modèle s'abstrayant d'une plate-forme particulière.

II.4.3.1 Service Binder et Declarative Services

Service Binder est un modèle à composant visant à faciliter la publication et la découverte de services et la liaison à ces services dans la plate-forme OSGi [Cervantes2004b]. La configuration du composant est exprimée dans un fichier de méta-données embarqué dans le bundle OSGi. Elle comprend en plus de la classe du composant, les services fournis, des propriétés de configuration, les services requis et les méthodes de rappel (*callbacks*) à invoquer quand un service requis s'enregistre ou se

désenregistre. En exploitant cette configuration, le conteneur Service Binder se charge de la publication et du retrait des services fournis dans le registre de services d'OSGi, mais il permet surtout de traquer les arrivées et départs de services. La sélection de services ne diffère pas du modèle OSGi, elle se fait sur la spécification et sur un ensemble de propriétés. Ainsi il est possible de déclarer un filtre dans la configuration du composant afin de qu'il ne se lie qu'aux fournisseurs correspondant au filtre. Ensuite, grâce à la déclaration des méthodes de rappel *bind* et *unbind* implémentées pour chaque dépendance par un composant consommateur de service, les références vers les objets de service sont injectées, à l'exécution, dans le composant via ces méthodes. Ce qui permet au composant de s'adapter dynamiquement aux changements de contexte et à la disponibilité dynamique [Cervantes2004a] des services.

Le modèle de composant Declarative Services est apparu dans la 4ème version de la spécification OSGi. Son approche reprend les objectifs et les principes de ServiceBinder, et utilise les mêmes primitives (*bind/unbind*) en se basant aussi sur une description déclarative du composant à travers un fichier de méta-données XML.

Un autre point majeur et commun à Service Binder et Declarative Services est l'assujettissement du cycle de vie du composant qui est géré par le conteneur. Le composant, peut en effet implémenter une interface de contrôle comportant une méthode d'activation et une méthode de désactivation qui sont appelées respectivement quand tous les services requis sont disponibles et quand un des services requis est retiré et devient indisponible. Ainsi le cycle de vie du composant est assujetti à la disponibilité dynamique des services qu'il requiert.

Par conséquent, si un composant implémentant une spécification de service (i.e., une ou plusieurs interfaces Java) déclare fournir ce service dans sa configuration, le conteneur du composant publiera automatiquement le service à l'activation du composant et le retirera du registre à sa désactivation. Le contexte passé aux composants dans des méthodes d'activation et désactivation permet en outre d'utiliser des fonctions simplifiant la recherche de services et donnant accès à des propriétés décrites dans les méta-données du composant.

Finalement Service Binder et Declarative Services se focalisent sur la gestion de la publication et du retrait de services tout en gérant la dynamique des services par création et destruction de liaison au cours du cycle de vie des composants, et ceci de manière déclarative à travers une description de configuration. L'utilisation de ces modèles à composants simplifie la tâche des développeurs d'applications à services sur OSGi, qui ne se soucient plus d'assurer par programmation le suivi des services, la découverte, la liaison ou la publication. Ils peuvent en effet se concentrer sur le code métier des applications dans lequel n'apparaissent que quelques méthodes de contrôle.

D'autres modèles ont étendu Service Binder et Declarative Services.

Extended ServiceBinder [Bottaro2006], offre la possibilité d'importer et d'exporter des services locaux et distants, OSGi ou non-OSGi (Jini, UPnP, ...), et ainsi permettre l'utilisation de services distribués au-dessus de ServiceBinder. Toujours dans l'optique d'aborder la problématique de l'intelligence ambiante, ServiceBinder étendu enrichit la description de composant au niveau des propriétés de service. Du point de vue du fournisseur, cette extension offre une alternative à la description de propriétés de service statiques. Les propriétés peuvent en effet être récupérées dynamiquement via une

méthode appelée par le conteneur avant la publication du service et de ses propriétés. Et du point de vue du consommateur, ServiceBinder étendu permet de spécifier une méthode de tri des fournisseurs (ou fonction d'utilité) afin de ne pas limiter la sélection de service à des critères définis statiquement dans la description du composant. Typiquement si un utilisateur se déplace, ses préférences concernant la localisation du service évoluent en conséquence, et ces besoins dynamiques ne peuvent être exprimés statiquement de manière déclarative.

On retrouve d'ailleurs cette volonté de palier à la description statique des composants dans SOSOC (pour *Service Oriented and Script Oriented Components*) [Donsez2009a], un langage de scripts dynamiques permettant de reconfigurer à l'exécution des composants SCR (pour Service Component Runtime, l'implémentation d'Apache Felix de Declarative Services). L'idée derrière SOSOC est d'utiliser la dynamicité des langages de script pour apporter de la flexibilité aux modèles de composants déclaratifs statiques et leur permettre d'être adaptés et reconfigurés à l'exécution.

II.4.3.2 Dependency Manager

Apache Felix Dependency Manager [Offermans2003] est un autre modèle à composant pour la plate-forme OSGi qui cible spécifiquement la construction de dépendances de service, et ceci de manière programmatique et non déclarative à la différence de SB ou Declarative Services. Pourtant le principe reste le même et les liaisons d'un consommateur de service vers des fournisseurs sont créées et détruites dynamiquement à partir de la configuration du composant et de la disponibilité dynamique des fournisseurs de services. Ce qui offre la même capacité d'adaptation que celle des composants orientés service construits par les modèles déclaratifs pour OSGi décrits dans la section précédente.

Néanmoins l'utilisation d'une interface de programmation apporte plus de flexibilité puisque de nouvelles liaisons peuvent être créées dynamiquement alors que la configuration descriptive d'un composant à travers des méta-données n'évolue plus après la phase de déploiement (si l'on excepte l'utilisation d'un outil externe comme SOSOC). Mais d'un autre côté, le code de gestion des liaisons, bien qu'il reste plus simple que l'utilisation des mécanismes de base d'OSGi, est mêlé au code métier et reste donc une source potentielle d'erreurs en comparaison d'une externalisation de la configuration par les méta-données. D'autant plus que la consultation des méta-données d'un composant permet de se faire rapidement une idée de l'architecture d'une application puisque les services requis et fournis y sont exprimés, ce qui n'est pas le cas lorsque ces informations sont enfouies dans le code source.

II.4.3.3 Apache Felix iPOJO

iPOJO [Escoffier2008] est un sous-projet d'Apache Felix qui reprend les principes de Service Binder et Declarative Services en allant plus loin. Ce modèle à composant s'intéresse non seulement à une gestion plus fine des dépendances de services, mais a aussi pour objectifs de fournir des mécanismes de composition hiérarchique hérités de l'ingénierie des composants, et de pouvoir traiter de manière orthogonale d'autres préoccupations extra-fonctionnelles, et ceci de manière extensible. Et ceci de manière la plus transparente possible pour le développeur qui dans l'idéal ne fait que manipuler des

objets métiers appelés POJO (*Plain Old Java Object*) [Richardson2006], dans lequel les propriétés extra-fonctionnelles sont injectées.

iPOJO introduit la notion de dépendance de service. Lorsqu'un composant iPOJO requiert un service, on dit alors qu'il a une dépendance vers ce service. La dépendance peut être résolue ou non. Tant qu'elle n'est pas résolue, c'est-à-dire tant que le consommateur n'est pas lié à un fournisseur satisfaisant les conditions de liaison exprimées par la dépendance, le composant consommateur n'est pas valide. Comme dans les approches Service Binder et Declarative Services, il est désactivé. De même lorsque toutes ses dépendances de service sont résolues, le composant devient valide et est activé. Cependant nous verrons ci-après que ce n'est pas toujours vrai et que le cycle de vie d'un composant iPOJO peut être affecté par d'autres événements.

Les dépendances de service sont injectées dans le POJO par la machine à injection d'iPOJO qui s'appuie sur la manipulation de bytecode Java et l'inversion de contrôle. Ainsi le développeur de composants iPOJO n'a qu'à déclarer les services utilisés dans des champs de son objet métier et les utiliser normalement. Le conteneur se chargera à l'exécution d'injecter des références vers les bons fournisseurs de service dans ces champs, ou de les passer en paramètres à des méthodes de rappel comme le font Service Binder et Declarative Services. Le conteneur iPOJO s'assure également que ces champs contiennent bien des objets de service lorsque le POJO accède au service.

Un composant iPOJO est constitué en plus du POJO et de sa configuration exprimée sous forme de méta-données dans un formalisme XML ou d'annotations, d'une membrane modulaire composée de briques logicielles appelés *handlers*. La Figure 18 décrit une interaction entre deux composants iPOJO via ce mécanisme de handlers : un consommateur et un fournisseur de service.

Chaque handler est le gestionnaire d'un aspect extra-fonctionnel. La configuration d'un composant iPOJO décrit en fait la configuration de chaque handler composant la membrane du composant. Le canevas iPOJO propose plusieurs handlers dans son noyau dont deux étant dédiés à la gestion dynamique des services : un handler destiné à la livraison de service, et un gestionnaire de dépendances. Le premier a pour seuls buts la publication et le retrait du service et de ses propriétés dans le registre OSGi. Le second permet la découverte passive, la sélection et la liaison aux services. Plus riche que Declarative Services, iPOJO propose différentes options de configuration pour une configuration plus fine des liaisons. Il permet par exemple, quand aucun prestataire n'est disponible, d'injecter une implémentation par défaut du service fournie par le consommateur.

Deux autres handlers du noyau rendent possible la configuration d'une instance de composant par la définition de propriétés, ou donnent le contrôle sur le cycle de vie du composant.

Le cycle de vie d'un composant iPOJO est conditionné par l'état de ses handlers. Tant que tous les handlers utilisés par le composant ne sont pas valides, le composant est désactivé. Il n'est réactivé que lorsque ses handlers sont valides. Par exemple si les dépendances de service d'un composant fournisseur de service ne sont plus satisfaites, le gestionnaire de dépendance devient invalide et le composant sera désactivé, entrainant le retrait automatique du service du registre.

En plus des handlers du noyau, le conteneur iPOJO peut être étendu par des handlers externes. Un exemple est le handler JMX (*Java Management eXtension*) qui crée

dynamiquement des MBeans exposant les méthodes et les attributs spécifiés dans la configuration du composant. Grâce à cette extensibilité construite sur une approche de conteneur modulaire, iPOJO peut potentiellement injecter n'importe quelle propriété extra-fonctionnelle via ce mécanisme de handlers.

Un autre exemple d'handler externe est le *TemporalHandler* qui étend la gestion des liaisons d'iPOJO en permettant de temporiser les liaisons de service. Ceci dans le but de fournir un certain niveau de tolérance aux interruptions.

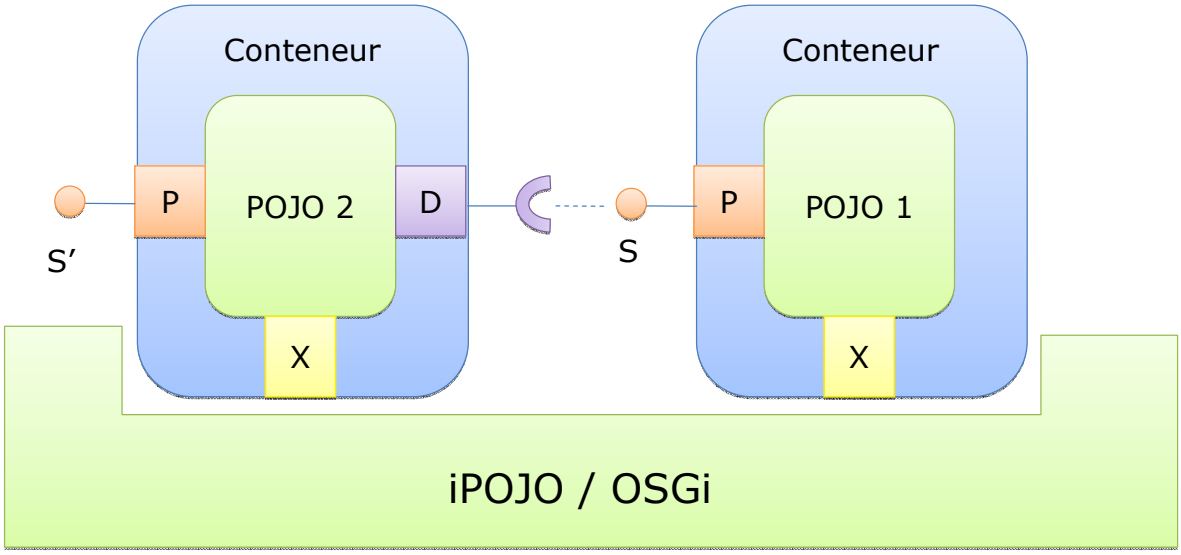

Figure 18. Structure de composants iPOJO et interactions entre consommateur et fournisseur de service

Sur la Figure 18 le POJO 1 publie un service S grâce au handler P (*ProvidedService handler*). Le POJO 2 est à la fois consommateur et fournisseur de service. En utilisant le service S auquel il se lie grâce au handler D (le *Dependency handler*), il est capable de fournir et publier un service S'. Le handler X est un handler externe utilisé par les deux composants.

Enfin, le modèle à composant iPOJO propose des mécanismes de composition hérités de l'ingénierie des composants. La composition dite horizontale ou fonctionnelle s'appuie sur la livraison de service et les dépendances de services : un composant prestataire de service peut fournir un service construit à partir d'autres services fournis par d'autres composants. La composition verticale ou structurelle correspond à la création de composites hiérarchiques dont la structure interne est construite à partir d'autres composants pouvant eux-aussi être des composites. Ces composants internes ne sont donc pas visibles ni utilisables par d'autres composants iPOJO que le composite les contenant. A moins que le composite n'exporte les services de ces composants. La Figure 19 présente les deux schémas de composition supportés par iPOJO.

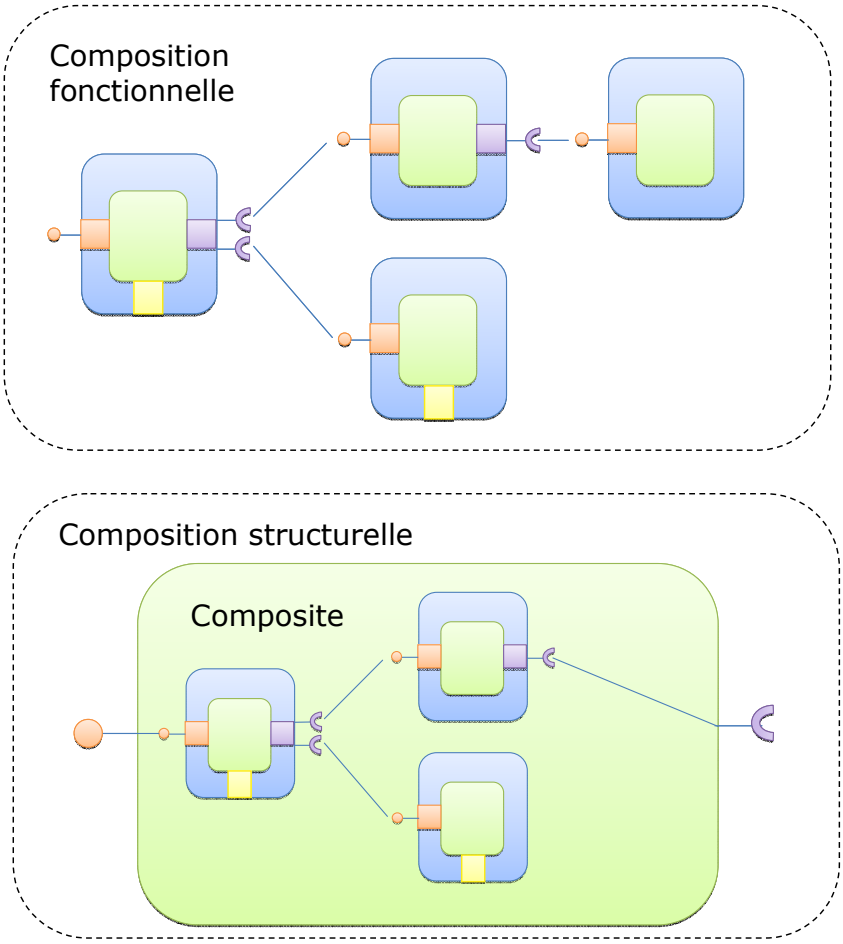

Figure 19. iPOJO supporte la composition fonctionnelle (en-haut) et structurelle (en-bas)

II.4.3.4 Spring Dynamic Modules

Spring Dynamic Modules [Colyer et al 2007] est un autre modèle à composants à services au-dessus d'OSGi. Le canevas Spring [Spring2010] cible les serveurs d'applications et a été porté sur OSGi pour tirer profit des capacités de déploiement et de modularité fournies par OSGi.

Spring Dynamic Modules a rajouté les notions de service au canevas Spring afin que les composants Spring, appelés *beans*, puissent fournir et requérir des services OSGi. L'idée principale de Spring Dynamic Modules est de créer des groupes de composants appelés contextes d'application à l'intérieur de bundles OSGi. Dans un contexte d'application, les composants communiquent en utilisant les mécanismes fournis par Spring. Cependant, entre deux groupes (et donc deux bundles puisqu'il n'y a qu'un contexte par bundle) les communications se font par l'intermédiaire de services OSGi.

Un descripteur d'un contexte d'application décrit à la fois la configuration d'un groupe de composants et ses liaisons de services avec d'autres applications. Les liaisons sont exprimées sous forme de dépendances vers une l'interface de service requise. A l'instar des autres modèles à composants pour OSGi, chaque dépendance peut spécifier les attributs suivants : le filtre, le fournisseur de service désiré, l'optionalité, la cardinalité, l'ordre de

classement des fournisseurs et un temps d'attente maximum. Le temps d'attente est utilisé lorsqu'un composant tente d'accéder à un service qui n'est pas disponible. Si le service n'est toujours pas accessible après le temps spécifié, une exception est levée.

Spring Dynamic Modules est un canevas utilisant l'inversion de contrôle et l'injection. Comme dans le cas de Service Binder, les dépendances de service sont injectées par réflexion via des méthodes. Tout comme les autres modèles à composants orientés service présentés précédemment, Spring Dynamic Modules conditionne le cycle de vie des services fournis par l'état des dépendances de services. Lorsque les dépendances de service d'un bean ne sont pas satisfaites, les services fournis par ce bean ne sont pas publiés.

Du point de vue de la composition, Spring Dynamic Modules propose un mécanisme de composition structurelle, tout comme iPOJO, puisque les contextes d'application sont des composites regroupant plusieurs beans. Cependant à la différence d'iPOJO les communications au sein d'un groupe de composants ne suivent pas les paradigmes de l'approche à service et sont spécifiques à Spring.

Spring Dynamic Modules a néanmoins l'avantage de proposer un modèle de développement simple (s'appuyant aussi sur les principes de POJO) et une infrastructure efficace pour créer des applications à services dynamiques. Cependant, le modèle de composition proposé est limité, il ne permet pas de substituer un fournisseur de service à l'intérieur d'une composition, et un contexte d'application n'est pas reconfigurable dynamiquement.

II.4.3.5 Peaberry

Le projet Peaberry [Peaberry2009] est une extension pour Google-Guice [Guice2009] qui supporte l'injection dynamique de dépendances de services en utilisant un mécanisme à base d'annotations. Google-Guice est une machine à injection suivant visant à simplifier le code métier grâce à l'utilisation d'annotations et du patron de conception de *fabrique*. Peaberry s'intègre parfaitement dans OSGi, mais propose aussi un mécanisme extensible à base de plug-ins qui lui permet de supporter d'autres annuaires de services, et donc d'autres plates-formes.

Le modèle à composants Peaberry se présente sous la forme d'un bundle OSGi qui peut être déployé sur n'importe quel plate-forme OSGi compatible avec la specification R4, telles que Apache Felix ou Eclipse/Equinox.

II.4.3.6 Service Component Architecture

Service Component Architecture (SCA) [Beisiegel et al 2007] est un ensemble de spécifications établi par le consortium OSOA rassemblant plusieurs grands groupes industriels (IBM, BEA, Oracle, SAP, SUN, SIEMENS, ...), avant que sa version 1.0 ne soit prise en charge par l'OASIS en 2007 afin d'être standardisée. Ces spécifications décrivent un modèle structurel pour bâtir des applications suivant une architecture orientée service. Le but de SCA est de simplifier l'écriture d'application indépendamment des technologies

utilisées pour la réalisation (par exemple, Java, BPEL, EJB ou C++). IBM WebSphere, BEA Aqualogic, ou le projet open-source Apache Tuscany sont les implémentations les plus connues de SCA.

La spécification SCA préconise la construction d'architectures orientées service à partir de composants. Dans ce but, elle fournit entre autres :

- un modèle d'implémentation, décrivant la structure d'un composant.
- un modèle d'assemblage spécifiant comment les compositions et applications se construisent.

Un composant SCA, qu'il soit primitif ou composite (cf. Figure 20), possède un conteneur en trois parties constitué :

- des **services**, c'est-à-dire les fonctionnalités qu'il offre aux autres composants. Bien que SCA reste agnostique de toute technologie, les spécifications de services sont exprimées dans le langage de la plate-forme à services utilisée. Le plus souvent il s'agit de descripteurs WSDL ou d'interfaces Java.
- des **références** qui expriment les services que requiert le composant pour assurer son fonctionnement. Ces services requis peuvent être soient fournis par des composants SCA, soit par des systèmes tiers (exposant des services Web ou communicant par JMS, par exemple). Ceci est exprimé à travers des liaisons (par exemple SOAP/HTTP, JMS, JCA, IIOP, ...). Les références sont en quelque sorte des dépendances de service. Ainsi, un composant doit décrire pour chaque référence l'interface requise, la cardinalité, l'optionalité et peut définir de manière déclarative des politiques concernant les propriétés extra-fonctionnelles (sécurité, transaction, ...) à assurer, ainsi qu'un identifiant unique.
- des **propriétés** qui correspondent à la configuration du composant. Ces propriétés sont configurées à l'assemblage et servent au moment de l'instanciation du composant SCA.

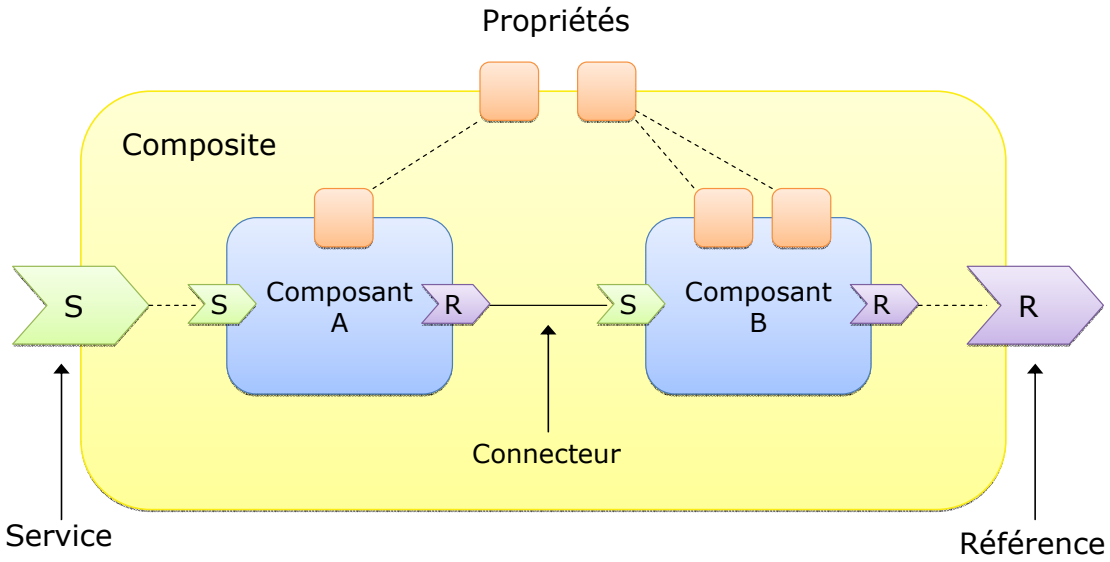

Figure 20. Modèle d'assemblage et d'implémentation de SCA

Le modèle d'assemblage proposé par SCA est un modèle de composition structurelle. SCA définit un langage de composition nommé SCDL (pour *Service Component Definition Language*) pour décrire les applications. Ainsi, dans SCA, une application est en fait un assemblage de types de composant communiquant via des services. Un type de composant peut avoir plusieurs implémentations, mais toutes doivent suivre le patron défini dans le type de composant. Les implémentations sont choisies lors du déploiement de l'application. A l'intérieur d'un composite, les composants sont liés à l'aide de connecteurs fixes appelés *wires*. Une fois l'assemblage réalisé, ces connecteurs ne peuvent plus évoluer durant l'exécution de l'application : la composition est établie en se basant sur les identifiants uniques des références

De plus, le modèle de composition décrit dans SCA est hiérarchique. C'est-à-dire qu'un composite réalisé à partir de composants de rang N est un composant de rang N+1, et se comporte conformément au modèle d'implémentation de SCA : il peut fournir des services, dépendre de références et être configuré par des propriétés.

Enfin SCA définit la notion de domaine. Chaque domaine correspond à une implémentation de la spécification SCA. Ainsi il n'est pas toujours possible que des applications SCA de deux domaines différents (construits sur deux infrastructures différentes) puissent interopérer. Ceci est du au fait que SCA n'est pas lié à une technologie particulière et chaque implémentation de SCA supporte un ensemble de technologies n'étant pas forcément compatible avec une autre implémentation.

En conclusion, l'indépendance technologique de SCA a permis à de nombreuses plates-formes de faire profiter SCA de leurs atouts et bénéficiant de la facilité d'intégration apportée par SCA et l'approche à services. C'est par exemple le cas d'Apache Tuscany [Tuscany2008] et Newton [Newton2009] qui utilisent OSGi pour offrir de la modularité et des facilités de déploiement à travers le concept de bundle aux composants SCA, tout en bénéficiant du modèle de composition hiérarchique de SCA et de l'accès distant à des services OSGi. C'est aussi le cas d'OW2 FraSCAti [Seinturier et al 2009] qui fournit un support d'exécution pour des composants SCA réalisés en suivant les principes du modèle à composants Fractal [Bruneton et al 2004]. FraSCAti permet ainsi à Fractal de pénétrer le monde des architectures orientées service, et à SCA de bénéficier de Fractal pour la création de composants reconfigurables.

Toutefois, même si SCA a l'avantage de proposer un modèle à composant indépendant technologiquement qui de surcroît supporte la composition structurelle hiérarchique, certains points clés sont laissés à l'initiative de ses implémentations. Par exemple, SCA ne définit pas ni ne réutilise de langage de description de service. De plus, les politiques associés à la gestion des connecteurs entre composants ne sont pas spécifiées et dépendent de chaque implémentation. Enfin, SCA partage un point faible avec les services Web du point de vue du dynamisme. Tout comme les liens entre composants référencent explicitement d'autres composants via leurs identifiants uniques, les fournisseurs de services sont choisis lors de la phase de déploiement, ce qui empêche leur substitution à l'exécution et ne permet pas de supporter la disponibilité dynamique de services.

II.5 Synthèse

L'approche à service a rencontré un franc succès à travers l'expansion des services Web et de la multitude de standards WS-* qui les étendent, à la fois sur le Web en plein essor (qui exploite également l'approche REST), et en tant que technologie d'intégration dans les entreprises, notamment par l'utilisation d'ESB. Le couplage faible entre consommateurs et fournisseurs de service et le mécanisme de liaison tardive permet aux éléments d'une application construite sur une architecture orientée service d'évoluer de manière indépendante.

Néanmoins l'approche préconisée par les services Web peut être qualifiée de statique dans le sens où elle n'exploite pas totalement les possibilités offertes par le paradigme de l'approche à service. Ceci est en partie du à la difficulté de gérer efficacement des annuaires de service à l'échelle de l'internet. De plus un service Web ne s'exécute pas sur la même échelle de temps qu'un service technique utilisé dans le cœur d'un serveur d'application, les besoins et l'usage sont différents. Dans le cas des services Web, la fréquence d'utilisation est moins haute, les liaisons sont de plus courte durée mais les interactions plus longues (certains processus peuvent prendre plusieurs jours).

Or l'approche à service révèle son utilité dans d'autres domaines que la récupération d'information sur la toile (prévisions météorologiques, cours de la bourse, ...), le e-commerce ou l'intégration de logiciels d'entreprise. En particulier dans les domaines où les services évoluent dynamiquement et où les applications doivent réagir face à ce dynamisme afin de prendre en compte les changements de contexte. Par exemple UPnP ou OSGi sont apparus pour répondre à la problématique des réseaux domestiques.

En effet, les architectures orientées service, même dans leur forme étendue ne permettent pas de répondre à la problématique du dynamisme. Même si le mécanisme de liaison retardée permet de connecter les éléments d'un SOA à l'exécution, l'architecture une fois établie reste statique. Les services Web et UDDI ne proposent pas par exemple de mécanisme de découverte passive. Par conséquent, non seulement la phase de découverte n'est pas toujours respectée, mais il est aussi habituel qu'un client ne cherche qu'une seule fois le service qu'il veut utiliser. Si le service devient indisponible par la suite, le client n'a pas de moyen de gérer ce niveau de dynamisme.

Bien que les services Web soient l'architecture orientée services la plus répandue, il existe d'autres technologies mettant en œuvre les concepts de l'approche à services et capables de prendre en compte la disponibilité dynamique des services. Nous les appelons *architectures dynamiques orientées service*.

Mais le gain en flexibilité et adaptabilité se paie par une augmentation de la complexité et du coût de développement. L'approche à composant permet de réduire cette complexité, le développeur peut se concentrer sur le code métier de l'application sans qu'il soit parasité par la gestion du dynamisme.

Enfin, l'intérêt pour l'approche à service continue sa phase de croissance. La tendance actuelle est d'un côté aux services ubiquitaires pour tout ce qui concerne l'informatique diffuse et l'internet des objets, et d'un autre côté l'internet des services est en train

d'émerger avec à sa tête l'informatique en nuages (*cloud computing*) et le SaaS (*Software as a Service* ou XaaS de manière plus générale pour *X as a Service*). SaaS est un paradigme qui présente l'application en tant que service accessible depuis n'importe où grâce à la démocratisation de l'accès à internet, par opposition à la vision de bureau où les applications sont installées sur chaque ordinateur personnel. Dans cette vision déjà réalisée par certaines applications internet riches –comme les Google Apps visant le grand public-, l'application est hébergée par des fournisseurs d'infrastructures louant des ressources de calcul. Les clients utilisent alors ce type d'applications comme un service, changeant par la même occasion le modèle économique du logiciel puisqu'il n'y a plus d'acquisition de licence, l'utilisation peut se faire selon une tarification forfaitaire pouvant dépendre de nombreux paramètres. De même comme l'application n'est plus installée sur le matériel du client, ce dernier perd le contrôle qu'il pouvait avoir dessus. Toutefois si ce système permet de réduire les coûts à court terme, les enjeux sur le long terme ne sont toujours pas clairs. Les notions de *SaaS* et de *fournisseurs d'applications hébergées* sont abordées dans le chapitre suivant, section III.1.

Pour conclure ce chapitre, les Tableau 2, 3 et 4 ci-dessous présentent un récapitulatif des architectures orientées service et des modèles à composants orientés services étudiés dans les sections précédentes. Plusieurs critères en rapport avec le dynamisme ont été retenus afin de comparer l'apport de chaque technologie dans des contextes applicatifs fortement dynamiques. Nous avons donc porté notre attention sur tout ce qui a trait à l'annuaire de service, à ses mécanismes permettant la découverte passive et donc l'adaptation au contexte et enfin à la gestion simplifiée du dynamisme et des liaisons à travers des modèles à composants.

	Services Web	**UPnP / DPWS**	**Jini**	**OSGi**
Spécification	WSDL et ses extensions	Format spécifique en XML (UPnP), WSDL étendu (DPWS)	Interface Java et attributs	Interface Java et propriétés de service
Annuaire	Registres UDDI répliqués; Peut exposer un service	Pas d'annuaire. (les consommateurs doivent maintenir leur propre liste de fournisseurs)	Fédération distribuée; Expose un service de *lookup*	Centralisé et unique
Découverte passive	N/A	SSDP	RemoteEvent	ServiceEvent
Sélection	Complexe	Manuelle, d'après le descripteur de service	Selon les valeurs exactes des attributs	Filtre LDAP sur les propriétés de service
Publication et retrait	Pas de retrait. Distribution du WSDL	Annonces multicast (GENA et SSDP)	Via le *lookup service* ou mécanisme de baux	Via le *Bundle Context*

Tableau 2. Comparatif de plates-formes à service

	Services Web	UPnP / DPWS	Jini	OSGi
Langage de programmation	Non spécifié	Non spécifié	Java	Java
Liaison	HTTP/SOAP	HTTP/SOAP	Java RMI	Référence Java locale
Composition	Orchestration (BPEL), chorégraphie (WS-CDL) et structurelle (SCA)	N/A Manuellement, aux points de contrôle	N/A Manuelle	OSGi de base: non Avec SOCM: oui
Modèles à composants	SCA	N/A	N/A	SB, DS, Spring DM, iPOJO, ...

Tableau 3. Technologies utilisées par différentes plates-formes à service

	Spring DM	Service Binder / DS	iPOJO	SCA
SOA cible	OSGi	OSGi	OSGi	Non spécifiée
Description	Déclarative (méta-données XML format Spring *bean*)	Méta-données XML	Déclarative (méta-données XML ou annotations) et program-matique	Déclarative (méta-données XML) types de composants et compositions
Gestion dynamisme	Oui	Oui	Oui	Non spécifiée, composition statique
Gestion autres aspects	Services techniques divers	Non	Extensible (mécanisme de *handlers*)	Politiques de sécurité et de transaction
Politiques de découverte et de liaison	Dynamique en dehors des compositions. Possibilité de spécifier un fournisseur et un temps d'attente	- optionnelle - statique - dynamique	- optionnelle - statique - dynamique - autres extensions (politique temporelle (temps d'attente))	Statique. Possibilité de reconfiguration dynamique (avec FraSCAti par exemple)

Tableau 4. Caractéristiques de modèles à composants orientés service

Chapitre III

Accords de niveau de service

> " *La vie c'est comme une boîte de chocolats, on ne sait jamais sur quoi on va tomber.* "
>
> Forrest Gump, extrait de *Forrest Gump*.

La notion d'accord de niveau de service est apparue dans le domaine des télécommunications il y a un peu moins d'une trentaine d'années, et est devenue aujourd'hui une préoccupation dans les architectures orientées services « étendues » pour la mise en œuvre des besoins extra-fonctionnels concernant la qualité de service (cf. le schéma pyramidal de la Figure 9). Un accord de niveau de service crée une compréhension mutuelle des besoins et capacités de chaque partie, et leur permet ainsi d'arriver à un ensemble d'aspirations communes et raisonnables concernant la livraison du service.

Le but d'un accord de niveau de service est de prévenir les conflits, en clarifiant les attentes mutuelles des parties qui y sont engagées [Miller1987]. Ainsi, l'accord est établi dès le départ plutôt qu'attendre que la situation ne dégénère et ne doivent être réglée à un stade ultérieur d'une collaboration où ce serait inapproprié.

Grâce à l'accord de niveau de service qui fournit un cadre général de compréhension des parties, le client dispose de garanties sur la qualité du service perçu, et le prestataire, bien qu'il soit tenu à des obligations de résultat, ne fait pas face à des attentes parfois irréalistes de la part de ses clients.

Ce chapitre présente une étude générale des accords de niveau de service (SLA pour *Service Level Agreements*). Il revient d'abord sur l'origine des accords de niveau de service et le développement de ce domaine, avant d'analyser les caractéristiques des SLAs et de présenter plusieurs travaux académiques et industriels s'intéressant à différents points des accords de niveau de service.

III.1 Fondements des accords de niveau de service

Un accord de niveau de service est un contrat de service garantissant un minimum de qualité de service. Les premiers accords sont apparus il y a plus de 20 ans dans le domaine des télécommunications et des réseaux informatiques.

En définissant les conditions de prestation d'un service par un accord de niveau de service, les parties engagées possèdent des garanties quant au service fourni et à son usage, tout en s'assurant la couverture des cas où les termes ne seraient pas respectés. C'est-à-dire dans les cas où le niveau du service fourni serait inférieur à ce qui est stipulé dans l'accord, ou bien en cas d'usage excessif de la part du consommateur.

III.1.1 Contrats et Qualité de Service

Un accord de niveau de service est un ***contrat*** *entre au moins deux parties, dont les clauses portent sur la* ***qualité*** *du (ou des) service(s) faisant l'objet du contrat.*

Le contrat tel que nous le connaissons actuellement est hérité du droit romain même si la première mention de la notion de contrat fut découverte dans le Code d'Hammurabi, sixième roi de Babylone (~1750 av. J.-C.). Un contrat se décompose en deux parties : le *negotium*, la substance sur laquelle les parties sont en accord, et *l'instrumentum*, le support du contrat ayant valeur de preuve en cas de litige. Généralement *l'instrumentum* est une trace écrite (parfois remplacée par le support numérique ces dernières années).

Une classification des contrats logiciels a été établie par Beugnard et al [Beugnard1999]. Elle distingue quatre niveaux de contrat représentés sur la Figure 21.

Le niveau **syntaxique** se résume à ce qu'on peut trouver en règle générale dans les interfaces des langages de programmation classiques et dans les langages de définition d'interfaces (IDLs, interfaces Java, ...). C'est-à-dire les opérations disponibles, les paramètres d'entrées et de sorties requis et éventuellement les exceptions qui pourraient être levées durant ces opérations.

A son état le plus basique un contrat de niveau **comportemental** spécifie le comportement de chaque opération en utilisant des assertions booléennes appelées pre- ou post-conditions et invariants. Dans [Meyer1992] la conception orientée contrat (*Design by Contract*) implémente ce niveau de contrat grâce au langage Eiffel. Par la suite, Java ou encore UML [OMG2010] ont adopté cette approche à travers les assertions Java et OCL (*Object Constraint Language*). Des langages de contrats comportementaux ont aussi été créés pour certains modèles à composants, notamment pour Fractal [Collet2005].

Un contrat de niveau **synchronisation** spécifie le comportement global d'objets en termes de synchronisations entre appels de méthodes. Une solution serait d'attacher aux objets des politiques de synchronisation telles qu'une stratégie à Mutex par exemple.

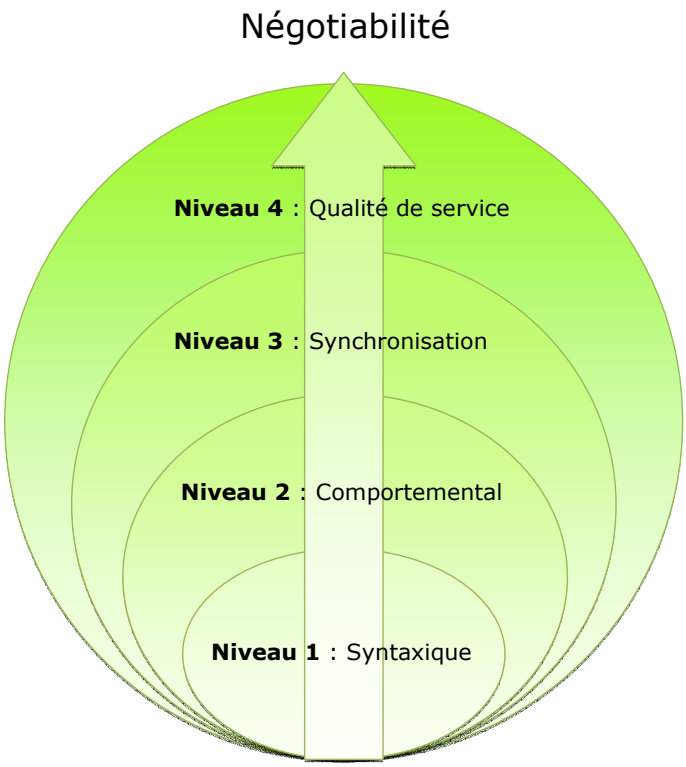

Figure 21. Les quatres niveaux de contrat

Le niveau **qualité de service** [Campbell1994] a pour but de qualifier et quantifier la livraison d'un service du point de vue de critères spécifiques à un domaine. Le Tableau 5 montre plusieurs critères de qualité de service en fonction des domaines d'application. La qualité de service (QoS) est une notion qui a émergé dans le domaine des réseaux informatiques. C'est pourquoi la QoS est souvent associée à des critères tels que le temps de réponse, la bande-passante, etc. C'est à ce niveau de contrat que se définissent les accords de niveau de service.

Domaines	Réseaux	Grilles de Calcul (réservation de ressources)	Serveurs Web	Vidéo
Critères	temps de réponse, gigue, latence, bande passante, débit, classe du trafic, ...	nombre de CPU, mémoire, capacité de stockage, bande passante, ...	nombre de hits par seconde, temps de réponse, max de connexions simultanées, disponibilité, MTTR, ...	frames/sec, résolution, encodage, échantillonnage, intégrité image, fréquence de la piste audio, ...

Tableau 5. Critères de qualité de service selon les domaines d'application

Un service composé d'un ensemble de fonctionnalités (propriétés fonctionnelles), peut-être décliné en plusieurs services aux propriétés extra-fonctionnelles différentes, et donc de qualité différente. Par exemple l'architecture réseau DiffServ [Nichols et al 1998] spécifie un mécanisme permettant de gérer le trafic réseau en différentes classes offrant des qualités de service différentes. Dans le cas de DiffServ la qualité de service portera sur des métriques propres au domaine du réseau : temps de réponse, gigue, latence, bande passante, débit, ... La qualité d'un service, et par extension les contrats portant sur cette qualité, sont parfois exprimés par les termes suivants : Best-Effort (aucune garantie), Silver, Gold, Platinum, Premium, ... [Dixit2001]

Par exemple, les applications critiques comme le trafic vidéo ou audio nécessitant de meilleures performances que le trafic web, se verront allouer plus de ressources.

Un accord de niveau de service est donc un contrat portant sur la prestation d'un service du point de vue de la qualité, d'un ensemble de critères.

III.1.2 Historique des accords de niveau de service

La notion d'accords de niveau de service a émergé au début des années 80. La frise présentée sur la Figure 22 retrace l'évolution des domaines d'application des SLAs depuis ce temps jusqu'à aujourd'hui.

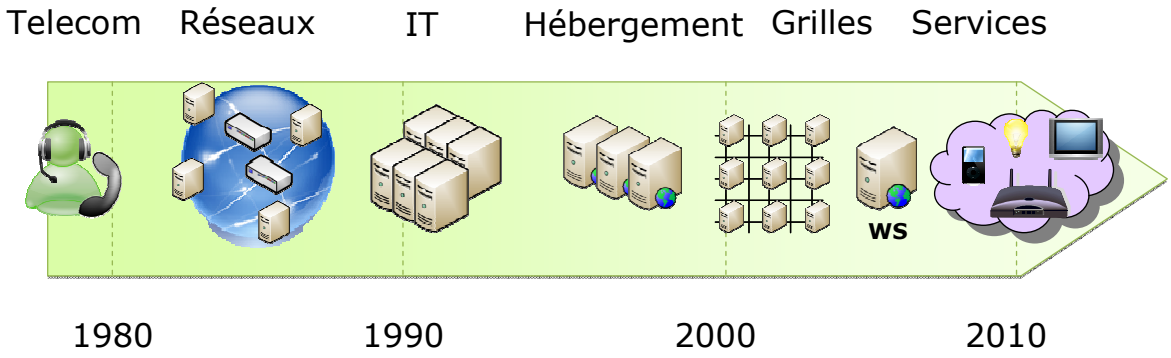

Figure 22. Tendance des domaines d'application des SLA depuis plus de 30 ans

Les opérateurs de télécommunications ont été les premiers à parler d'accords de niveau de service. Ils pouvaient ainsi garantir aux entreprises à qui ils vendaient leur service un certain niveau de qualité, en se basant par exemple sur la disponibilité de leur service, le nombre d'appels perdus ou le temps réponse.

Par la suite, cette pratique s'est étendue aux opérateurs de réseaux informatiques du fait de la nécessité de différencier le trafic et d'offrir différents niveaux de qualité de service. Avec l'essor d'internet, les fournisseurs d'accès ont eux aussi établi des accords de niveau de service avec leurs clients [Asawa1998].

En parallèle les SLAs ont fait leur apparition dans des secteurs « critiques » tels que l'armée [Adam1986] ou la finance [Propst1985, Sherkow1986] afin de garantir le niveau de service des infrastructures informatiques utilisées. Les accords de niveau de service montrent en effet leur utilité dans ces domaines critiques (secteurs de la santé, financier,

des transports, de l'énergie et militaire) où il est important de disposer de garanties à défaut de pouvoir assurer une sûreté de fonctionnement.

Le procédé a été ensuite repris et généralisé par d'autres secteurs qui trouvèrent dans les SLAs un moyen d'évaluer l'efficacité de leur département informatique. Ainsi si les résultats ne sont pas satisfaisants, ces entreprises peuvent décider de sous-traiter la gestion de leurs systèmes à d'autres fournisseurs de services IT. L'accord de niveau de service en permettant de règlementer les collaborations transversales entre organisations, facilite l'externalisation et la sous-traitance de tâches. De manière plus générale, un SLA ne se limite pas forcément à des organisations indépendantes et peut contribuer au bon fonctionnement du marché interne d'une entreprise entre ses différentes composantes.

Depuis la fin des années 90, il existe de nombreuses solutions d'hébergement, aussi bien pour des serveurs (web, courrier électronique, ...) que pour l'hébergement de données (offres qui restent d'actualité [Amazon2007]). La plupart de ces fournisseurs de services d'hébergement proposent alors à leurs clients des accords de niveau de service afin que ce dernier puisse avoir des garanties lui permettant de différencier les offres du marché.

Dans le domaine de l'hébergement, on parle aussi d'externalisation (*outsourcing*) pour les fournisseurs qui confient l'hébergement de leurs applications à un hébergeur externe. Ces fournisseurs ne contrôlent pas les infrastructures sur lesquelles sont déployées leurs applications et ont donc besoin de garanties sur le service d'hébergement.

Plus récemment se sont développés ce que l'on nomme des fournisseurs d'applications hébergées (FAH ou ASP pour *Application Service Provider*) fournissant, sur abonnement, un accès à des applications ainsi que l'infrastructure et la maintenance nécessaires. Cette approche rejoint les domaines de l'informatique utilitaire [Buco et al 2004] ou en nuage [Vouk2008, Cloud2010] qui proposent aux utilisateurs des « applications en tant que service » (*Software as a Service*). Dans un tel contexte, le FAH est amené à collaborer avec d'autres partenaires spécialistes dans leur domaine : service, technologie des réseaux et leur gestion, administration, intégration, gestion et support des applications, etc. Toutefois, du point de vue de l'accord de niveau de service, le FAH est le seul responsable de la livraison du service à l'égard du client. D'où l'intérêt pour le FAH de contracter des SLAs avec ses multiples partenaires (fournisseurs d'infrastructures matérielles, opérateurs réseaux, etc).

Enfin, à partir du début des années 2000 les SLAs se sont vus appliqués dans les grilles informatiques (*grid computing*) pour garantir l'exécution de tâches et la réservation de ressources [Quan2006], et ont naturellement fait leur apparition dans le domaine des architectures orientées service, principalement au niveau des services Web [Jin et al 2002]. La gestion des accords de niveau de service est effectivement une problématique représentée comme transversale aux architectures orientées service étendues (cf. section II.2.2).

Avec les services Web, certains aspects comme la définition, la négociation et la supervision des SLAs, qui jusqu'alors étaient des tâches conduites par des opérateurs humains, commencent à être traitées par l'informatique.

En effet, avec l'évolution de l'informatique et des infrastructures sur lesquelles elle s'appuie, et en considérant les tendances actuelles autour des grilles, des nuages et de l'informatique utilitaire d'une part, et de l'informatique enfouie et ubiquitaire d'autre part, la granularité des services ciblés par les SLAs diminue et passe au niveau « macro ». Ce

qui est justement possible grâce à « l'informatisation » des accords de niveau de service et de leur gestion.

Il est donc envisageable d'imaginer pour l'avenir, des SLAs dans un contexte d'intelligence ambiante, entre des services ambiants rendus par des équipements et/ou des briques logicielles. En particulier dans des secteurs « mission-critiques » où la qualité de service et son respect jouent un rôle primordial.

Toutefois malgré l'émergence de nouveaux domaines d'application, il y aura toujours besoin d'accords de niveau de service pour les domaines d'application de la première heure tels que les opérateurs réseaux [Boschi et al 2006] ou les services d'hébergement. Et même si certaines considérations des SLAs étaient propres à l'informatique de l'époque (on y parle de centre de données, de bandes magnétiques, d'unités centrales, … [Miller1987]), d'autres préoccupations telles que la disponibilité d'un système, peu importe sa granularité, sont toujours d'actualité.

III.2 Principes généraux des accords de niveau de service

Dans cette partie nous examinerons les accords de niveau de service sous trois angles :

- le contenu d'un SLA, c'est-à-dire les éléments constitutifs qui le composent ;
- le cycle de vie d'un SLA en quatre étapes: définition, négociation, renégociations et terminaison ;
- la gestion des SLA plus connue sous l'acronyme SLM : les outils et procédures visant à superviser le degré de respect d'un SLA.

III.2.1 Forme

A partir de modèles existants d'accords de niveau de service [Miller1987, Lamanna2003, Ludwig et al 2003, Andrieux et al 2004, Aiello2005], nous avons pu dégager les principales composantes d'un SLA. Le schéma UML sur la Figure 23 présente la modélisation de la forme d'un SLA.

Pour citer la Bibliothèque pour l'Infrastructure des Technologies de l'Information [ITIL2008] :

> *" L'accord de niveau de service doit être lui-même suffisamment détaillé et avoir une portée adaptée au service couvert. Un SLA inclut typiquement les sections suivantes : introduction, champ d'action, performance, suivi et rapport, gestion des problèmes, compensation, devoirs et responsabilités du client, garanties et recours, sécurité, droits de la propriété intellectuelle et informations confidentielles, conformité légale et résolution des litiges, terminaison et signatures. D'autres sections peuvent être aussi envisagées. "*

Bien qu'elle soit plus détaillée que notre proposition, cette définition corrobore le découpage effectué et les aspects techniques tels que les devoirs, recours, droits, etc, peuvent être regroupés sous le concept de « terme » ou « clause ».

Les sections suivantes décrivent les trois grandes parties que nous avons dégagées : la définition du service faisant l'objet du contrat, le contexte de l'accord, et enfin les clauses du contrat.

III.2.1.1 Définition du service

Une première partie concerne la définition du service que le prestataire s'engage à fournir (cf. partie supérieure sur la Figure 23). Il s'agit de la spécification fonctionnelle du service. Par exemple, dans un document SLA « classique », la spécification serait textuelle et verbeuse, décrivant précisément le service rendu, tandis que dans le cas des services Web, la spécification serait exprimée par un lien vers un descripteur WSDL.

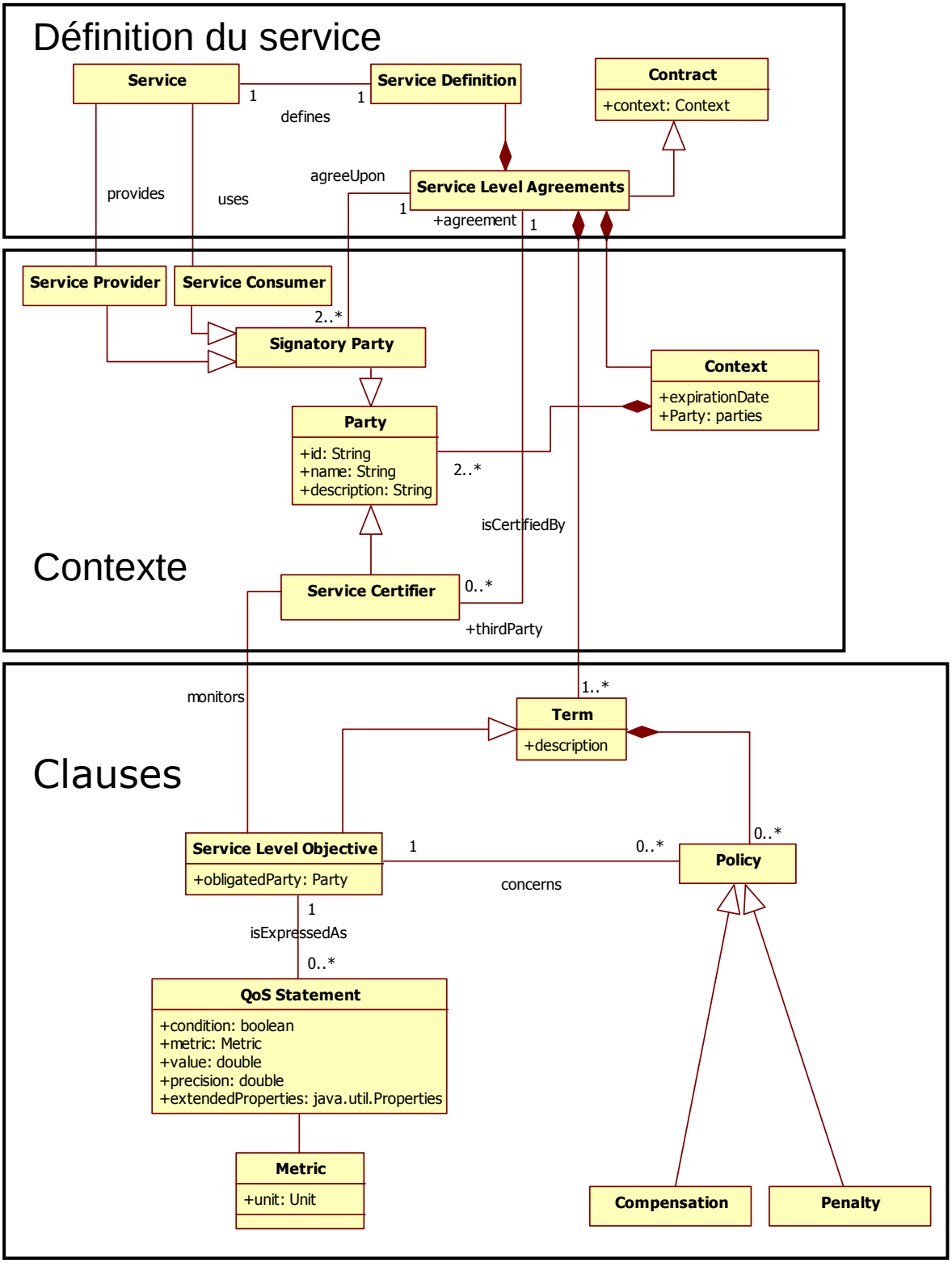

Figure 23. Représentation UML du modèle de contenu d'un accord de niveau de service

III.2.1.2 Contexte

La seconde partie (partie centrale sur la Figure 23) concerne le contexte de l'accord : quelles sont les parties qui y sont impliquées et sur quelle période est-il valide.

Les parties décrites dans l'accord de niveau de service désignent aussi bien les parties signataires (i.e., le ou les prestataire(s) comme le ou les consommateur(s)) que les parties tierces chargées de superviser les interactions de ces dernières et d'évaluer le respect de l'accord (cf. section III.2.3). Les informations sur les parties doivent au minimum permettre de les identifier de manière unique, mais on peut aussi trouver dans un SLA des données plus complètes telles que leurs adresses ou des descriptions détaillées.

Ces parties signataires peuvent être de natures diverses, et comme souligné dans la partie historique (section III.1.2), leur granularité peut aller de l'organisation (réelle ou virtuelle [Arenas et al 2008, Hovestadt et al 2006]) à l'entité logicielle (contrats entre système, plate-forme, conteneur et composant [Zschaler2004]).

Enfin la validité d'un SLA comprend la date à partir de laquelle il entre en application et sa date d'expiration (qui peut aussi être déduite de sa durée de validité), c'est-à-dire le moment à partir duquel il ne sera plus considéré comme valide.

III.2.1.3 Clauses

Les termes du contrat représentent le cœur de l'accord. *Termes*, *clauses* ou *stipulations*, sont des dénominations assez générales et désignent aussi bien les engagements chiffrés d'une partie que les politiques de terminaison ou le prix du service. Les clauses comportent donc d'un côté ce qui est appelé communément « objectifs de niveau de service » (SLOs pour *Service Level Objectives*), et de l'autre diverses politiques.

Objectifs de niveau de service

Les SLOs sont des expressions de la qualité de service que garantie ou exigée par chaque partie. C'est-à-dire qu'ils concernent non seulement les objectifs du fournisseur de service, mais dans le cas où le prestataire imposerait des contraintes au consommateur, même si c'est plus rare, ce dernier pourrait aussi se voir attribuer des « objectifs de niveau d'utilisation ». Par exemple, un utilisateur peut être amené à s'engager à ne pas dépasser un certain volume de données transférées s'il veut continuer à utiliser le service d'un hébergeur. Dans la littérature, ces objectifs sont parfois appelés *obligations*.

Un objectif de niveau de service comprend donc :

- la **partie obligée**. Généralement cette partie n'est pas désignée, il est alors sous-entendu qu'il s'agit du prestataire.
- le **critère de qualité de service** aussi appelé métrique, indicateur de performance ou KPI (pour *Key Performance Indicator*) qui peut être exprimé dans une certaine unité. Par exemple un temps réponse maximum en millisecondes, une disponibilité en pourcentage, un espace de stockage en giga-octets. Cet indicateur peut être assorti d'une précision de la mesure ; ainsi pour

un temps de réponse de 100 ms, une marge d'erreur de 10 ms pourrait être tolérée.

- de manière optionnelle, les obligations peuvent être soumises à certaines **conditions de validité**. La validité peut se réduire à une fenêtre temporelle précise (e.g., à l'image du principe d'heures « creuses »), ou à d'autres conditions plus particulières (comme l'état d'un système à un instant donné par exemple). On parle parfois de portée ou de *scope* pour désigner le domaine de validité d'une obligation.
- des **informations complémentaires** : les SLOs peuvent être identifiés par un nom, un identifiant (numéro d'article) et être détaillés par une description textuelle.

Politiques

Les clauses d'un accord de niveau de service comportent à l'instar de tout contrat diverses politiques. Ces politiques, en plus de définir par exemple le prix du service permettent de prévoir les futurs litiges potentiels et de leur apporter des solutions. Ces politiques font généralement mention des crédits de service, compensations financières, pénalités ou autres conséquences de la fourniture défectueuse des services.

Des politiques peuvent faire référence à un SLO ou de manière globale à plusieurs SLOs. C'est le cas des **politiques de recours**. Ainsi quand un ou plusieurs SLOs ne sont pas respectés, ces politiques définissent si l'accord doit être résilié, si une compensation est prévue pour la partie lésée, ou bien si une pénalité doit être appliquée à la partie fautive.

Les parties peuvent bénéficier de diverses formes de **compensation** dont :

- la compensation d'un consommateur en cas de défaillance de la part du prestataire peut-être financière (remboursement, réduction sur le prix du service, ou dédommagement : une somme prévue par l'accord est versée à la partie lésée) ou se faire sur le service (le consommateur peut bénéficier d'un plus gros volume de service ou d'une meilleure qualité par exemple).
- la facturation du service (le prix pouvant être forfaitaire, dépendre de l'utilisation du service, ...) est en quelque sorte une compensation perçue par le fournisseur en échange de sa prestation.

Les **pénalités** pouvant être appliquées à la partie fautive sont diverses. Une pénalité peut, comme une compensation être strictement financière (amende ou baisse de la tarification dans le cas du prestataire, et surfacturation dans le cas du client), ou bien de nature différente (un fournisseur peut-être mis sur la liste noire d'un client, recevoir une évaluation négative, etc).

Il est aussi courant qu'une politique soit dédiée à la définition des conditions de résiliation. Elle définit alors entre autres les conditions sous lesquelles l'accord peut-être résilié, et ce qu'entraîne la rupture de l'accord. Par exemple, si la résiliation est initiée par un client alors que ce dernier n'est pas encore arrivé au terme de sa période d'engagement, il peut-être amené à dédommager le prestataire auprès duquel il s'était engagé.

III.2.2 Cycle de vie

Le cycle de vie d'un accord de niveau de service se découpe en quatre phases représentées sur la Figure 24 ci-dessous : une première phase qui conduit à la définition des grandes lignes de l'accord et fournit une base à la négociation. Une fois négocié, l'accord peut optionnellement être renégocié au cours de sa vie. Enfin l'accord peut se terminer pour diverses raisons.

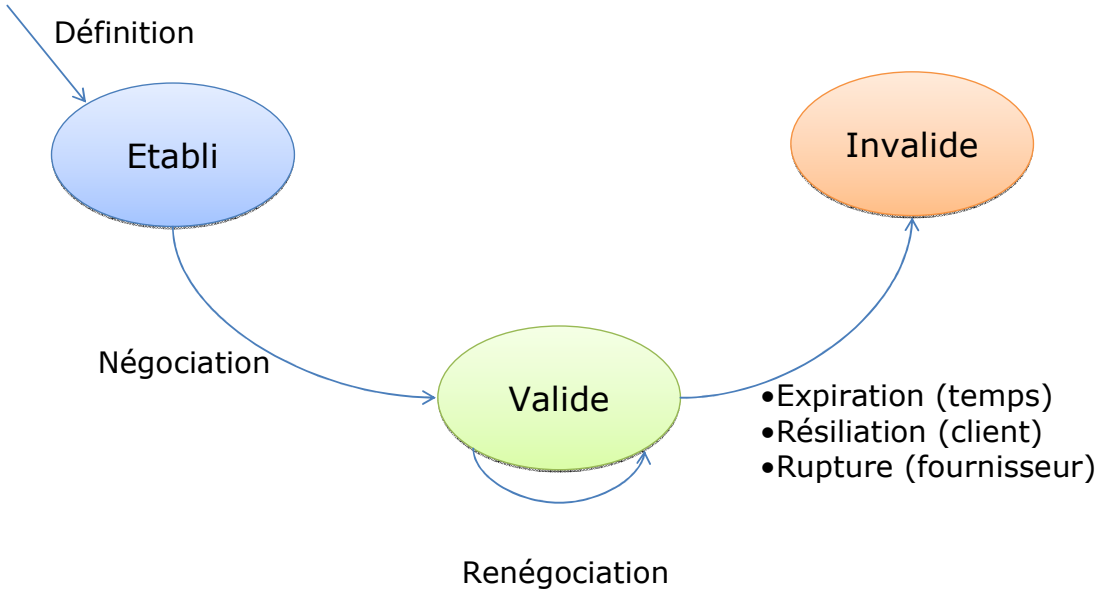

Figure 24. Cycle de vie d'un accord de niveau de service

III.2.2.1 Définition d'un accord de niveau de service

La définition d'un SLA est selon de nombreux guides des bonnes pratiques [Karten2001] une étape importante qui ne doit en aucun cas être négligée et demande une attention particulière, du fait de ses enjeux.

Cette phase consiste à définir les grandes lignes de l'accord pour préparer les négociations. En effet, à partir de cette définition plusieurs accords peuvent être établis suite à un processus de négociation détaillé dans la sous-section suivante.

Pour pouvoir proposer une base de négociation, un fournisseur de service doit connaître ses capacités en termes de niveau de service. Ainsi l'étape de définition implique généralement une évaluation du service, la définition des indicateurs de performance suivie de la prise de mesures. Afin d'établir les lignes directrices d'un SLA, il est aussi souvent fait usage de modèles, plus communément appelés *templates* pour constituer la base du document.

Par analogie avec l'approche à objets, la définition d'un accord est semblable à la définition d'une classe, qui peut être réalisée à partir d'un modèle comme une classe peut

implémenter une spécification ; et l'accord résultant de la négociation est alors assimilable à la création d'un objet par l'instanciation d'une classe.

III.2.2.2 Négociation

Quelque soit le domaine, pour devenir valide et effectif, un contrat doit être approuvé (la signature est une forme d'approbation) par les parties qui y sont engagées. Cette approbation est le fruit d'un processus appelé négociation.

La négociation repose sur la recherche du compromis optimal entre les besoins et les capacités de chaque partie dans les limites de ce que ces dernières peuvent offrir.

Au terme de la négociation les parties échangent leurs consentements pour exprimer l'accord des volontés, qui se matérialisent à travers l'acceptation de l'offre. L'acceptation, qui peut être tacite ou expresse, est l'adhésion au contenu précis de l'offre.

Durant notre étude des accords de niveau de service nous avons distingué trois niveaux de négociations, de la plus simple à la plus sophistiquée (voir Figure 25).

1. Niveau « **sélection** » : la première forme de négociation tient plus de la simple sélection. Le prestataire propose un SLA, et le consommateur a uniquement le choix de l'accepter ou le refuser. S'il accepte, l'accord est validé tel quel, dans le cas contraire le consommateur doit trouver un autre fournisseur. Afin de l'assister dans cette recherche, il est courant qu'un courtier se charge de trouver le service le plus adapté. L'approche la plus classique consiste en l'utilisation de méthodes à base de coefficients [Bottaro2007], de *scoring* [Sierra1997], de réputation [Jurca2005], ou de fonctions d'utilité, pondérant les besoins et offres des consommateurs et des fournisseurs, puis à en déduire la sélection optimale.
    - ex : Avec les offres grand public des fournisseurs d'accès à internet, le client n'a pas le choix de la qualité de sa connexion, il ne peut que sélectionner parmi les abonnements proposés.
2. Niveau « **personnalisation** » : Au second niveau, le fournisseur laisse plus de choix aux consommateurs. Il peut proposer différentes offres prédéfinies et fixes comme dans le cas précédent parmi lesquelles le consommateur choisit celle qui lui convient (ex: niveau Premium, Gold, Silver, ...). Ou bien il peut aussi proposer un contrat dont certaines clauses sont négociables : le consommateur peut choisir la qualité de service qu'il désire tout en restant dans des intervalles prédéfinis, et son choix pourrait avoir des répercussions sur le prix du service ou la durée d'engagement par exemple.
    - ex : Les offres professionnelles des fournisseurs d'accès à internet, laissent quant à elles la possibilité au client de choisir parmi différents débits et qualités de service.
3. Niveau « **conversationnel** » : Enfin le dernier niveau beaucoup moins courant correspond à une véritable négociation stricto sensu : l'accord est construit et élaboré par les différentes parties qui en fixent les clauses au terme d'une discussion. Ce niveau peut être atteint par la mise en place de stratégies et de politiques de négociation [Gimpel2003]. Toutefois le gain dans la liberté de choix se fait au détriment du processus qui est plus complexe et implique plus d'échanges que les deux premiers niveaux.
    - ex : Ce niveau correspond au principe des offres « sur mesure » et formules « à la carte ».

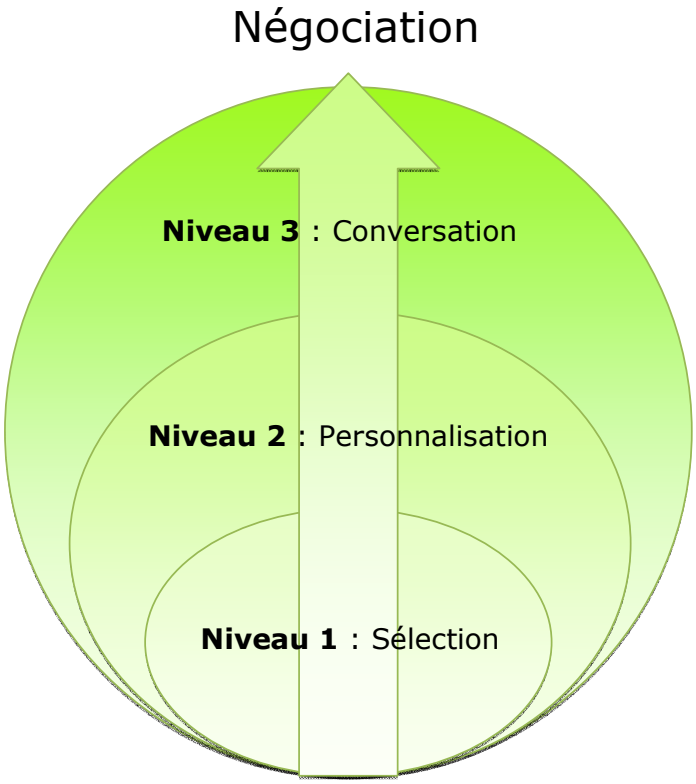

Figure 25. Niveaux de négociation

Au final, selon le niveau de la négociation, l'accord négocié peut plus ou moins différer de l'offre initiale issue de la phase de définition. Néanmoins, une négociation n'arrive pas toujours à son terme et peut échouer si les parties n'arrivent pas à un accord satisfaisant les attentes mutuelles de chacun.

III.2.2.3 Renégociation

Au cours de sa vie, un accord de niveau de service peut être renégocié. Un nouveau besoin ou de nouvelles offres de la part d'une des parties peuvent être à l'origine de cette renégociation. La renégociation d'un accord peut résulter en un simple prolongement de la durée, mais il est aussi possible que les clauses du contrat soient renégociées partiellement ou même dans leur totalité [DiModica2007, Herssens2008]. La renégociation peut alors affecter la valeur d'un paramètre de qualité de service ou toute autre modification de clause (politiques, tarifs, etc.). Par exemple, si un fournisseur d'applications hébergées change son infrastructure, il peut proposer à ses clients une nouvelle grille tarifaire, de meilleures performances (niveau e service plus haut), et les clients peuvent soit accepter la renégociation, soit garder l'ancien contrat.

Dans un autre domaine, celui de la télévision par IP, lorsque les conditions du réseau sont défavorables et le flux vidéo saccadé, un utilisateur pourra accepter de basculer en mode « bas-débit » où les contraintes concernant la qualité de l'image sont moins fortes;

ce qui permet au prestataire de transmettre un flux continu et de respecter ses engagements concernant le nombre d'images par seconde.

III.2.2.4 Terminaison

Un accord de niveau de service peut se terminer pour trois raisons différentes.

- L'accord peut tout simplement ne plus être valide une fois que sa date d'expiration est atteinte : il arrive à son terme.
- La violation d'une clause peut entraîner la rupture de l'accord et donc sa terminaison si une telle mesure est stipulée dans les politiques.
- Enfin, si l'une des parties décide de mettre fin à l'accord, ce dernier n'est plus valide et la partie ayant rompu le contrat peut-être amenée à subir des pénalités. Typiquement, si un client s'engage pour une certaine durée et résilie avant le terme de son engagement, la plupart des contrats stipulent des indemnités financières à verser au prestataire. C'est le cas des abonnements de téléphonie mobile et des contrats où le client engage sa fidélité.

III.2.3 Supervision du niveau de service

Un accord de niveau de service définit la manière dont est contrôlé le niveau de service, les outils et procédures d'audit utilisés ainsi que les parties tierces chargées de l'audit. La supervision et la gestion du niveau de service est un des domaines de recherche les plus abordés parmi l'ensemble des problématiques soulevées par les accords de niveau de service.

III.2.3.1 Supervision et gestionnaire de niveau de service

La supervision du niveau de service (SLM pour *Service Level Management*), consiste à mesurer la qualité du service rendu en regard de l'accord négocié par les parties. Le SLM fait plus largement référence à l'ensemble des procédures et outils d'audits chargés de surveiller le respect des objectifs de niveau de service (SLOs) et de faire appliquer l'accord. Si des clauses ne sont pas respectées, le SLM peut, à travers un gestionnaire de niveau de service (*Service Level Manager*), appliquer des politiques stipulées dans l'accord.

Afin de conserver une certaine impartialité dans cette tâche d'arbitrage, la supervision et les tâches du gestionnaire de niveau de service sont généralement déléguées à des parties tierces, appelées auditeurs tiers ou certifieurs de service.

Un auditeur tiers est un composant chargé de surveiller le respect des accords et de reporter toute infraction. Ce composant doit être neutre ou choisi par les parties signataires concernées par le contrat pour des raisons légales [Mahler2006]. Ensuite, la manière dont il mesure la qualité du service doit être connue des parties, ceci pour éviter

les litiges. Keynote Systems, Inc. [Keynote2010] est un exemple de fournisseur de services d'audit.

Le SLM couvre un spectre assez large de problèmes, de la prise de mesures et de leurs traitements, à l'application de politiques définies dans l'accord en cas de violation d'une clause du contrat par exemple. Il a donc non seulement pour but d'évaluer et assurer le respect du contrat, mais aussi parfois de réagir en cas de non-respect.

III.2.3.2 Mesure du niveau de service

La plupart du temps, le gestionnaire n'est qu'un ensemble de mécanismes servant à détecter des infractions et à générer des rapports ayant valeur de preuves en cas de litige. Il joue en quelque sorte un rôle de contrôleur de la conformité. D'ailleurs l'acronyme SLM peut parfois signifier *Service Level Monitoring* [Hanemann2005], ce qui sous-entend un rôle plus orienté vers la surveillance.

La supervision (*monitoring* en anglais) d'un système repose sur l'utilisation de sondes (Figure 26). Les sondes remontent au gestionnaire les informations nécessaires permettant de juger de la conformité vis à vis d'un objectif de niveau de service, et donc d'un critère de qualité de service. Pour chaque métrique, une sonde adaptée est nécessaire. L'étude des sondes, de leur caractère intrusif, de leur génération fait l'objet de recherches spécifiques. Notamment dans le domaine de l'informatique autonomique [Kephart2003] où les sondes sont utilisées par la partie *supervision* constituant avec l'*analyse*, la *planification*, l'*exécution* et la *connaissance*, la boucle autonomique que l'on retrouve dans la plupart des gestionnaires autonomiques. Cette boucle est représentée sur la Figure 27.

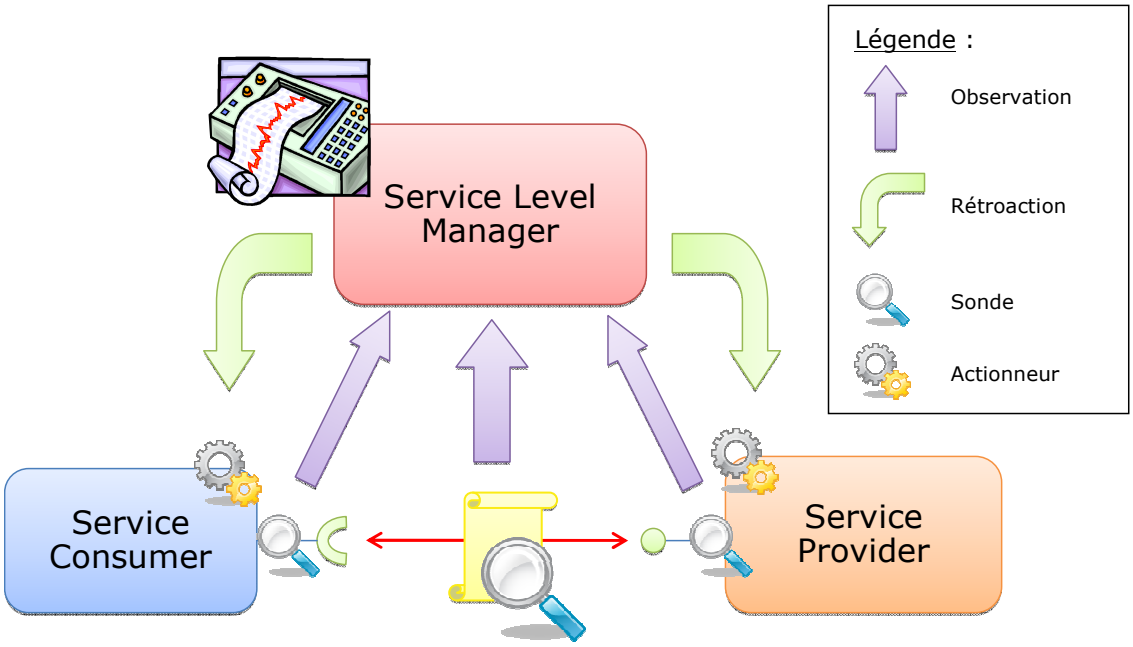

Figure 26. Gestionnaire de niveau de service et sondes

Par ailleurs, pour un même système les sondes peuvent servir à la fois au gestionnaire de niveau de service et à un gestionnaire autonomique. Ce dernier est alors chargé de rendre le système auto-reconfigurable et auto-réparable afin de rester le plus possible dans le cadre fixé par les SLOs, et ainsi minimiser les pertes dues à un non-respect de l'accord [Zhang2004].

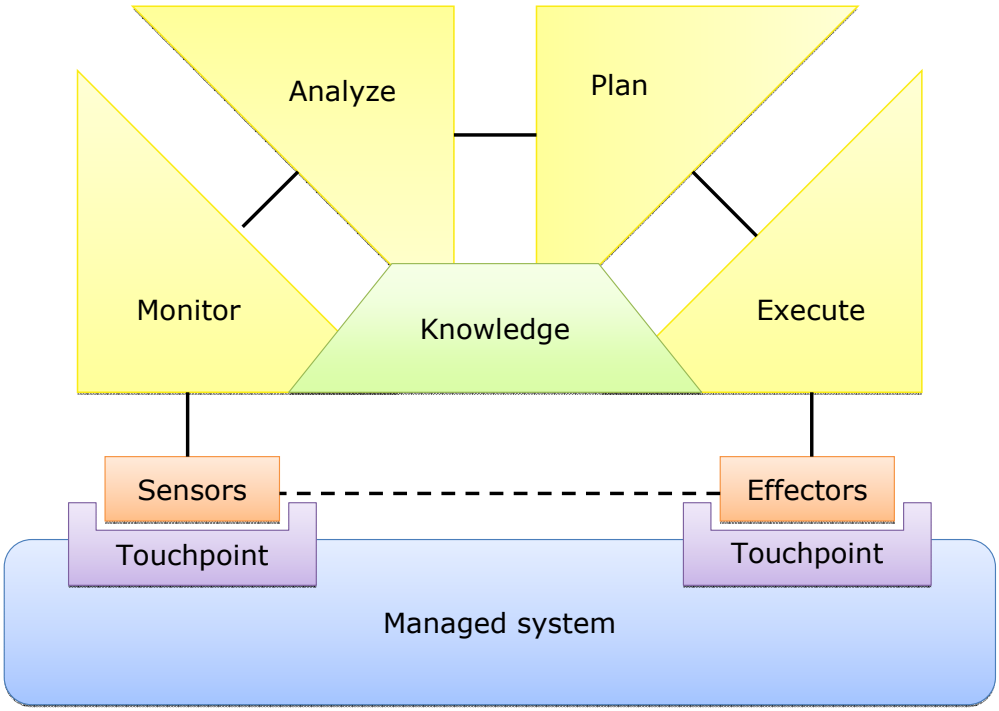

Figure 27. Boucle autonomique MAPE-K

Chaque critère de qualité se mesure différemment et nécessite des sondes spécifiques afin de les évaluer, d'où l'intérêt de bien définir la métrique des critères. Par exemple, une sonde déployée sur un routeur pourrait mesurer la latence en millisecondes d'un réseau, tandis qu'une sonde déployée sur un serveur web donnera des informations sur le nombre de connexions.

Enfin comme pour tout système basé sur l'instrumentation, se pose des problèmes d'intrusion et de perturbation du comportement : les mesures peuvent influer sur les résultats. D'où la nécessité de tolérer des marges d'erreur selon la grandeur mesurée et la nature de la sonde.

III.2.3.3 Exécution des politiques et réactions

Il est en revanche plus rare de voir attribuer au gestionnaire un rôle actif comme la mise en application de politiques. En effet, la plupart des accords de niveau de service ne sont pas « informatisés », et les clauses y décrivent des actions à entreprendre « manuellement ». C'est ensuite aux parties de mener ces actions.

Dans la section III.2.1 nous avons dégagé deux types de politiques : les politiques générales concernant la mise en application de l'accord, et les politiques de recours qui concernent les actions à entreprendre dans le cas d'un non-respect d'un objectif de niveau de service.

Par rapport à la première catégorie, il est possible qu'en plus de la journalisation, un gestionnaire de niveau de service agisse par exemple sur la facturation. Ce type de politiques concerne tout ce qui n'est pas en rapport avec la violation de clause.

Mais le rôle du SLM porte plus principalement sur le traitement des politiques de recours et l'application de pénalités ou de compensations. Une politique de recours est liée à un ou plusieurs objectifs de niveau de service.

Un fournisseur de service s'engage par les accords de niveau de service à fournir un service d'une certaine qualité. Un manquement à ces obligations peut porter préjudice à l'utilisateur du service et dans ce cas être pénalisé. De la même manière, l'usager peut lui aussi avoir des obligations décrites dans les accords et donc être pénalisé s'il ne les remplit pas. La politique la plus simple à mettre en œuvre est la rupture et donc la terminaison de l'accord. Mais cette rupture peut s'accompagner d'autres actions comme par exemple la mise sur liste noire de la partie fautive pour l'éviter à l'avenir, le basculement vers un autre fournisseur du même service.

Une approche commune pour l'application de ces politiques est la définition de politiques sous forme de règles (Evénement-Condition-Action par exemple) exécutées par un moteur de règles [Paschke2005].

III.3 Etat de l'art

La majorité des ouvrages traitant des accords de niveau de service se limitent à des guides de bonnes pratiques ou à la provision de modèles pour l'étape de définition (manuelle) d'un SLA. Ce qui s'adresse plutôt aux personnes chargées d'établir des SLAs manuscrits et présente finalement peu d'intérêt du point de vue scientifique et du génie logiciel.

Concernant les publications de recherche, les deux pans les plus importants du domaine des SLAs s'intéressent respectivement à la manière de représenter un accord de manière formelle, ainsi qu'à la supervision des accords et de leur respect. Les publications portant sur d'autres aspects des accords de niveau de service sont plus rares. Certains traitent de la négociation et de la sélection, par exemple d'un point de vue des stratégies [Gimpel2003], ou du point de vue de l'appariement sémantique [Oldham2006].

Cette section commence par présenter WSLA, un travail de fond à l'initiative d'IBM sur les accords de niveau de service dans le contexte des services Web. Seront ensuite étudiés WS-Agreement, qui comme WSLA propose un formalisme de représentation ainsi qu'une spécification pour la mise en place et la supervision des accords [Bianco2008], puis RBSLA, SLAng, et Grand SLAM. La dernière partie présente diverses solutions industrielles permettant la supervision (SLM) de systèmes et des métriques classiques associées (temps de réponse, disponibilité, ...). Enfin en conclusion de cette section, une synthèse dressera un bilan de l'état de l'art des accords de niveau de service.

III.3.1 WSLA: Web Service Level Agreements

Hewlett Packard et IBM font partie des premiers groupes à avoir tenté d'appliquer les concepts des accords de niveau de service aux services Web. Nous présentons ici la spécification *Web Service Level Agreements* (WSLA) d'IBM qui a rencontré plus de succès que son homologue *Web Service Management Language* (WSML) de HP [Sahai2001].

Le langage WSLA est un langage de description d'accords de niveau de service basé sur une syntaxe XML. La spécification WSLA [Ludwig et al 2003] intègre les différents concepts liés aux accords de niveau de service (voir Figure 28). Un accord de type WSLA est divisé en 3 parties:

- les parties engagées dans l'accord qui peuvent être les parties signataires comme les parties tierces chargées de l'audit;
- la description du service. Elle contient les opérations fournies par le service, le protocole de transmission des messages, les paramètres d'accord de niveau de service, c'est-à-dire les critères de qualité de service ainsi que les métriques associées à ces critères. Le terme *métrique* fait ici en plus référence aux fonctions et algorithmes nécessaires aux mesures;
- et les obligations qui concernent les garanties données par le fournisseur et les contraintes qui lui sont imposées. De plus cette partie spécifie une date de validité de l'accord, des formules permettant de détecter une violation de contrat et les actions à entreprendre en cas de violation, ce qui correspond aux politiques de recours.

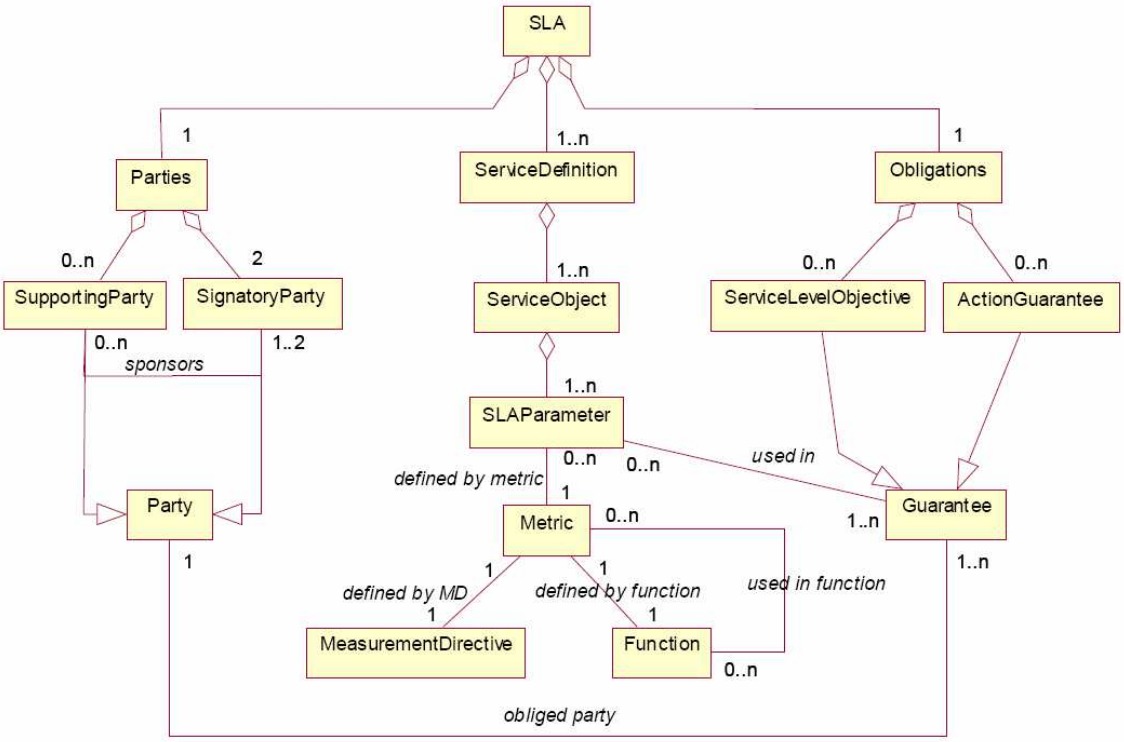

Figure 28. Représentation UML des éléments de WSLA [Ludwig et al 2003]

La définition ainsi que la négociation des accords WSLA se font avant le déploiement. La manière dont se déroulent ces activités ne sont pas couvertes par la spécification WSLA. Généralement le fournisseur de service rédige une offre d'accord et le consommateur peut accepter ou refuser cette offre.

Ces accords sont par la suite utilisés lors de l'exécution par un outil de supervision chargé de surveiller leur respect, le SLA *Compliance Monitor* [Dan et al 2004].

IBM a proposé le canevas WSLA pour la prise en charge des accords de niveau de service dans le contexte des services Web avec comme objectif de pouvoir répondre aux besoins suivants [Keller2002]:

- pouvoir s'adapter à une large gamme d'accords de niveau de service,
- fournir des mécanismes automatiques de négociation,
- chaque partie impliquée ne doit recevoir parmi les accords de niveau de service que le fragment qui la concerne. Par exemple un auditeur tiers chargé de surveiller uniquement le délai de réponse n'a pas besoin de connaître les garanties de bande passante,
- déléguer la surveillance et l'instrumentation à des parties tierces.

La Figure 29 illustre le fonctionnement du canevas et du *SLA Compliance Monitor* qui tient le rôle du gestionnaire de niveau de service (SLM). Une fois que les accords de niveau de service ont été conclus et exprimés dans le formalisme WSLA, ils sont déployés, c'est-à-dire distribués entièrement ou partiellement aux différentes parties par le *SLA*

Compliance Monitor. Ce composant est ensuite responsable des mesures et de la surveillance du respect des obligations. Si le service d'évaluation de condition détermine qu'une obligation n'a pas été respectée, des actions prévues dans les accords doivent être entreprises. L'entité métier (*Business Entity*) contient des informations privées propres au fournisseur de service telles que des objectifs et des politiques, et n'est là que pour valider ou invalider les actions si elles sont ou non en accord avec les objectifs actuels du fournisseur.

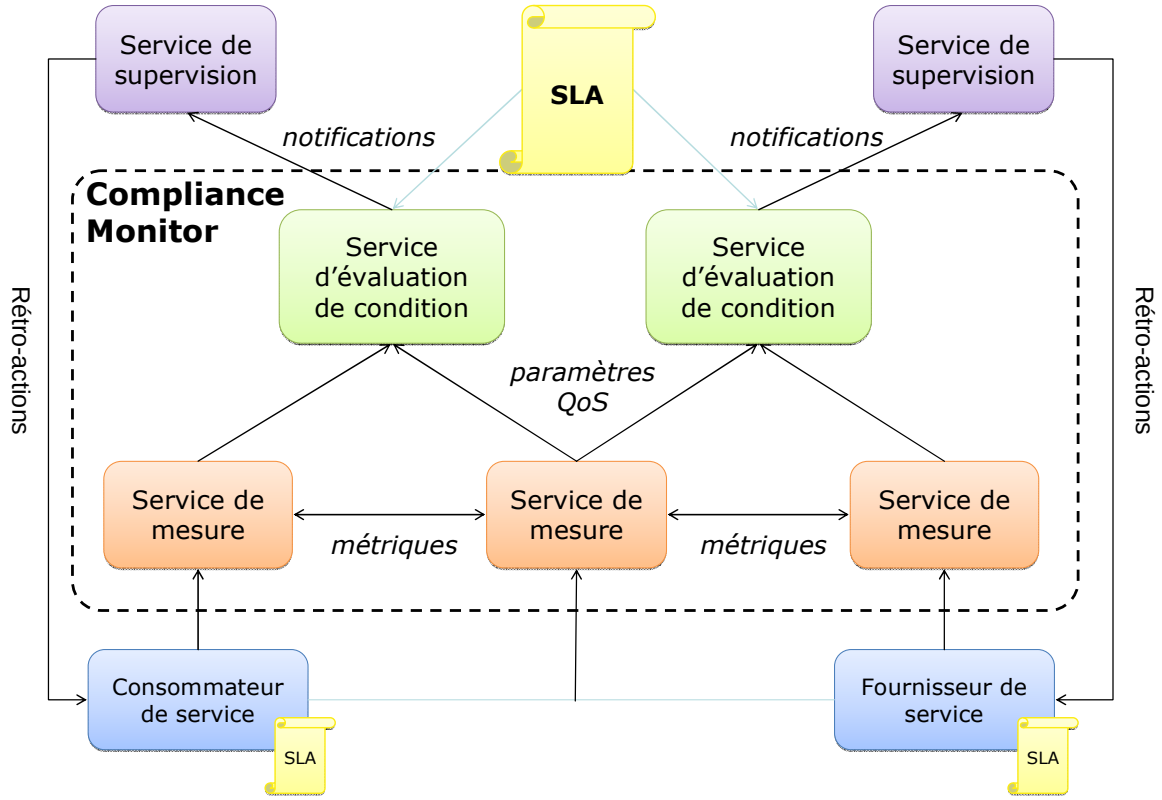

Figure 29. Schema d'interaction du compliance monitor pour WSLA

Nous pouvons par ailleurs noter que le modèle de supervision proposé ici est similaire à celui de la boucle autonomique MAPE (*Monitor, Analyze, Plan & Execute*) promue par IBM [Kephart2003] et présentée section III.2.3.2.

IBM définit avec la spécification WSLA un langage formel extensible, non restreint à un domaine, pour la définition d'accords de niveau de service. Un canevas pour aider la création de ces accords et contenant un outil de supervision du respect des accords WSLA est également proposé. Ce canevas est livré avec l'ETTK (*Emerging Technologies ToolKit*) d'IBM (depuis Avril 2003) et a également été intégré dans les suites de produits WebSphere et Tivoli.

III.3.2 WS-Agreement

WS-Agreement [Andrieux et al 2004] est une spécification établie par l'Open Grid Forum [OGF2008] qui se positionne comme un standard émergent succédant à WSLA. En effet, Ludwig et Dan qui ont activement contribué à WSLA font partie des membres ayant établi la spécification WS-Agreement. Bien que les grilles de calcul représentent le domaine principal ciblé à l'origine par la spécification, WS-Agreement s'adresse principalement aux services Web mais aussi à divers types de services en général.

La spécification WS-Agreement définit un protocole pour l'établissement des accords entre fournisseur et consommateur d'un service Web en utilisant un langage XML extensible, ainsi que des modèles d'accords (*templates*) pour faciliter la découverte de parties compatibles.

WS-Agreement offre un langage formel riche pour l'expression des garanties et des besoins des services Web, ce qui permet de capturer et représenter la nature complexe des accords de niveau de service du monde réel. La Figure 30 représente les éléments constitutifs d'un accord.

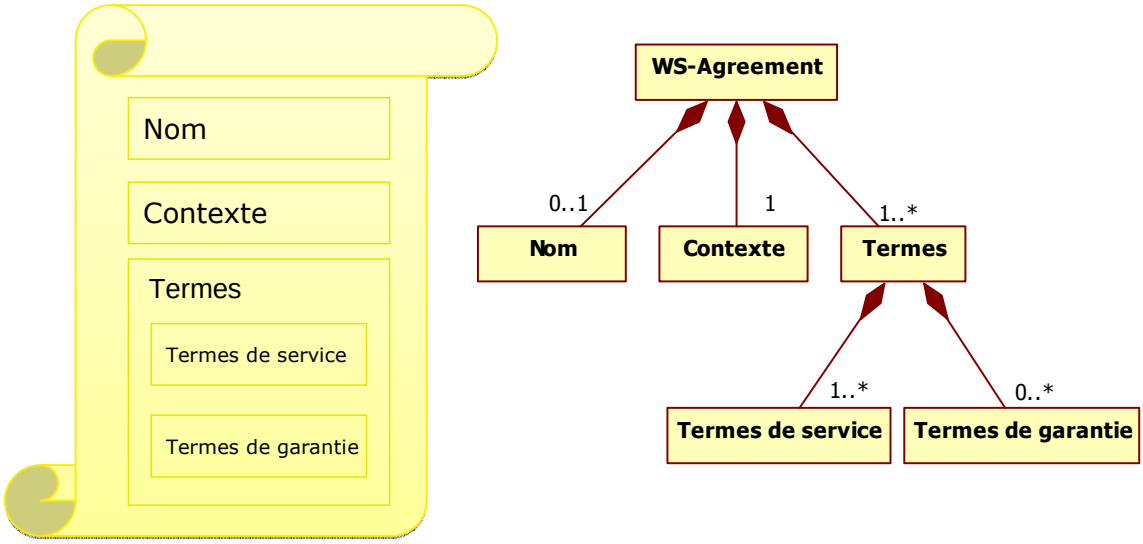

Figure 30. Contenu d'un accord WS-Agreement

Après le nom (optionnel) de l'accord, vient son contexte, qui contient les méta-données pour l'accord entier. Le contexte sert principalement à identifier les parties engagées dans l'accord et à préciser sa durée de vie. Cette section peut aussi contenir des informations sur le modèle ayant servi à la création de l'accord. Dans l'élément relatif aux parties, l'accord précise, en plus du prestataire et du consommateur, qui sont « l'*initiateur* » et le « *répondeur* » de l'accord, ces derniers pouvant être l'un ou l'autre.

La troisième section constitue le corps de l'accord, c'est-à-dire les termes. WS-Agreement différencie deux types de termes : les termes de service et les termes de garantie.

Les termes de service décrivent les services concernés par l'accord, donnent une référence vers le ou les services (l'adresse d'un service Web ou les en-têtes de paquets pour la qualité de service sur un réseau, par exemple), et précisent les propriétés attachées à un service. Ces propriétés désignent les critères de qualité de service qui seront réutilisés dans les objectifs de niveau de service. Chaque propriété définit un ensemble de variables qui la qualifient. Ces variables sont préfixées par un espace de nommage unique qui permet aux SLOs d'y faire référence de manière non ambigüe.

Un accord WS-Agreement peut contenir zéro ou plusieurs termes de garantie où chaque terme est composé des éléments suivants :

- La **partie obligée** à laquelle s'applique ce terme
- La **portée de la clause**: une liste de services auxquels s'applique cette garantie.
- L'**objectif de niveau de service** (SLO) exprimé en fonction des descriptions de service.
- Une **condition qualifiante** optionnelle qui doit être satisfaite pour que la garantie soit valable. Par exemple un consommateur peut exiger que tous les temps de réponse d'un service soient inférieurs à 5 secondes. Cependant, en conditions réelles, cet objectif ne peut être garanti en toutes circonstances. Par exemple, un service pourrait être seulement capable de traiter une tâche en moins de 5 secondes s'il reçoit moins de 1000 requêtes à ce moment. WS-Agreement permet d'exprimer et d'associer ce type de conditions à des SLOs.
- Une liste de **valeurs métier** associées à chaque SLO. Cette liste représente la valeur qu'attachent les parties à un objectif. Une valeur métier peut être exprimée à travers les notions d'*importance*, de *pénalité* si l'objectif n'est pas atteint (qui exprime indirectement l'importance, car plus la pénalité est élevée, plus l'objectif est important) et *récompense* pour la partie obligée si l'objectif est atteint, ou bien de *préférence*. Dans WS-Agreement, l'élément préférence est une liste de valeurs métier correspondant à différentes alternatives, où satisfaire chaque alternative renvoie à différentes valeurs métier.

Pour créer un accord, un client doit faire une offre à une fabrique d'accords. Une offre de création a la même structure qu'un accord WS-Agreement. La fabrique d'accords publie les types d'offres qu'elle peut accepter grâce aux modèles d'accords, appelés *templates* dans la spécification.

La structure d'un modèle d'accord est la même que celle d'un accord WS-Agreement à la différence près qu'un modèle peut aussi contenir des contraintes de création de manière optionnelle. C'est-à-dire, une section exprimant des contraintes sur les valeurs possibles des termes de l'accord pour que ce dernier puisse être créé. Les contraintes spécifient les intervalles de validité ou des valeurs exactes que peuvent prendre les termes. Les contraintes font référence aux termes auxquels elles s'appliquent en utilisant XPATH.

Ceci dit la spécification ne définit pas de protocole de négociation, activité qu'elle classe dans ses « non-buts ». Par contre elle donne accès à un ensemble d'opérations et de modèles permettant de définir ses propres protocoles de négociation.

Du point de vue des mécanismes de mise en œuvre, la spécification WS-Agreement se découpe en trois parties pouvant être utilisées ensemble ou indépendamment :

- un protocole d'établissement des accords,

- un protocole de création des modèles d'accords,
- et un ensemble de types de ports et d'opérations pour gérer le cycle de vie de l'accord : création, expiration, et supervision des états de l'accord. Ces opérations sont basées sur un ensemble de standards issus du *Web Services Resource Framework* (WSRF).

Il existe un canevas, mettant en œuvre ces protocoles. Cremona [Ludwig2004] permet la création et la supervision d'accords WS-Agreement.

Pour que les fournisseurs bénéficient d'une infrastructure pour la gestion des modèles d'accord, pour l'implémentation des interfaces, la vérification de la disponibilité du service et pour exposer l'état de l'accord à l'exécution; et pour que les demandeurs d'accord disposent aussi d'une infrastructure capable d'interpréter les modèles, de les remplir afin d'établir des accords, et de superviser l'état de l'accord à l'exécution, les auteurs proposent une implémentation de référence des mécanismes de WS-Agreement. Cremona (*Creation and Monitoring of Agreements*) offre une architecture d'intergiciel mettant en œuvre les concepts de WS-Agreement. De plus, la bibliothèque Java Cremona implémente les interfaces définies dans WS-Agreement, fournit des fonctionnalités d'administration pour les modèles et instances d'accord, et met à disposition quelques abstractions de systèmes fournissant des services pouvant être implémentés pour un domaine spécifique.

Pour conclure, WS-Agreement a un potentiel d'expressivité plus élevé que WSLA. Par exemple, en plus des éléments classiques tels que les objectifs de niveau de service (SLO) ou obligations, un accord peut contenir des domaines de validité (*scopes*) et des conditions en dehors desquels les SLOs ne peuvent être garantis, ainsi que des politiques de recours sous la forme de pénalités et compensations.

Chaque accord peut en outre définir plusieurs alternatives d'ensemble de garanties afin d'enrichir le choix des consommateurs et permettre d'automatiser ce choix. La sélection manuelle d'un prestataire adéquat étant une opération complexe, consommatrice de temps et sujette aux erreurs.

WS-Agreement introduit en plus une symétrie qui n'était pas présente dans WSLA puisque les fournisseurs peuvent exprimer leurs besoins et non pas seulement les garanties qu'ils offrent. De plus, l'initiateur à l'origine d'une offre d'accord peut être le fournisseur d'un service tout comme son consommateur.

Cependant, contrairement à WSLA, WS-Agreement n'indique rien quant aux parties tierces chargées de la supervision du respect des termes.

WS-Agreement étant un langage extensible, des travaux ont cherché à l'étendre afin d'aborder d'autres problématiques. Ainsi, une extension permet d'anticiper les violations d'un SLO et de proposer des renégociations avant que l'objectif ne soit effectivement raté [Aiello2005].

SWAPS [Oldham2006] (pour *Semantic WS-Agreement Partner Selection*) est une autre extension de WS-Agreement pour l'appariement de consommateurs et fournisseurs de service. L'approche utilise les technologies du Web sémantique dont des ontologies multiples pour gagner à la fois en précision et en flexibilité dans l'expression des accords.

III.3.3 Rule-Based SLA

Des recherches menées par Adrian Paschke à l'université technique de Munich ont conduit à un formalisme d'expression des accords de niveau de service à partir de règles : RBSLA pour *Rule-based Service Level Agreements* et à un canevas pour la représentation et l'application d'accords de niveau de service utilisant l'approche RBSLA, ContractLog [Paschke2005].

RBSLA se positionne dans le domaine de l'externalisation des infrastructures informatiques et des prestataires de services IT [Paschke2008]. Ces derniers doivent faire face à une multiplication des accords à maintenir, compte tenu de leurs différents clients qui souscrivent à différents types de service, et du succès grandissant des architectures orientées service.

Ces travaux proposent une représentation des accords de niveau de service à partir de règles en utilisant des concepts de représentation de connaissances sophistiqués, s'appuyant sur diverses logiques. RBSLA offre une montée en abstraction et propose une alternative aux contrats écrits en langage naturel, ou dans des langages tels que Java ou C++ et dont la gestion se retrouve disséminée dans le code applicatif. En combinant des formalismes logiques dans le canevas ContractLog, il est possible de décrire des spécifications de contrat à base de règles formelles qui peuvent être surveillées et exécutées de manière automatique par des moteurs de règles.

La Figure 31 illustre l'implémentation à trois niveaux qui a été réalisée : le canevas ContractLog, le langage déclaratif RBSLA et un outil de gestion de niveau de service basé sur des règles, RBSLM. ContractLog est implémenté sur Mandarax [Mandarax2007], une librairie Java qui s'appuie sur Prova, un moteur de règles ouvert pour le web sémantique. Les règles RBSLA sont une extension à RuleML [RuleML2010] et s'expriment donc dans un langage XML conçu pour satisfaire les besoins du domaine des SLAs. Ces règles sont ensuite transformées pour être traitées par ContractLog et pouvoir tirer parti de ses composants logiques de dérivation, d'*event calculus*, de logique déontique, de logique défaisable, et de logique de description.

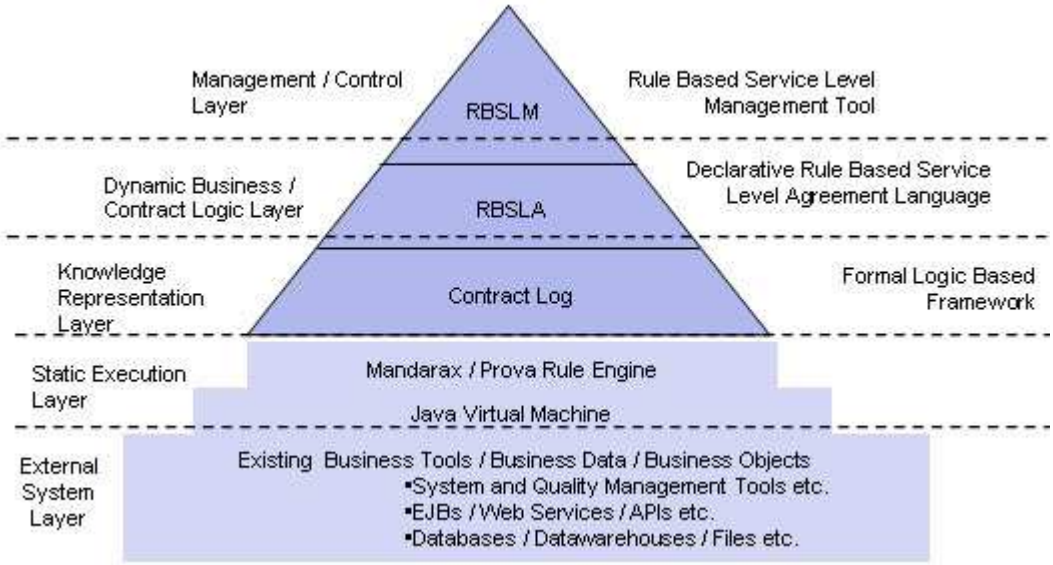

Figure 31. Architecture en couches de l'approche Rule Based SLA [Paschke2006]

Enfin, les accords RBSLA peuvent être écrits et modifiés en utilisant l'outil RBSLM qui permet la gestion, la maintenance et la supervision des règles du contrat. RBSLM utilise un modèle de calcul basé sur la couche de représentation des connaissances ContractLog et le moteur de règles ouvert Prova.

Nous ne nous étendrons pas sur le fonctionnement complexe de ContractLog qui intègre de nombreux formalismes logiques, mais nous retiendrons que les clauses des accords de niveau de service sont décrites par des règles ECA actives (Evénement, Condition, Action), c'est-à-dire qu'elles sont exécutées toutes les périodes T et sont du type *eca(T, E, C, A)*. A l'instant T, si un événement E est reçu et que la condition C est vérifiée, alors l'action A est exécutée. Par exemple, chaque minute le système vérifie si un événement « service indisponible » a été reçu. Dans ce cas, si aucune maintenance n'est prévue, un message d'avertissement est envoyé à un administrateur.

Les principaux avantages de cette approche logique, par contraste avec les approches traditionnelles par programmation, sont son degré d'expressivité élevé, son extensibilité dynamique et un fort potentiel pour ce qui concerne l'automatisation de tâches telles que la détection de violation de contrat, le contrôle d'autorisation (par la logique déontique), ou la détection de conflits.

III.3.4 SLAng

SLAng [Lamanna2003, Skene2009] est un formalisme de représentation d'accords de niveau de service. Le formalisme SLAng s'intéresse à la garantie de la qualité de service bout-à-bout, entre organisations, grâce aux accords de niveau de service entre services de réseau, de stockage, et de l'intergiciel. Ce langage a pour objectif de fournir un format pour la négociation de propriétés de qualité de service, les moyens de capturer ces propriétés de manière non ambigüe et de les intégrer dans des accords, ainsi qu'un langage interprétable qui permet des traitements automatiques.

Dans son approche, SLAng considère deux catégories d'accords de niveau de service: les SLAs horizontaux, passés entre parties de même niveau qui fournissent un service du même type, et les SLAs verticaux passés entre infrastructures de niveaux différents. Par exemple, un SLA vertical est établi entre un conteneur et un composant, ou bien entre un conteneur et la couche réseau, alors qu'un SLA horizontal concerne les interactions entre fournisseurs de services Web, entre conteneurs ou entre fournisseurs de réseau.

Un des avantages de SLAng est l'expression des accords de niveau de service de manière assez concise, et dans un langage XML, ce qui permet une indépendance vis-à-vis de l'architecture orientée service sous-jacente et offre une compatibilité élevée avec les technologies internet. Toutefois SLAng se propose comme un complément aux services Web et reste donc très lié à cette plateforme. De plus SLAng n'est pas un langage extensible, et sa sémantique n'est définie que de manière informelle. Les termes des différents SLAs verticaux et horizontaux ne peuvent donc que porter sur des critères prédéfinis, dont la sémantique est sujette à différentes interprétations.

Enfin SLAng ne se soucie que de la représentation de l'accord de niveau de service, et aucun outil de supervision de ces accords n'était proposé jusqu'au développement récent de SLAngMon au sein de la plate-forme PLASTIC [Bertolino et al 2009]. Toutefois SLAngMon n'est qu'un outil (greffon pour l'IDE Eclipse) permettant la génération de sondes AXIS pour surveiller les propriétés extra-fonctionnelles définies dans un accord au format SLAng. Il n'intervient donc pas sur la négociation et ne permet pas l'application de politiques. Certains événements peuvent néanmoins être journalisés et servir de preuve en cas de litige.

III.3.5 Grand SLAM

Grand SLAM est une initiative récente faisant partie de la plate-forme TECЛA [TECЛA2009] promue par l'université technique de Dresden.

L'outil de supervision de SLAs Grand SLAM [Spillner2009] se positionne dans le contexte des places de marché de services Web à l'échelle d'internet. Cet outil d'audit supporte les accords de niveau de service exprimés dans le formalisme WS-Agreement.

L'architecture de Grand SLAM s'appuie sur Java EE et OSGi. OSGi n'est cependant pas utilisée en tant que plate-forme à services, mais pour la gestion de la modularité et du déploiement à chaud : à l'exécution de nouveaux bundles de mesure peuvent être déployés si un accord nécessite la mesure de critères de QoS additionnels. Grand SLAM propose en outre, un service Web permettant l'enregistrement et le désenregistrement des accords à observer ainsi qu'un accès aux mesures relevées par l'outil.

Grand SLAM propose deux types de supervision : locale et distante. La supervision locale concerne les mesures spécifiques au serveur sur lequel est déployé l'outil, comme l'utilisation du CPU, tandis que la distante est spécifique aux services Web surveillés et cible des métriques telles que la disponibilité et le temps de réponse.

Enfin, une interface de requêtes SOAP appelée MaaS (*Monitoring as a Service*) permet d'accéder aux données mesurées et stockées dans une base de données SQL. En tant qu'exemple d'utilisation de cette interface, l'implémentation actuelle de Grand SLAM offre une interface graphique, le « *cockpit* », permettant de visualiser les mesures de qualité de service sous forme de graphiques SVG.

Malheureusement ces travaux ne présentent pas de cas concret de validation et la contribution consiste principalement en l'apport d'une plate-forme extensible pour la mesure et l'agrégation de critères de qualité de service définis dans des SLAs. A terme les auteurs prévoient un mécanisme de génération de modèles de prévision à partir des données collectées, ainsi qu'une collaboration hiérarchique entre différentes plates-formes Grand SLAM distribuées grâce à un intergiciel orienté messages.

III.3.6 Autres travaux

Il existe d'autres solutions industrielles, commerciales ou libres, qui proposent des outils pour la gestion des accords de niveau de service, mais ces dernières n'ont pas été jugées suffisamment pertinentes pour cette étude du point de vue de l'apport scientifique.

Par exemple la contribution apportée par l'ITIL [ITIL2008] est rudimentaire puisqu'elle ne constitue qu'un ensemble de recommandations pour les accords de niveau de service manuscrits. L'ITIL propose une boîte à outils contenant des guides de conception de SLAs et des modèles d'accords pré-remplis au format MS-Word et PowerPoint.

Outre la publication de guides de bonnes pratiques, d'aides, ou de modèles d'accords, on trouve sur le marché de nombreux outils généraux de supervision de systèmes. Ces outils n'exploitent pas de formalisme standard. Les métriques, les critères supervisés et la logique de mesure se trouvent directement dans le code de l'outil. L'utilisateur de ce type d'outil doit le configurer manuellement à partir de l'accord de niveau de service établi au préalable, ce dernier n'étant pas exprimé dans un formalisme interprétable par la machine. Par exemple NMS (*Nimsoft Monitoring System*, ex-NimBUS) de NimSoft [NimSoft2009] est un produit commercial capable de mesurer la disponibilité ou le temps de réponse d'un serveur ou d'un lien réseau, et de générer des rapports sur cette disponibilité au cours des jours/mois/années. Son pendant libre, Nagios [Nagios2009], est un outil plus général de supervision de système pour lequel NagioSLA, un greffon de supervision des accords de niveau de service, est proposé.

D'autres solutions commerciales sont proposées par les grandes industries spécialisées dans les technologies de l'information et la gestion des infrastructures : IBM Tivoli [Tivoli2008], HP OpenView, CA Unicenter, BMC Performance Manager (ex-Patrol) ou BMC Service Level Management qui suit les recommandations de l'ITIL pour la gestion des incidents [BMC2010], et le Microsoft Application Center store pour ne citer qu'eux. Ces solutions ciblent des critères généraux de qualité de service tels que la disponibilité ou le temps de réponse, qui sont intégrés en tant que paramètres dans le code applicatif ou la couche base de données.

III.4 Synthèse

Le domaine des accords de niveau de service est étudié depuis que la nécessité de différencier la qualité de service dans les réseaux informatiques et de télécommunication s'est faite ressentir. Néanmoins l'automatisation du traitement des accords, notamment dans les domaines des architectures orientées service d'une part et des grilles informatiques avec la montée de l'informatique en nuage et des XaaS [5] d'autre part, est plus récente et soulève encore des problèmes de recherches.

Les accords de niveau de service fournissent un cadre de définition des droits et devoirs des parties interagissant autour d'un service. Ils permettent d'assurer des garanties aux parties, et dans le cas où ces garanties ne seraient pas respectées, l'accord peut définir des politiques de recours. Plusieurs activités gravitent autour des SLA : la définition, la négociation, l'appariement des parties, la supervision de l'accord, …

Cette section dresse un bilan des pratiques et technologies présentées dans la section précédente, et conclut ce chapitre en présentant des pistes de recherche autour des accords de niveau de service.

III.4.1 Bilan

Le Tableau 6 présenté ci-dessous récapitule les principales caractéristiques des technologies étudiées. Six critères ont été retenus pour comparer leurs capacités : le formalisme proposé, l'attachement à une technologie particulière, le support et le niveau de négociation, et du point de vue de la gestion des accords, les outils de supervision et les politiques de recours.

Les contributions majeures dans le domaine du SLA appliqué au SOC, telles que WSLA ou WS–Agreement, ciblent essentiellement les services Web à l'instar de la plupart des travaux (Web Service Offering Language (WSOL) [Tosic et al 2003], Web Service Management Language (WSML) [Sahai2001], …). Néanmoins, quelques contributions s'intéressent à d'autres plates-formes. C'est le cas de Rio qui n'est pas présenté ici. Rio est une architecture dynamique pour le déploiement et la gestion de systèmes distribués et des accords de niveau de service au dessus de Jini [Rio2010].

Ce tableau montre que WS-Agreement est un des langages de représentation des SLAs les plus aboutis à ce jour et semble avoir supplanté WSLA, comme en témoignent les articles présentant des extensions à WS-Agreement. De plus WS-Agreement propose un mécanisme de négociation et d'obligations qui place le consommateur et le fournisseur de service au même niveau, chaque partie pouvant avoir des obligations. Enfin bien que Grand SLAM soit l'initiative la plus récente, elle reprend le formalisme de WS-Agreement et ne propose pas de spécification aussi complète. De plus la plate-forme GrandSLAM utilise OSGi uniquement pour la livraison dynamique de sondes et les accords de niveau de service ciblent seulement les services Web.

[5] Rappel : XaaS (*X as a Service*) est un terme générique pour désigner une infrastructure, une plate-forme, ou une application vus comme un service.

		WSLA	WS-Agreement	RBSLA	SLAng	Grand SLAM
Services ciblés		Services Web	Services Web et autres	Non spécifié	Non spécifié	Services Web
Formalisme Langage		XML Extensible	XML Extensible	Règles au format XML	XML Non-extensible	WS-Agreement
Etablissement et Négociation		Non spécifiés	Interfaces de création à partir d'offres Rôles d'initiateur et répondeur Aucun protocole spécifié	N/A	N/A	N/A
Outils de supervision	**SLM**	**Compliance Monitor**	**Cremona** (canevas et librairie extensibles)	**RBSLM**	**SLAngMon**	**Grand SLAM**
	Evaluation de la conformité	Les parties tierces doivent implémenter des services de mesure et d'évaluation spécifiés dans l'accord.	Un *Compliance Monitor* évalue la conformité à partir d'une instance de SLA	Evalu-ation de règles T,E,C,A	Génération auto-matique de moniteurs AXIS	Extensible par l'ajout de bundles de mesure
	Politiques de recours	WSLA définit des *garanties d'action* : des notifications ou des opérations de gestion en cas de violation de clause	Définies par des *valeurs métier*. Actions entreprises par un *Compliance Manager* spécifique au système	Basées sur des règles logiques	Simple journa-lisation d'évé-nements	Des bundles applicatifs (ad-hoc) peuvent réagir selon le résultat des mesures

Tableau 6. Comparaison de WSLA, WS-Ag., RBSLA, SLAng et Grand SLAM

III.4.2 Pistes de recherche

Le bilan précédent ne couvre pas tous les aspects liés aux accords de niveau de service. Il subsiste encore des zones d'ombre et des questions de recherche restent ouvertes. Notamment sur la composition de SLAs, la négociation, les activités annexes telles que la facturation, l'application des SLAs à un domaine émergent, et enfin sur le formalisme, où malgré les efforts il reste des verrous à lever.

Si la **composition** de services du point de vue de leurs fonctionnalités est étudiée, le résultat d'une composition de services sur leurs propriétés extra-fonctionnelles telles que la sécurité ou la qualité de service reste non-maitrisé. Ainsi la question de la composition de service pose celle de la composition d'accords de niveau de service. Pour reprendre l'exemple des fournisseurs d'applications hébergées (FAH), ces derniers offrent un service composite puisque pour le réaliser ils ont généralement besoin d'autres services (cf. section III.1.2). Cependant, pour le consommateur cette composition structurelle est invisible et seul le FAH est responsable de la bonne livraison du service. C'est donc au FAH qu'échoit la responsabilité d'adapter les SLAs qu'il contracte avec ses multiples fournisseurs et les différents recours, afin que les conséquences et le coût d'une erreur de l'un d'entre eux s'équilibrent avec les pénalités reçues vis à vis de l'utilisateur du FAH.

Par contre si on considère une composition fonctionnelle de type orchestration, le client faisant appel à l'orchestration, donc l'utilisateur du processus, doit contractualiser avec chaque fournisseur de service utilisé dans l'orchestration.

Ensuite, comme le montre le Tableau 6, **les protocoles de négociation** sont rarement définis par des spécifications d'accords de niveau de service. Seul WS-Agreement propose une base solide pour la négociation, mais la partie à qui s'adressent les offres ne peut que répondre et compléter le modèle d'accord, pas poser de nouvelles conditions non définies dans le modèle. Loin des processus poussés proches de la négociation entre humains, la négociation automatisée se résume la plupart du temps à de simples protocoles d'acceptation ou de rejet d'offres (niveau sélection). Les protocoles de négociation plus complexes doivent alors être mis en œuvre de manière ad-hoc.

De manière générale la plupart des travaux sur la négociation proposent des modèles basés sur des algorithmes d'attribution de points, sur la définition de stratégies ou sur les fonctions d'utilité pour la sélection. Par exemple une approche nommée PANDA (pour *Policy-driven Automated Negotiation Decision-making Approach*) [Gimpel2003] permet d'automatiser la prise de décision et de spécifier des politiques et stratégies pour la négociation, en combinant des règles et des fonctions d'utilité.

En outre la négociation peut soulever des questions de confidentialité. Un contexte multi-organisations est propice à la concurrence, c'est pourquoi les stratégies ainsi que les facteurs influençant la négociation (leur poids pour un système par pondération par exemple) doivent parfois être cachés à la concurrence ou aux partenaires. Il en va de même pour les plans d'action d'agents chargés de la négociation dans un contexte autonomique [Sierra1997].

Avec l'automatisation de la gestion des accords de niveau de service et des contrats, se pose aussi la question de la **facturation** automatique. En effet, le prix du service ainsi

que les pénalités chiffrées en cas de non respect des clauses font partie de l'accord. Ils rentrent ainsi dans le processus de sélection et de négociation du service [Zhang2004] et il est envisageable d'établir une facturation à partir de ces informations et des résultats de la supervision de l'accord.

Bien que les accords de niveau de service et e-contrats aient pris peu à peu pied dans le monde de l'entreprise et des solutions IT bâties sur des architectures orientées service, la question reste en suspend en ce qui concerne le domaine émergent de l'**informatique diffuse**. Afin de réguler et faciliter les coopérations inter-organisations dans ce domaine, comme c'est le cas dans les systèmes industriels, il est impératif d'aboutir à un niveau de garantie équivalent malgré les différences de caractéristiques de ces domaines d'application.

Ce problème représente effectivement un obstacle majeur à l'essor de l'intelligence ambiante. La vision de l'informatique diffuse dans laquelle les fonctionnalités de dispositifs physiques sont représentées par des services utilisables par des logiciels ou des utilisateurs a été largement acceptée. Cependant pour concrétiser cette vision, il est nécessaire de pouvoir garantir la qualité des services dans cet environnement plus complexe, plus dynamique, où le contexte évolue, et où davantage de contraintes physiques entrent en jeu.

Cette problématique mènera éventuellement à un traitement des contrats de qualité de service à l'échelle du service et de l'objet physique, c'est-à-dire à un grain plus fin que celui de l'organisation.

La plupart des autres points portent sur des questions concernant le formalisme des accords de niveau de service et leur représentation.

Un de ces points de recherche porte sur la **sémantique** des accords de niveau de service [Oldham2006]. Si un opérateur humain peut comprendre ce que signifie « indisponible 1h/semaine », un agent de négociation ou un certifieur de service doivent être capables d'identifier des objectifs similaires exprimés dans des sémantiques distinctes. Par exemple, en sachant reconnaître que « indisponibilité de 60 minutes tous les sept jours » est équivalent à l'objectif ci-dessus.

La **désambiguïsation** est également un requis majeur concernant la formalisation d'un accord. Il rejoint les préoccupations sur la sémantique. Un accord désambigüisé fournit à toutes les parties le même cadre de compréhension et permet de déléguer la supervision à des parties externes. D'ailleurs la manière de prendre les **mesures** doit elle aussi être décrite de manière non-ambigüe. La mesure d'un temps de réponse ne donnera pas les mêmes résultats si elle est effectuée depuis le client, le serveur d'application ou un intermédiaire, ni si l'objectif de niveau de service est évalué par rapport au temps de réponse moyen ou à chaque mesure; pour le service global ou pour chaque opération du service.

Enfin, quelques-uns des formalismes étudiés tentent de fournir un **langage extensible** afin de pouvoir représenter et raisonner sur des accords de niveau de service quelque soit le domaine métier. Néanmoins certains langages n'offrent pas cette extensibilité et se concentrent sur quelques critères récurrents.

En effet, bien que les langages formels de représentation des SLAs aient besoin d'être extensibles afin de pouvoir être utilisés dans n'importe quel domaine, ce sont souvent les

mêmes critères qui apparaissent, et ce quelque soit le domaine. C'est le cas par exemple de la disponibilité d'un système ou de son temps de réponse. Dans la suite de cette thèse nous nous intéresserons d'ailleurs en particulier aux accords de niveau de service portant sur la disponibilité et les interruptions de service.

Deuxième Partie : Proposition

Chapitre IV Interruptions et services intermittents

“ La théorie, c’est quand on sait tout et que rien ne fonctionne. La pratique, c’est quand tout fonctionne et que personne ne sait pourquoi. Ici, nous avons réuni théorie et pratique : Rien ne fonctionne… et personne ne sait pourquoi ! ”

Albert Einstein

Dans ce chapitre nous abordons les interruptions et la disponibilité des services. Les approches utilisées habituellement pour faire face aux interruptions sont adaptées aux systèmes où le cycle de vie des composants est maîtrisé. Dans de tels systèmes quand un composant ne remplit plus ses fonctions, le problème relève de l’erreur et est généralement assimilé à une panne.

Or dans un système suivant les paradigmes de la programmation orientée services, la situation est différente. Premièrement, le cycle de vie d’un composant fournissant un service n’est en principe pas contrôlé par l’entité qui l’utilise. Un service peut s’interrompre et devenir indisponible alors qu’un consommateur souhaitait l’utiliser. Ceci est d’autant plus vrai dans le domaine de l’informatique ubiquitaire ou dans celui de l’intelligence ambiante où les services peuvent être liés à des objets et des ressources physiques. Deuxièmement, l’approche à service dynamique permet aux consommateurs de service de s’adapter aux changements de contexte et de réagir aux changements dans le cycle de vie des services utilisés, en substituant par exemple le fournisseur d’un service par un autre fournisseur du même service.

Ainsi, dans le contexte des architectures orientées service, un consommateur de service peut réagir au moment des arrivées et des départs de services. En revanche il ne sait pas quand un service deviendra indisponible suite à l’interruption d’un fournisseur, ni si et quand le fournisseur de service sera à nouveau disponible.

L’idée véhiculée par cette partie de notre contribution consiste à considérer

- l’**interruption** comme un phénomène normal de l’approche à service pouvant être transitoire, **temporaire** ;
- le **fonctionnement** « normal » d’un fournisseur de service **intermittent**, c’est-à-dire entrecoupé d’interruptions.

IV.1 Interruptions de service

L'interruption de service est un phénomène impliquant l'indisponibilité du service qui doit être pris en compte dans la construction d'architectures orientées service. Du fait de la nature des objets communicants, le contexte de l'intelligence ambiante est un domaine particulièrement propice aux interruptions. Cette section présente et classifie les principales causes d'interruption. Elle explique également en quoi les solutions existantes de traitement des interruptions ne sont pas adaptées à tous les cas de figure, en présentant plus spécifiquement les limites des politiques les plus dynamiques suivies par certains modèles à composants orientés service.

IV.1.1 Définitions

Une interruption de service correspond à une rupture dans la livraison continue d'un service. Lorsqu'un système est interrompu, il passe d'un état disponible à indisponible. Une interruption s'étale sur un intervalle de temps à durée variable, mais le terme « interruption », qui étymologiquement signifie « rupture entre » (des périodes de fonctionnement), présuppose que le système tend à reprendre son fonctionnement normal au bout d'un certain temps.

Les notions d'interruption et de disponibilité sont étroitement liées au concept de sûreté de fonctionnement (*dependability*). La sûreté d'un système désigne sa capacité à délivrer un service. Plus la sûreté est élevée, plus les utilisateurs du système peuvent s'y fier. La sûreté se définit par quatre critères : fiabilité, maintenabilité, disponibilité et sécurité [Nelson1990, Avizienis2001].

Du point de vue de l'interruption, nous nous intéressons en particulier à la disponibilité et la fiabilité qui se définissent ainsi :

" *La fiabilité correspond à la probabilité qu'un système soit fonctionnel sur un intervalle de temps donné, dans des conditions données. La fiabilité réfère donc à l'intervalle de temps pendant lequel le système s'exécute et réagit correctement sans aucune interruption. La fiabilité concerne la continuité de service.* "

" *La disponibilité correspond à la probabilité qu'un système soit fonctionnel à un instant donné. La disponibilité d'un système réfère non pas à un intervalle de temps, mais à un instant précis, où le système s'exécute et réagit correctement. On dit aussi qu'un système est disponible (respectivement, indisponible) à un instant donné s'il est (respectivement, n'est pas) fonctionnel.* "

Le domaine de recherche sur les pannes définit les métriques suivantes (voir aussi Figure 32 qui présente les notions suivantes sous forme de chronogramme) :

- MTTF (*mean time to failure*) : Temps moyen de fonctionnement entre pannes qui est aussi la durée moyenne pendant laquelle le système peut-être considéré comme fiable.

- MTTR (*mean time to repair*) : Temps moyen de réparation (qui représente la durée moyenne d'une défaillance)
- MTBF (*mean time between failure*) : Temps moyen entre deux défaillances (entre le début d'une défaillance et le début de la suivante).

On a donc la relation suivante :

$$MTBF = MTTF + MTTR$$

Ces métriques sont utilisées pour calculer la disponibilité D :

$$D = \frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF}$$

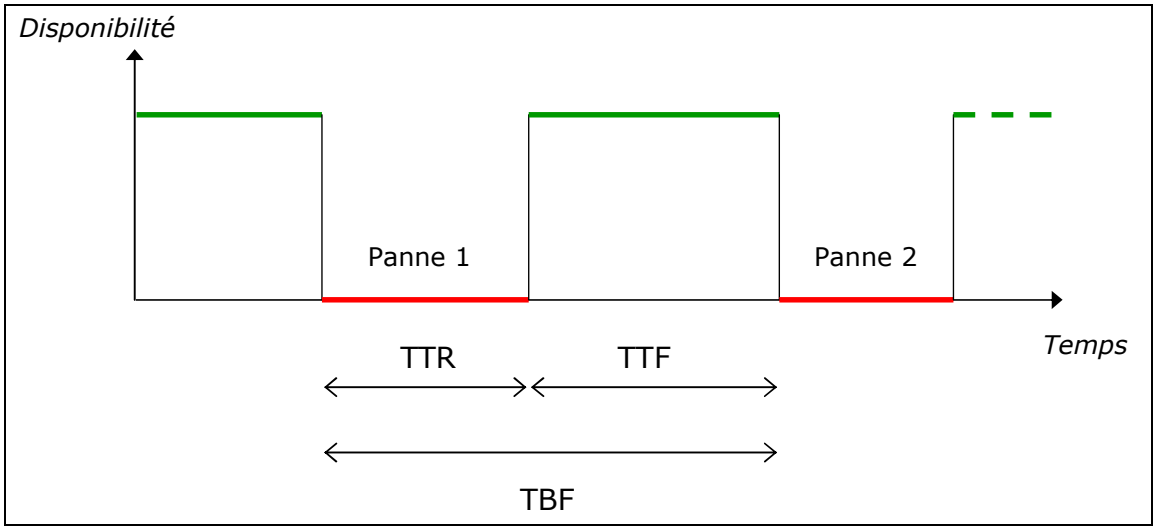

Figure 32. Notions de Time to Repair (TTR), Time to Failure (TTF) et Time between Failure (TBF)

Parfois, on parle également de haute disponibilité [Gray1991] ou de système hautement disponible pour désigner un système s'exécutant sur de longues périodes en étant très rarement interrompu. Un tel système définit sa disponibilité, c'est-à-dire le ratio du temps de fonctionnement sur le temps d'interruption, par le nombre de 9 derrière la virgule. Par exemple un système hautement disponible « à cinq neufs » garantira une disponibilité de 0.99999, ce qui équivaut à un temps d'indisponibilité de 25,9 secondes par mois.

Contrairement à la disponibilité, la fiabilité n'autorise aucune interruption pendant le laps de temps où le système est dit fiable. Or un système peut-être hautement disponible, mais parsemé d'interruptions très courtes et fréquentes. Cependant il ne pourra pas être qualifié de « fiable ».

Nous considérerons donc qu'un service est interrompu lorsque son fournisseur devient indisponible et que le service ne peut plus être rendu. La durée de l'interruption correspond alors au temps d'indisponibilité du fournisseur. Ici le terme « fournisseur » fait référence à un composant ou système fournissant un service.

IV.1.2 Causes d'interruption

Les causes d'interruption de service sont diverses. Et cette diversité est d'autant plus flagrante dans applications décrites dans l'introduction de cette thèse. Si l'on reprend le schéma de la Figure 2 présenté en introduction, plus l'informatique est diffuse, enfouie, dématérialisée et distribuée, plus les risques d'interruption d'un composant (qui sont plus nombreux et impliquent plus de points de panne) d'un système augmentent.

Le Tableau 7 ci-dessous propose une classification des causes possibles d'interruption dans les applications orientées service. Il passe en revue les différentes causes d'interruption d'un système et donc du service qu'il fournit. Afin de catégoriser les causes d'interruption, nous les avons classées selon les critères suivants :

- **Prévisibilité** : dans certains cas l'interruption peut-être signalée à l'avance ou anticipée.
- **Défaillance** : si l'interruption a un caractère accidentel et peut-être assimilée à une panne.
- Son origine, **matérielle** ou **logicielle**.

		Prévisible	Défaillance	Matériel	Logiciel
Maintenance (MàJ, entretien)		✓	✗	✓	✓
Arrêt (veille, périodes)		✓	✗	✓	✓
Energie	**Batterie déchargée**	✓	✗	✓	✗
	Coupure électrique	✗	✓	✓	✗
Perte de connectivité		✓ ✗	✓ ✗	✓	✗
Dépendance non résolue		?	?	✗	✓
Equipement endommagé ou défectueux		?	✓	✓	✗
Erreur		✗	✓	✗	✓
Malveillance (DoS, ...)		✗	✓	✓	✓

Tableau 7. Causes d'interruptions de service

Toutes les interruptions ne relèvent pas de la panne et certaines peuvent parfois même être prévues. C'est le cas des opérations de maintenance, qu'elles soient logicielles (mises à jour) ou matérielles (entretien, remplacement d'un composant/équipement). Les mises à jour représentent la principale cause d'interruption de service. Par exemple les serveurs de jeu en ligne, comme les jeux de rôles massivement multi-joueurs (MMORPG) tels que *World of Warcraft*, sont soumis à des opérations de maintenance (tous les mercredi matin pour WoW).

Un service peut aussi être simplement arrêté sans que ce soit une panne. Par exemple, un équipement peut être mis en veille pour des raisons d'autonomie ou d'économie d'énergie (Green IT) et ne fonctionner qu'à certaines périodes de la journée, ou plus simplement un service peut être livré uniquement sur certaines périodes. Dans ce cas les interruptions peuvent être anticipées puisque le fournisseur de service pourrait annoncer à l'avance ses plages de disponibilité.

Certaines interruptions sont quant à elles inhérentes au domaine de l'intelligence ambiante où le logiciel est fortement lié à des objets physiques communicants (il y est embarqué, ou bien il communique avec). Ces objets ont rarement une alimentation sur secteur et sont généralement alimentés par une source d'énergie portable (batterie) afin de gagner en mobilité. Ainsi lorsque la batterie est déchargée, le service est interrompu jusqu'à ce que la batterie soit remplacée ou rechargée (manuellement ou automatiquement –robot autonome ou panneaux solaires-), ou que l'appareil bascule sur une autre source d'énergie. En outre, quand ces dispositifs ont la capacité de suivre l'état de leur batterie, le temps de déchargement peut être estimé et l'interruption prévue.

De la même manière, dans le cas d'un appareil mobile équipé d'un moyen de communication sans fil, si cet appareil est déplacé hors de portée du récepteur (ou inversement), la communication est coupée et le service est interrompu. Lorsqu'il est possible de connaître la puissance du signal radio, l'interruption peut parfois être anticipée en inférant qu'une diminution continue de la puissance conduit à une perte du signal.

Bien que les réseaux sans fil soient plus propices aux interruptions, même dans le cas d'une connectique filaire (réseau ou alimentation), une application n'est pas à l'abri d'une coupure de courant ou de réseau, ou même du débranchement accidentel d'un câble.

Ensuite, lorsque l'interruption est la conséquence d'une panne, cette panne peut avoir une origine matérielle ou logicielle. Les cycles de vie du matériel et du logiciel sont différents. Pour du matériel, il existe un phénomène de « mortalité infantile » qui signifie qu'un équipement est moins fiable au début de sa vie. Puis la probabilité de panne décroit avec le temps jusqu'à ce que le matériel vieillisse et s'use, augmentant les risques de panne.

En revanche, le logiciel n'est pas soumis à ces phases de fragilité en début et fin de vie. Néanmoins, les logiciels produits sont de plus en plus complexes et donc plus enclins aux erreurs. En principe les erreurs logicielles sont détectées pendant la phase de développement où le logiciel est testé. Mais en pratique ces phases de test sont menées de manière plus ou moins rigoureuse, et des erreurs peuvent également échapper aux tests. De plus, lors de l'exécution d'un système des erreurs non-déterministes peuvent apparaitre. Ces bugs, appelés *Heinsenbugs* par opposition aux *Bohrbugs* déterministes, sont dus par exemple à des problèmes d'horloge interne, de synchronisation de threads, etc. Ces pannes logicielles peuvent se manifester aussi bien au niveau du fournisseur de

service qu'au niveau du système d'exploitation ou de l'infrastructure sur laquelle s'appuie un fournisseur de service.

Enfin, dans le cas des architectures orientées service, si un fournisseur de service composite dépend d'un autre service qui est interrompu, il ne pourra plus fonctionner et sera lui aussi interrompu. Selon la connaissance qu'a le composite sur ses fournisseurs de service, il pourra anticiper et signaler sa propre interruption (quand le service requis n'est pas disponible sur certaines plages horaires par exemple).

IV.1.3 Tolérance aux interruptions

Les interruptions de service et plus largement de système font partie d'un problème de recherche connu, plus large, celui des pannes qui est étudié depuis plus de 20 ans. Cette section examine dans un premier temps les solutions existantes de traitement des interruptions de manière générale, puis, dans un second temps, comment sont gérées les interruptions dans le cas spécifique de l'approche à services.

IV.1.3.1 Solutions générales

La tolérance aux pannes est un des moyens permettant d'améliorer la sûreté en offrant une meilleure résistance aux pannes (matérielles ou logicielles) qui constituent une menace pour la sûreté.

En effet, plusieurs techniques existent pour prévenir les pannes ou minimiser leur impact :

- la **prévention** de pannes consiste à contrôler la qualité en amont, pendant les phases de conception et de fabrication du matériel ou du logiciel. Les pannes physiques peuvent être prévenues grâce à des procédés de renforcement, d'isolation ou de protection face aux ondes électro-magnétiques, tandis que les pannes logicielles d'origine malicieuse peuvent être prévenues par des pares-feux, des anti-virus et autres défenses similaires. Le rajeunissement qui consiste à redémarrer de temps en temps un système pour qu'il soit plus « propre » et moins sujet aux pannes est également une forme de prévention.
- la **prévision** de pannes consiste à évaluer le comportement d'un système, à la fois de manière qualitative (analyse des événements pouvant causer des pannes) et quantitative (probabilités et degré de satisfaction ses propriétés de sûreté).
- la **correction** de pannes s'effectue à la fois pendant la phase de développement et la phase de vie du système. Durant la phase de développement les tests permettent d'éliminer des erreurs dans le logiciel ou de détecter des défauts de fabrication, tandis que les autres pannes seront corrigées plus tard, par des opérations de maintenance.
- la **tolérance** aux fautes a pour objectif de maintenir la continuité de service malgré les pannes. Généralement la tolérance aux pannes s'appuie sur des mécanismes de détection d'erreur et de restauration. La restauration consiste à

ramener le système à un état sans erreur. Ce qui signifie éliminer les erreurs détectées, et empêcher la panne identifiée de se reproduire. Deux approches principales permettent l'élimination d'erreurs : le *rollback* qui consiste à un retour à un état sans erreur précédemment sauvegardé, et le *roll-forward* qui crée un nouvel état. Quant au traitement de la panne, il s'effectue généralement en quatre étapes : diagnostique de la panne, isolation de la panne, reconfiguration du système et réinitialisation du système.

Ces techniques peuvent évidemment être utilisées conjointement pour améliorer la sûreté d'un système. Cependant certaines pannes ne sont pas identifiables ou corrigibles, et seule la technique de tolérance aux pannes permet d'y faire face.

Afin de mettre en œuvre des systèmes tolérants aux pannes, il existe deux types d'approches classiques: la reprise sur panne comme évoqué ci-dessus, et la redondance. Dans [Huang1993], cinq niveaux de tolérance aux pannes sont répertoriés. Les niveaux 0 et 4 correspondent respectivement aux systèmes n'offrant aucune tolérance aux pannes et aux systèmes capables de fonctionner en continu, sans interruption. Entre ces deux extrêmes se situent la détection automatique d'erreur suivie du redémarrage du système (niveau 1), les points de sauvegarde périodiques, la journalisation et la restauration de l'état interne (niveau 2), et la restauration de données persistantes (niveau 3). Ces trois niveaux relèvent des techniques de reprise sur panne qui consistent à récupérer d'une erreur et revenir à un état opérationnel (conservation de l'état, *checkpoints*, *retry* et *restart*, ...).

L'approche la plus commune s'appuie sur la redondance matérielle ou logicielle des parties d'un système. Tandis que la réplication matérielle est réservée aux composants critiques ou peu coûteux, la réplication logicielle est plus simple à réaliser. Trois techniques différentes existent pour rendre un système tolérant aux pannes par le principe de redondance :

- la **réplication** consiste à fournir plusieurs instances identiques du même composant auxquelles on assigne les mêmes tâches en parallèle. Un résultat jugé correct est alors choisi parmi les multiples réponses.
- la **redondance** consiste également à fournir plusieurs instances identiques du même composant, mais une seule instance traite les tâches, et cette instance est remplacée par une autre en cas de panne.
- la **diversité** s'appuie quant à elle sur différentes implémentations de la même spécification. Ces implémentations sont alors utilisées en parallèle comme les instances d'un composant répliqué afin d'éviter les erreurs liées à une implémentation spécifique.

IV.1.3.2 Solutions dans les architectures orientées service

Dans le domaine de l'approche à service, quand les interruptions ne sont pas liées à la distribution des services et à des problèmes au niveau du réseau, elles correspondent à une incapacité du prestataire à rendre son service. Cette incapacité peut provenir soit d'une panne, soit d'un arrêt volontaire du service.

Pour simplifier le problème, nous faisons l'hypothèse que dans les plates-formes dynamiques à service, c'est-à-dire dans les architectures orientées services supportant le retrait de service et la notification (des arrivées ou départs de service), les pannes peuvent être détectées et le service désenregistré en conséquence (grâce à des mécanismes de *ping*, *timeout*, *heartbeat* ou *keepalive*, ...). En effet les pannes ne peuvent pas toujours être « propres » (i.e., envoi d'un signal pour avertir le système). Par exemple, dans le cas où technicien éteint l'équipement pour une opération de maintenance on peut imaginer qu'un signal de terminaison peut-être envoyé au SOA pour désenregistrer le service correspondant. Ceci permet de rendre explicite l'interruption et d'en informer les consommateurs du service interrompu.

Avec une technologie telle que les services Web qui supporte mal le dynamisme, une interruption se traduira soit par un *timeout* ou une erreur puisque généralement le consommateur utilise directement l'adresse du service Web pour y accéder, sans passer par un registre. Il utilisera donc une adresse inactive et n'obtiendra pas de réponse du service.

Ensuite, si le consommateur de service ne s'appuie pas sur un modèle à composant pour la gestion des liaisons, les interruptions devront être gérées par le développeur qui devra penser à traiter les cas d'erreur ou à capturer des exceptions (*java.rmi.RemoteException*, *TimeOutException*, *ServiceNotFoundException*, ...) en plus d'écouter et surveiller la disponibilité dynamique des fournisseurs.

C'est pour ces raisons qu'est préconisée l'approche récente des modèles à composants orientés service qui délèguent la gestion des liaisons et donc des interruptions à un conteneur. Parfois ces modèles offrent une gestion personnalisée des liaisons comme iPOJO qui propose plusieurs politiques de liaison. La politique la plus courante, qualifiée de dynamique par opposition à une liaison statique, et que l'on retrouve dans la plupart des modèles à composants, consiste à invalider le composant consommateur de service quand une dépendance n'est plus satisfaite. A moins qu'un autre fournisseur du même service soit disponible, auquel cas il remplace le prestataire interrompu par une opération appelée substitution, et le consommateur continue son activité avec un autre prestataire.

Ces politiques d'adaptation aux interruptions sont abordées plus en détail dans le Chapitre V, partie V.1.3.

IV.1.4 Limites

Que ce soit du point de vue des solutions « classiques » ou des solutions proposées par les modèles à composants orientés service, le traitement des interruptions reste insuffisant dans le cas des services intermittents si l'on considère les besoins et contraintes des nouvelles applications.

IV.1.4.1 Des solutions de tolérance aux pannes

Les méthodes classiques de tolérance aux pannes s'appliquent à des systèmes où toute partie du système est contrôlée par la même entité et où les cycles de vie de ses

composantes sont maîtrisables, ce qui n'est pas le cas dans l'approche à services. Même si cela peut arriver, a priori le consommateur de service n'est pas propriétaire des services utilisés étant donné qu'un consommateur et ses fournisseurs sont en principe opérés par des entités indépendantes. Ainsi un consommateur de service percevant l'interruption comme une panne -puisque l'absence d'un service requis l'empêche de fonctionner-, ne pourra exploiter les techniques de reprises sur panne. En effet, le consommateur ne peut pas agir sur le fournisseur pour réparer une erreur, le réinitialiser ou modifier son état puisqu'il ne le contrôle pas. De plus l'interruption peut être normale et ne pas résulter d'une erreur logicielle ou d'une défaillance (si le service n'est rendu qu'à certains horaires par exemple).

En revanche, les techniques de réplication peuvent aider les consommateurs à tolérer les interruptions de service dans une certaine mesure. La réplication, la redondance ou la diversité sont assez simples à mettre en œuvre dans le contexte de l'approche à service étant donné qu'il suffit de disposer de, ou à défaut fournir, plusieurs composants offrant le même service. C'est une technique employée par exemple dans le gestionnaire de dépendances du modèle à composant iPOJO qui propose au consommateur d'un service d'en livrer une implémentation « par défaut », afin de palier à l'absence de fournisseur de service (suite à une interruption).

Toutefois cette approche montre ses limites dans le cas de services spécifiques dont les fonctionnalités ne peuvent être « mimées » ni rendues par des composants tiers. De même, la réplication n'est pas forcément applicable dans le cas de services liés à des objets physiques dans un contexte d'intelligence ambiante. S'il est possible de répliquer des capteurs pour relever la température d'une pièce, il n'est pas envisageable de dupliquer un téléviseur UPnP.

IV.1.4.2 Des politiques de liaison des composants orientés services

Si les solutions spécifiques à l'approche à service proposées par les modèles à composant facilitent la construction d'applications à service et permettent de réagir aux interruptions de service, elles ne conviennent néanmoins pas à toutes les situations. En effet, les approches des modèles à composant orientés service font l'hypothèse « pessimiste » que si un fournisseur devient indisponible à un moment donné, il doit être considéré comme « mort », et son retour n'est donc pas envisagé. Par conséquent, le retour d'un fournisseur ayant été interrompu sera seulement perçu comme l'arrivée d'un nouveau prestataire du service. C'est pour cela que dans le doute d'un retour probable les politiques de liaison usuelles préfèrent substituer le fournisseur ou invalider le composant consommateur. La liaison entre un consommateur et un fournisseur est donc défaite quand le fournisseur n'est plus disponible et une nouvelle liaison est créée vers un autre fournisseur s'il en existe un. Dans le cas contraire la liaison est recréée si le fournisseur initial redevient à nouveau disponible. Dans tous les cas une interruption entraîne une destruction de liaison et éventuellement la création d'une nouvelle. De plus, cette rupture de liaison, lorsqu'elle entraîne la désactivation d'un composant, peut provoquer l'arrêt d'une application.

Or cette approche n'est pas adaptée à au moins deux cas de figure. Premièrement, le service interrompu s'il est requis par le composant consommateur n'allait pas nécessairement être utilisé pendant la durée de l'interruption. Par exemple un service peut être utilisé à l'initialisation du composant et plus du tout ensuite, ou bien utilisé à des

moments précis en dehors desquels le service peut être indisponible (comme le ferait un *cron* lançant des actions planifiées). Bien que certains modèles définissent des dépendances de service « optionnelles » afin de tolérer les interruptions, si le service n'est pas présent au moment où il doit être utilisé, le problème reste entier. Deuxièmement, la durée de l'interruption d'un fournisseur de service peut-être éphémère ou simplement acceptable pour le consommateur du service. La section IV.1.2 sur les causes des interruptions de service décrit plusieurs cas de figure où l'interruption est de durée limitée. C'est le cas par exemple des mises à jour logicielles, de la maintenance matérielle (e.g., changement de batterie) ou d'un équipement mobile fournisseur de service qui sortirait temporairement de la zone de couverture du consommateur (et vice versa).

De plus, dans la plupart des modèles à composants orientés service, l'activation et la désactivation d'un composant sont traitées par deux routines de démarrage et d'arrêt. Or l'interruption d'une dépendance de service n'est pas la seule cause qui peut mener à cet état « désactivé ». Ainsi, l'arrêt de l'application, une dépendance non satisfaite, ou un *handler* externe invalide dans le cas d'iPOJO, sont traités par la même routine. En règle générale cette approche pose problème dès lors que l'interruption est éphémère : le composant va tour à tour relâcher les ressources via sa routine d'arrêt puis les réserver à nouveau via sa procédure de démarrage.

C'est pourquoi il est parfois préférable d'attendre le retour du fournisseur interrompu et de conserver la liaison, plutôt que de le remplacer ou détruire/recréer la liaison en causant un redémarrage des composants consommateurs, et donc la perte de l'état courant dans le fil d'exécution ainsi que le relâchement et la réservation successifs de ressources.

Par conséquent dans les cas où l'interruption de service est de durée « acceptable », sans incidence ou lorsqu'elle intervient hors d'une période d'utilisation du service, les politiques de liaison fortement dynamiques usuelles perdent de leur efficacité pour les raisons suivantes :

- La **perte du fil d'exécution** est un des premiers problèmes causé par la désactivation d'un composant consommateur ou par le changement de fournisseur suite à une interruption.

Bien que la plupart des définitions de la programmation orientée service préconisent qu'un service doit être sans état, en pratique ce n'est pas toujours le cas et certains services ont conscience de leur état (on parle de services *stateful*). Ainsi, lorsqu'un fournisseur est interrompu et remplacé par un autre, le nouveau fournisseur ne se trouve peut-être pas dans un état équivalent et la cohérence est perdue.

De plus, le consommateur du service peut lui aussi conserver l'état de son interaction avec un service sans état. L'état peut résulter d'une succession d'opérations. Si une interruption survient au milieu d'une suite d'actions de la part du consommateur, à moins que ce dernier ne soit pourvu de mécanisme de sauvegarde et de reprise d'état, le désactiver lui ferait perdre son état courant. Ce qui peut s'avérer plus ou moins gênant selon la durée et le coût des opérations réalisées par le consommateur et le service.

Il existe aussi des services que l'on peut qualifier de *conversationnels*. L'utilisation de ces services repose sur un protocole connu des deux parties, sur un fonctionnement en plusieurs étapes ou un ordre d'appel de méthodes à respecter. De la même manière que dans les deux cas précédents, si le consommateur est désactivé et perd son fil d'exécution, il recommencera la conversation au début, et si le fournisseur de service est substitué, alors le nouveau fournisseur ne sera pas au même point de la conversation que le

consommateur. Prenons un exemple simple. Si un service a besoin d'être initialisé par une première méthode *m1* avant d'être utilisé via les méthodes *m2* et *m3* et qu'il est interrompu alors qu'un consommateur avait appelé *m1* et *m2* et s'apprêtait à appeler *m3*, les politiques de liaisons dynamiques mènent à trois issues. Soit un autre fournisseur est disponible et le consommateur invoquera *m3* sans avoir initialisé la communication avec *m1*, soit le consommateur est désactivé jusqu'au retour du fournisseur sur lequel il appellera à nouveau *m1* et *m2*, soit le consommateur se lie à un nouveau fournisseur après avoir été désactivé et les deux entités peuvent commencer une nouvelle conversation. Cette troisième issue représente le cas favorable aux politiques fortement dynamiques.

Enfin si l'on considère une orchestration de services où une série d'activités est coordonnée sur un ou plusieurs services en suivant un processus, le même problème se pose si les liaisons sont gérées par des politiques fortement dynamiques. En cas d'interruption d'un des services, l'orchestrateur est désactivé et le flot d'exécution reprendra à son point de départ si la gestion de l'état n'est pas correctement mise en œuvre. Ce scénario est illustré par la Figure 33 sur laquelle un composant utilise séquentiellement deux services S1 et S2 qui fournissent trois méthodes, m1, m2 et m3. Si le second service est interrompu alors qu'il n'était pas encore utilisé, le consommateur est désactivé et perd les résultats de m1 et m2. Par conséquent lorsque S2 est à nouveau disponible, le composant reprend son activité depuis le début de la séquence.

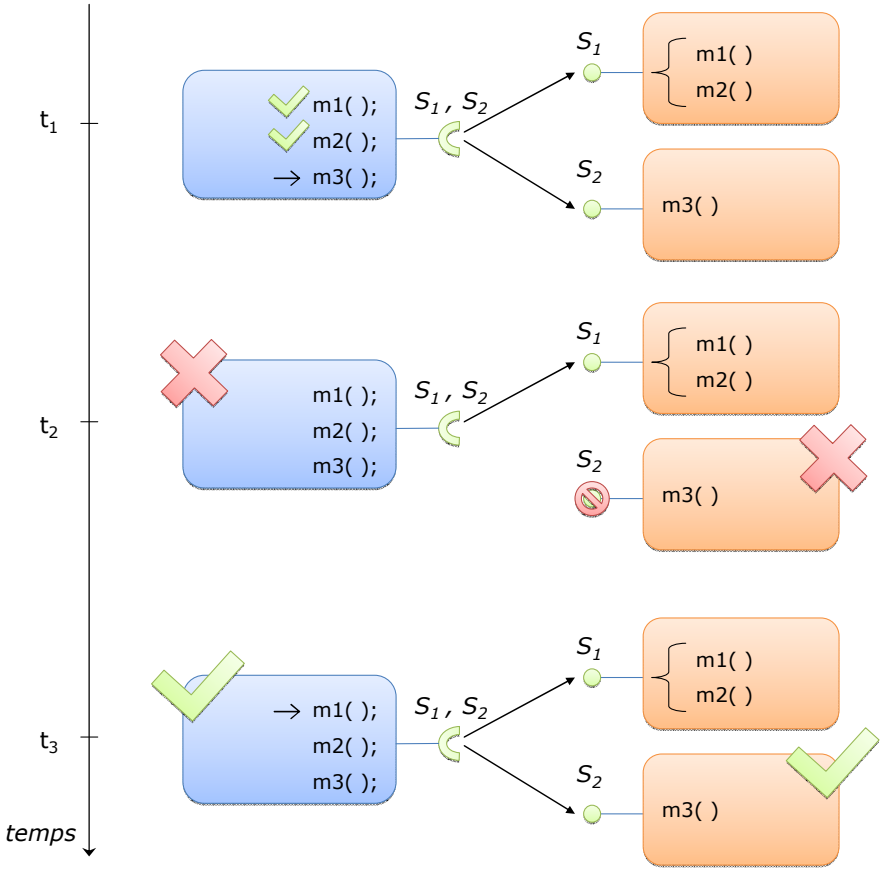

Figure 33. Perte du flot d'exécution suite à une reconfiguration dynamique

- La **perte de la liaison consommateur-fournisseur** est un autre facteur qui remet en question les approches trop dynamiques.

La destruction comme la création d'une nouvelle liaison, selon la technologie de la plate-forme à service sous-jacente, ont un coût qui peut-être évité si l'interruption est éphémère. De même si aucun autre fournisseur du service interrompu n'est disponible pour le remplacer, la désactivation et la réactivation ont elles-aussi un coût, comme expliqué dans le point précédent.

Les politiques de liaison fortement dynamiques poussent à l'extrême le principe du découplage de l'approche à services. La relation consommateur-fournisseur s'efface derrière la notion de service. A la différence des services Web où le fournisseur est souvent clairement identifié, ici l'identité du (ou des) prestataire(s) n'importe pas tant que le service est livré. Or dans une interaction définie par un contrat il peut être nécessaire de pouvoir identifier les parties. Par exemple dans le cas de services payants, si la tarification suit un modèle forfaitaire ou avec engagement, un changement de fournisseur peut entraîner une double facturation et donc des coûts superflus pour le consommateur. Par ailleurs deux prestataires n'offrent pas nécessairement la même qualité de service et il peut être plus intéressant pour un consommateur d'attendre le retour du fournisseur interrompu plutôt que de basculer vers un fournisseur de qualité inférieure ou qui correspond moins à ses attentes.

- Les politiques de liaison très dynamiques désactivant des composants et détruisant des liaisons dès que survient une interruption posent un problème d'**architectures scintillantes**.

Si les interruptions sont de courte durée et répétées, les liaisons de service sont constamment créées et détruites, modifiant continuellement l'architecture de l'application. On parle alors d'architecture scintillante ou instable.

Sitôt qu'une application fait intervenir une interface graphique, un scintillement de l'architecture risque d'être propagé à l'utilisateur qui voit les composants graphiques de l'IHM évoluer indépendamment de sa volonté. Ses actions peuvent alors être perturbées par un degré de dynamisme trop élevé.

- Enfin lorsque le consommateur est lui-même fournisseur d'un service et fait partie d'une composition, l'interruption peut se propager à la composition et désactiver composants et applications en cascade. La **répercussion de l'interruption sur une composition de services** est un problème qui survient quand la réponse immédiate à une interruption est la désactivation du consommateur/fournisseur et donc le désenregistrement de son propre service, ce qui se traduit par une interruption de ce service et donc une désactivation de ses consommateurs, et ainsi de suite.

La Figure 34 représente une chaîne de composition de services. Plus celle-ci est longue, plus l'impact d'une interruption d'un service en début de chaîne sera important car déclencheur d'interruptions en cascade.

Figure 34. Chaîne de composition de services de rang n

IV.2 Service intermittent

Dans cette section nous supposons que les limites concernant la gestion des interruptions mises en évidence dans la section précédente viennent d'une interprétation incomplète du caractère de l'interruption. Etant donné ces limitations, nous introduisons ici le concept de service intermittent. La définition de ce concept nous amène à traiter de manière différente les interruptions de service puisque nous avons précédemment mis en évidence l'existence de cas où l'interruption n'est qu'un événement temporaire faisant partie intégrante du cycle de vie du service. Ainsi, dans les cas où l'interruption d'un service n'entrave pas de manière significative le fonctionnement de ses consommateurs, elle ne devrait pas être considérée comme une panne.

C'est pourquoi nous avons défini la notion de service intermittent. Il s'agit de services dont l'interruption fait partie dans une certaine limite (temporelle) de leur fonctionnement normal.

Par exemple, un service délivré par un équipement mobile est intermittent par nature.

IV.2.1 Caractérisation des services intermittents

On dit d'un phénomène qu'il est intermittent lorsqu'il est discontinu. Ce que nous appelons service intermittent désigne un service qui alterne périodes de fiabilité (i.e., de disponibilité continue) et d'indisponibilité.

"*Un service peut-être qualifié d'intermittent s'il est sujet, pendant ses périodes de fonctionnement, à des interruptions limitées dans le temps, de manière plus ou moins régulière. La limite temporelle et la fréquence des interruptions dépendent de l'appréciation de chaque service, de leur nature et du cadre de leur utilisation.* "

En fait, par abus de langage, en parlant de service intermittent, nous faisons référence à un fournisseur de service intermittent. En théorie tout service est intermittent, mais dans la pratique le caractère intermittent du service est souvent ignoré. Quand un fournisseur de service devient indisponible, son retour n'est pas envisagé car l'interruption est assimilée à une panne.

Un service intermittent se caractérise par un ensemble d'interruptions bornées dans le temps. Et la source de ces interruptions, prévisibles ou non, ne réside pas dans l'erreur logicielle. L'intermittence est donc une caractéristique faisant partie du fonctionnement normal d'un service dont les interruptions doivent être prises en compte.

Un fournisseur de service intermittent doit aussi être identifiable de manière unique et persistante afin de pouvoir être reconnu d'une période de fonctionnement à l'autre à travers ses interruptions. Ainsi les consommateurs du service peuvent tracer le fournisseur et avoir une « mémoire » du service intermittent.

La Figure 35 représente le service intermittent sous forme de schéma UML. Par rapport à un service classique, en plus de sa capacité à être identifié et de ses interruptions, le service intermittent suit un cycle de vie légèrement différent de celui des autres services. La section suivante explique cette différence de vision du cycle de vie.

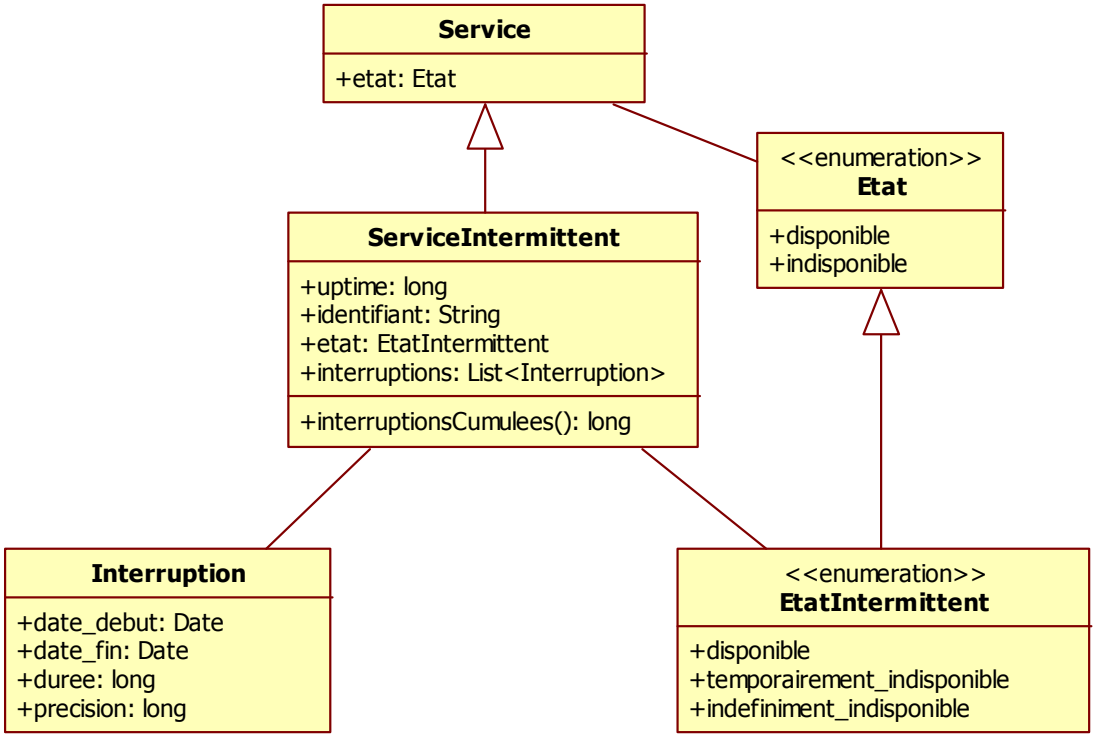

Figure 35. Schéma UML de caractérisation des services intermittents

Note : nous faisons la distinction entre service intermittent et service éphémère. Cette dernière catégorie correspond aux services ayant une durée de fonctionnement limitée dans le temps et qui ne redeviennent pas disponibles après interruption.

IV.2.2 Cycle de vie

Les approches classiques décrites en deuxième partie de la section IV.1.4.2 ont une vision partielle du cycle de vie des services, induite par le fort découplage de l'approche à service. Ainsi avec les politiques de liaison dynamiques, si le fournisseur d'un service est momentanément indisponible, son retour à un état de disponibilité (de fonctionnement) ne sera pas traité différemment de l'arrivée de n'importe quel autre fournisseur du même service (Figure 36.a). La notion de fournisseur de service s'efface derrière le concept de service.

Ainsi comme expliqué précédemment les choix de liaison résultant des politiques de liaison ne sont pas toujours optimaux et peuvent induire des comportements inefficaces

voire mauvais. Afin de pouvoir prendre une décision sur la création, destruction ou conservation de la liaison, il est nécessaire d'avoir une indication sur la probabilité de retour d'un fournisseur de service. Ce qui ne peut pas être déduit de la simple observation d'un désenregistrement/retrait du service traduisant une interruption de service. D'où la nécessité d'avoir une telle information pour guider le choix de la stratégie à adopter.

Les services que nous considérons comme intermittents ont un cycle de vie prenant en compte le phénomène de « déjà-vù », c'est-à-dire qu'un consommateur de service doit garder en mémoire le fournisseur de service précédemment utilisé pendant un certain temps durant lequel ce dernier est seulement considéré comme *interrompu*. En effet, lorsqu'un fournisseur devient indisponible, sa probabilité de retour est prise en compte afin d'offrir une tolérance à son interruption. Ceci dans une certaine limite au-delà de laquelle l'indisponibilité est constatée et la possibilité de retour écartée. Ainsi, sur la Figure 36.b l'état classique d'indisponibilité est décomposé en deux états : *interrompu* et *inconnu*. Au départ le service est « inconnu » avant d'être publié pour la première fois, et de devenir alors disponible. Puis, suite à une interruption le service intermittent est retiré, et avant d'être considéré comme indisponible et donc oublié de ses consommateurs, il passe par l'état *interrompu* où son retour est attendu jusqu'à une certaine limite temporelle.

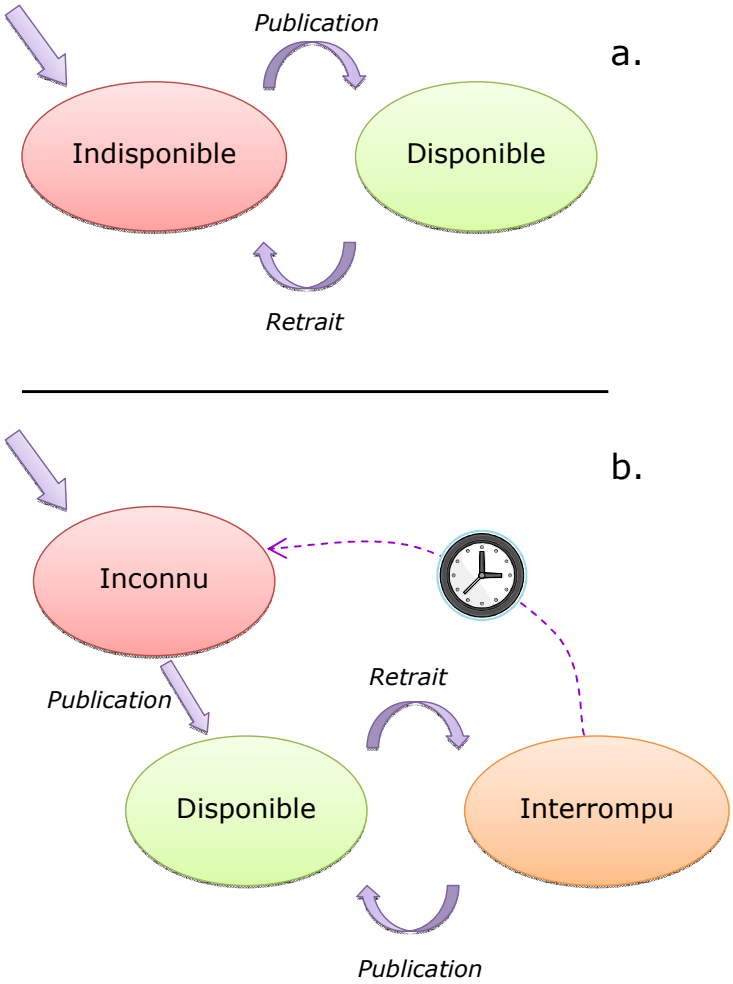

Figure 36. Cycle de vie d'un service intermittent

IV.2.3 Critères de l'intermittence

La disponibilité telle qu'elle est traitée de manière classique est exprimée par des indices (ratio ou pourcentage) trop peu précis pour qualifier les interruptions de services intermittents qui sont plus courantes que celles des services Web par exemple. Alors que les serveurs Web sont hautement disponibles et qu'une interruption est le plus souvent la conséquence d'une panne, les services enfouis dans un environnement intelligent sont plus sujets aux interruptions (alimentation, mobilité, maintenance, … cf. section IV.1.2) et n'ont donc ni la robustesse ni la disponibilité d'un serveur tournant 24h/24 et 7j/7. C'est pourquoi nous proposons un raffinement de la disponibilité telle qu'elle est traitée classiquement dans les SLAs, spécifiquement pour les services intermittents.

Pour cette extension nous avons fait le choix de trois indicateurs permettant de caractériser les services intermittents :

- Temps d'interruption maximum ;
- Cumul des temps d'interruption ;
- Temps minimum de fonctionnement entre interruptions.

Le premier critère retenu est le **temps maximum d'une interruption**, ou seuil de tolérance de la durée d'interruption. C'est-à-dire le temps d'indisponibilité maximum entre deux périodes de fonctionnement. En reprenant la terminologie classique utilisée dans le domaine de la « sûreté », il s'agirait du MaxTTR (*Maximum Time To Repair*).

Toutefois ce critère seul ne permet pas de caractériser les services intermittents. En effet, même des interruptions de durée inférieure à ce seuil de tolérance peuvent se succéder et ainsi gêner une application à services, les périodes de fonctionnement étant entrecoupées d'interruptions. Pour répondre à cette problématique nous proposons deux critères supplémentaires : la somme des temps d'interruption cumulés sur une période, et le temps minimum obligatoire de service sans interruption.

Le premier, le seuil de **cumul des temps d'interruption**, s'exprime sur une fenêtre temporelle. Une fenêtre temporelle peut-être décrite de deux manières différentes : soit par une durée (et dans ce cas on dit que la fenêtre est « glissante », ses dates de début et de fin ne sont pas fixes), soit avec une date de début et une durée (ou une date de fin, ce qui revient au même) et la fenêtre est alors fixe dans le temps. La fenêtre de temps peut s'étendre sur la durée totale du contrat ou bien représenter un intervalle fixe (e.g., Janvier 2009, entre 8h et 18h, …) ou mobile (e.g., 24h, un mois, …).

Soit I_t un ensemble discret d'interruptions sur une période T et d_i la durée de l'interruption i tel que i appartient à I_t. Le critère CTTR (*Cumulated Time To Repair*) sur une période T se mesure alors de la manière suivante :

$$CTTR(T) = \sum_{i \in I_t} d_i$$

Et de la même manière que pour la durée d'une interruption, le cumul ne doit pas dépasser un certain seuil de tolérance.

Notons que l'on peut ramener facilement la disponibilité à un ratio/pourcentage classique à partir du temps d'interruption cumulé sur une certaine période (telle que la durée d'un contrat). Ainsi la disponibilité D_T sur un intervalle T s'exprime de la manière suivante :

$$D_T = 1 - \frac{CTTR(T)}{T}$$

Le second critère que nous proposons, le **temps minimum de fonctionnement** correspond à un temps minimum de fonctionnement entre pannes (*MinTBF pour Minimum Time Between Failures*). Ce dernier prend son sens pour les architectures logicielles requérant une certaine stabilité où se pose le problème des interruptions éphémères mais répétées à l'origine des architectures « scintillantes ». Ce qui est typiquement la situation des applications dites « mission-critiques ».

Néanmoins nous ne prétendons pas l'exhaustivité de cette liste et l'intermittence de service devrait éventuellement faire l'objet d'une étude plus approfondie afin de couvrir tous les cas de figure. Nous avons en effet identifié des cas d'utilisation où ces trois critères ne suffisent pas à caractériser l'intermittence d'un service. Par exemple, certains services peuvent avoir une disponibilité liée à des fenêtres de temps bornées par des dates fixes (ex : de 8h à 20h, du lundi au vendredi), et les consommateurs de services sont susceptibles d'exprimer leurs besoins d'une manière similaire, notamment dans le cadre de tâches calendaires ou ordonnancées (C*ron*) nécessitant l'utilisation de services sur une période spécifique.

L'idée d'un seuil de tolérance temporel nous rapproche alors des problématiques du temps-réel et du principe d'échéances (ou *deadlines*). Toutefois, nous ne supposons pas ici des conditions de temps-réel. Ce point est abordé en détail dans les perspectives de cette thèse (cf. section IX.1).

Chapitre V
Politique de liaison dirigée par les accords de niveau de service

> *“ Le mariage, la confiance n’y est pas. Il faut des témoins, comme dans les accidents. ”*
>
> Coluche

Afin de palier aux manques des politiques de liaison proposées par les modèles à composants orientés service, nous proposons ici une politique sensible aux services intermittents, baptisée DSLA.

Cette politique considère le caractère intermittent des services afin d’optimiser la gestion des liaisons en offrant une tolérance aux interruptions de service. Pour cela elle a besoin de s’appuyer sur des indicateurs de disponibilité tels que ceux décrits dans le chapitre précédent. Il est donc nécessaire pour la mise en œuvre de cette politique que consommateurs et fournisseurs de services expriment à la fois leurs besoins et leurs capacités en regard de ces indicateurs, et de disposer d’un moyen de mesurer ces derniers.

En outre, cette approche repose sur la fiabilité des informations données par les consommateurs et fournisseurs de service, et nécessite donc un rapport de confiance. Les accords de niveau de service sont le moyen que nous avons choisi pour exprimer à la fois les garanties de disponibilité et la fiabilité de ces garanties. Les accords de niveau de service offrent de surcroit un moyen de surveiller le respect des engagements des parties, tout en définissant un cadre d’application des politiques de recours afin de faire face aux cas où ces engagements ne seraient pas respectés.

Ces accords peuvent également être bilatéraux. C’est-à-dire que la partie obligée par l’accord à une certaine disponibilité peut aussi bien être la partie prestataire que la partie cliente. Bien que le problème soit abordé sous l’angle classique de la disponibilité de service, il est à noter que l’intermittence est un phénomène pouvant toucher aussi bien les fournisseurs que les consommateurs de service. Un fournisseur peut en effet intégrer la notion de session et décider d’attendre le retour d’un consommateur indisponible pour le servir. Toutefois les problèmes posés ne sont pas les mêmes et ce point de recherche fait partie des perspectives de cette thèse.

V.1 Vers une politique de liaison consciente des interruptions

Notre proposition consiste en la définition d'une politique de liaison de service permettant d'optimiser la gestion automatique des liaisons par les modèles à composants orientés services, dans le cas des services intermittents. Cette politique se positionne à mi-chemin entre les approches statiques et dynamiques en tentant de combiner la stabilité architecturale des premières aux capacités d'adaptabilité et de reconfiguration des secondes.

V.1.1 Motivations

Les politiques de liaisons de service existantes dans les modèles à composants orientés services proposent essentiellement deux approches. Une politique statique, une fois que la liaison est établie elle n'évolue plus et une interruption de service signifie l'arrêt du composant, et une politique dynamique dite « adaptable » où les liaisons d'une application et par conséquent son architecture logicielle évoluent avec la disponibilité des fournisseurs de services. Toutefois cette approche dynamique ne prend pas en compte le caractère intermittent de certains services, ce qui peut amener l'application à adopter une architecture instable, indésirable ou encore induire des reconfigurations inefficaces. La section IV.1.4.2 détaille les cas défavorables liés à l'intermittence de services.

En plus du caractère statique ou dynamique des liaisons, la plupart des modèles à composants orientés services offrent une certaine tolérance aux interruptions à travers la notion de liaison *optionnelle*. Cette forme de liaison est utile pour les composants peu critiques et n'apportant pas de fonctionnalité essentielle. Un composant peut fonctionner sans qu'une de ses dépendances de service ne soit résolue (i.e., sans que le service ne soit disponible). Bien que cette technique permette d'ignorer les services indisponibles, elle ne répond pas à la problématique des services intermittents pouvant être essentiels à une application et ne devant donc pas être écartés sous couvert d'optionalité. Par exemple il pourrait être tentant de définir comme liaison optionnelle une liaison à un service ne servant qu'une seule fois à l'initialisation d'une application afin d'éviter qu'une interruption future (post-initialisation) de ce service n'entrave l'application. Mais si ce service n'est pas présent à l'initialisation alors il sera simplement ignoré et l'étape d'initialisation ne se sera pas déroulée correctement.

La décision menant à la reconfiguration de l'architecture ne prend pas en compte la nature de l'interruption. Ainsi sa durée est considérée comme indéfinie et donc potentiellement infinie. Néanmoins certains modèles à composants se sont récemment penchés sur le problème et proposent une gestion temporisée des liaisons.

A l'instar de SpringDM, qui permet d'attacher un *timeout* aux dépendances de service, et du TemporalService, une extension proposée en 2009 pour le DependencyManager de

Felix, le *TemporalDependencyHandler* d'Apache Felix iPOJO [Escoffier2009] permet de moduler le dynamisme de l'approche adaptative en liant une temporisation (*timeout*) aux dépendances de service. Cette politique de liaison permet de définir un délai d'interruption acceptable en dessous duquel l'interruption n'entraine pas la désactivation du composant consommateur. Cependant le découplage reste au cœur de cette politique, ainsi la relation client-fournisseur n'est pas considérée et il n'y a par conséquent pas de notion d'interruption cumulée dans le temps. De plus du fait de la vision uniquement « service », si un fournisseur est interrompu mais qu'un autre fournisseur du même service est disponible alors une reconfiguration architecturale sera opérée : la décision d'effectuer une substitution sera prise. Enfin le délai est fixé par le consommateur de service et ne dépend donc pas des capacités du fournisseur. Ce délai étant fixe, il n'évolue pas dans le temps et ne permet donc pas de considérer les interruptions répétées d'un service intermittent.

V.1.2 Principes

La politique que nous proposons offre une gestion plus fine des liaisons de service afin d'optimiser les reconfigurations architecturales des applications dynamiques orientées services, en particulier dans le cas de services intermittents. L'idée directrice repose sur le principe de temporisation et de conservation des liaisons, ce qui s'avère dans certains cas identifiés précédemment plus intéressant qu'une destruction suivie d'une recréation de liaison. En conservant la liaison, le lien consommateur-fournisseur n'est pas perdu et le composant consommateur reste actif. Ainsi le contexte d'exécution peut être conservé jusqu'au retour du fournisseur, les ressources réservées ne sont pas relâchées immédiatement, etc. De plus, un consommateur étant aussi fournisseur de service ne sera pas interrompu par effet domino, ce qui permet d'éviter les désactivations en cascade et les soucis d'architectures scintillantes ou instables.

Comme la liaison ne peut pas être conservée éternellement, en particulier si le fournisseur reste indisponible, les stratégies de substitution, ou de destruction dans le cas où la substitution est impossible, sont adoptées a posteriori. C'est-à-dire une fois que le conteneur du composant consommateur a attendu le retour du fournisseur jusqu'à son seuil de tolérance. Ce délai d'attente dépend à la fois des besoins de l'application, et de la nature des services.

Enfin, lorsqu'un consommateur est activé, s'il était lié à un fournisseur et que la liaison a été conservée, il doit pouvoir continuer à utiliser ce même fournisseur si ce dernier est toujours disponible.

Considérons une application simple comprenant un consommateur de service et deux fournisseurs de ce service (cf. Figure 37). La Figure 38 décrit un automate représentant les différentes reconfigurations de l'architecture de cette application. Les reconfigurations sont le résultat de changements dans la disponibilité des fournisseurs et sont représentées par les transitions. L'architecture peut se trouver dans trois états distincts :

- 0 : le consommateur n'est lié à aucun fournisseur de service, il est désactivé ;
- 1 : le consommateur est lié au prestataire n°1 ;
- 2 : le consommateur est lié au prestataire n°2 ;

Soit d_1 (et d_2) les transitions signifiant que le fournisseur 1 (ou 2) sont disponibles. ε désigne le seuil de tolérance aux interruptions. i < ε ou i > ε font référence à la durée de l'indisponibilité (i) et sa relation par rapport au seuil de tolérance.

Les transitions d_x, $i_x > \varepsilon$, et $i_x < \varepsilon$ impliquent respectivement la création d'une liaison, sa destruction ou sa conservation. Les transitions de type $d_x.i_y$ correspondent à une substitution (destruction de l'ancienne liaison et création d'une nouvelle vers le fournisseur x) ; au moment où le fournisseur y devient indisponible, si un fournisseur x est présent alors la substitution s'opère et le composant consommateur n'est pas désactivé.

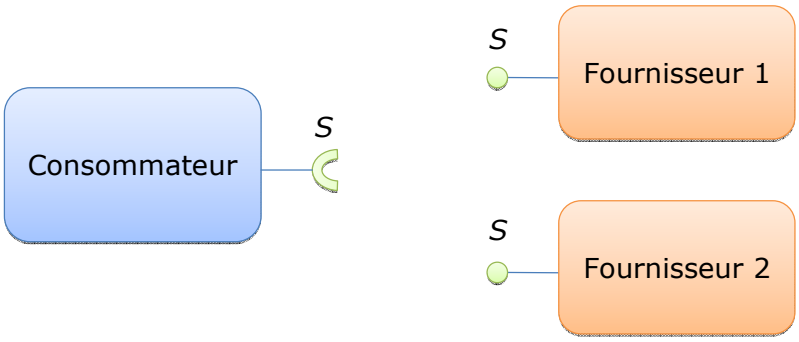

Figure 37. Configuration simple d'application multi-fournisseurs

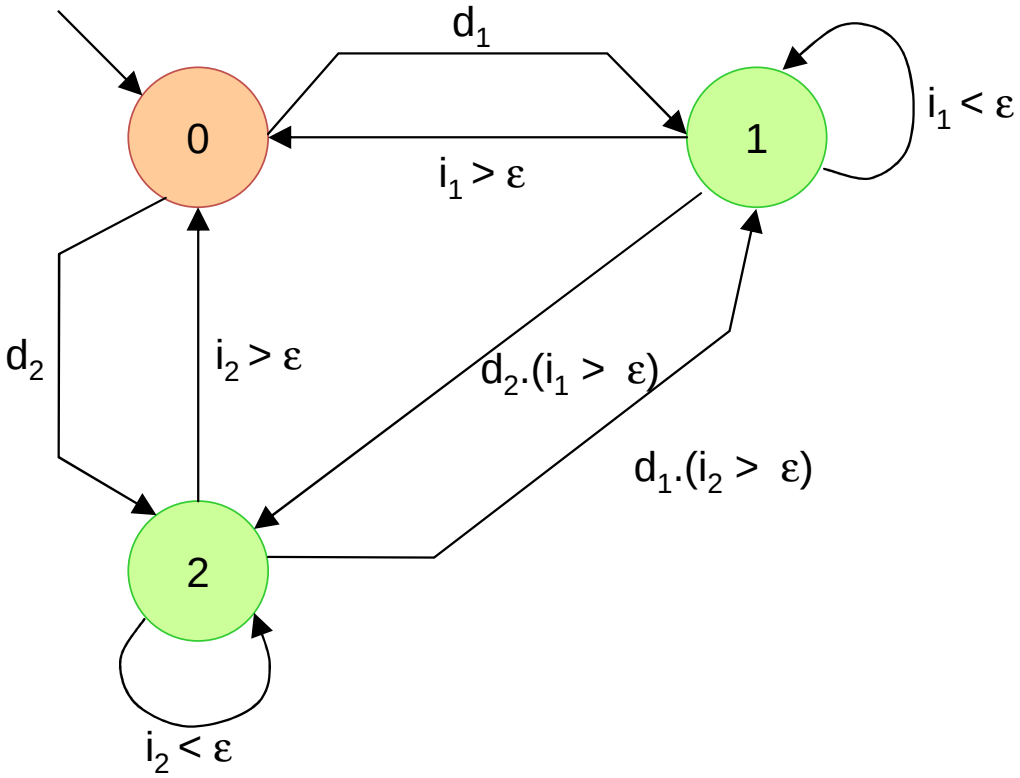

Figure 38. Automate des reconfigurations d'architecture suivant la politique DSLA

Afin d'adopter une stratégie optimale et de prendre la bonne décision concernant la conservation ou la destruction d'une liaison, il est premièrement nécessaire pour le conteneur de posséder certaines métriques qualifiant la disponibilité des fournisseurs de service impliqués et plus précisément leurs probables interruptions.

Deuxièmement, cette approche nécessite d'introduire un mécanisme de « mémoire » des fournisseurs de service. D'une part pour être capable d'identifier le retour d'un fournisseur suite à une interruption. Et d'autre part, afin d'avoir accès à l'historique des interruptions d'un prestataire, ceci dans le but d'assister la prise de décision concernant la conservation ou la destruction d'une liaison.

Ceci est possible à partir du moment où les prestataires peuvent être identifiés de manière unique et persistante. L'unicité permet de ne pas confondre deux fournisseurs, tandis que la persistance de l'identification permet d'identifier et reconnaître un même fournisseur d'une interruption à l'autre. Toutefois cette connaissance des fournisseurs de service ne doit apparaitre qu'au niveau du conteneur. Le composant consommateur, quant à lui, conserve une vue à l'échelle du service, afin de conserver le découplage entre composants consommateurs et fournisseurs.

L'optimisation de la gestion des liaisons présente donc les trois requis suivants :

- Un moyen d'identification unique et persistante des fournisseurs de service ;
- Un accès à l'historique des interruptions d'un fournisseur ;
- Un accès à des informations concernant la capacité de disponibilité d'un fournisseur.

Notons que ces exigences peuvent tout aussi bien porter sur le consommateur de service dans le cas d'une liaison avec contraintes de disponibilité bilatérales.

V.1.3 Positionnement entre architectures statiques et dynamiques

Reprenons les notations précédentes afin de comparer les politiques de reconfiguration de liaisons existantes à notre approche dans le cas de l'application simple ou un consommateur utilise un service fourni par deux prestataires distincts. Les transitions i_1 et i_2 correspondent à l'interruption du fournisseur qui devient alors indisponible, et impliquent la destruction de la liaison et donc une désactivation du consommateur qui n'est plus lié. La Figure 39 représente, par le biais d'automates, les différentes configurations d'architecture possibles selon les approches statique, adaptive (dynamique), temporelle et celle que nous proposons.

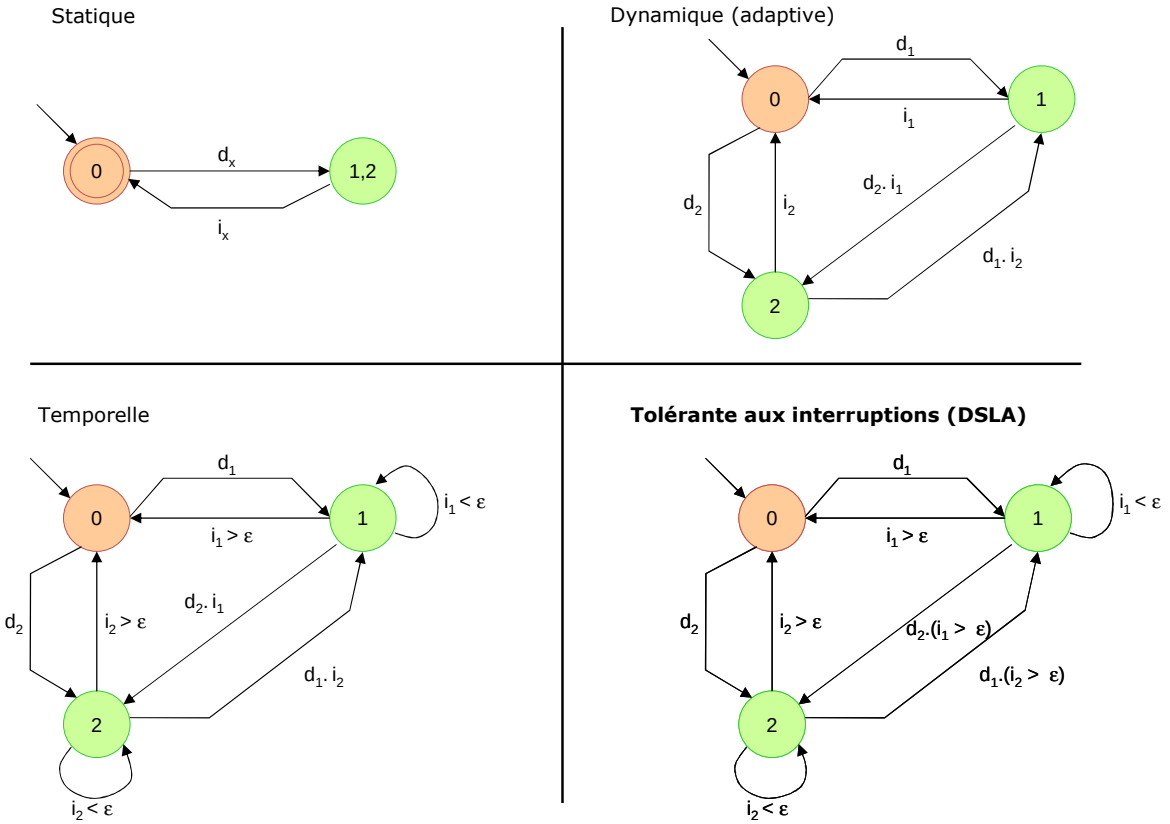

Figure 39. Automates résultants des différentes politiques de liaison

Nous pouvons constater sur ces automates que la politique statique ne tolère pas l'interruption tandis que l'approche dynamique entraine des reconfigurations à chaque transition. La politique « temporelle » réduit le nombre de reconfigurations en limitant les transitions vers l'état 0 et donc les désactivations du consommateur. Par rapport à cette dernière, notre approche permet de réduire le nombre de reconfigurations causées par les substitutions.

Le but de nos travaux est en effet de proposer une alternative à l'approche adaptative des politiques (que nous jugeons trop dynamique) afin de limiter ses effets néfastes sur une classe particulière de service, les services intermittents. Sur la Figure 40 nous positionnons notre politique (DSLA) à mi-chemin entre les politiques offrant une gestion fortement dynamique des liaisons qui autorisent ainsi les reconfigurations architecturales à l'exécution, et celles garantissant la stabilité d'une architecture à travers des liaisons statiques mais qui ne permettent pas la construction d'applications auto-adaptables. La politique que nous cherchons à mettre en œuvre est de surcroît un peu moins dynamique que les politiques de liaisons temporisées, puisque la substitution de service ne s'opère pas tant qu'un composant est lié à un fournisseur même si celui-ci est momentanément indisponible.

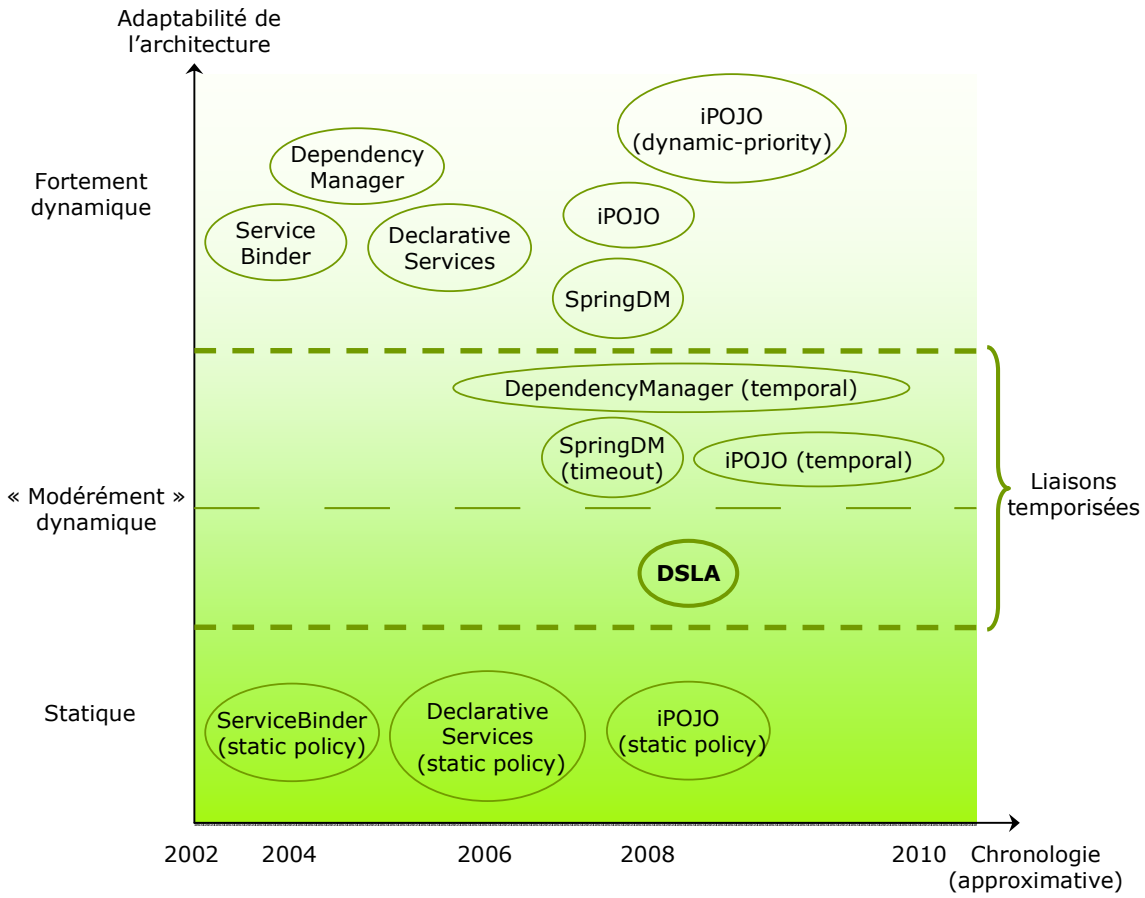

Figure 40. Positionnement des différentes politiques de liaison

V.2 Caractérisation des interruptions par les accords de niveau de service

Afin de mettre en œuvre une politique de liaison sensible aux interruptions la section précédente a mis en évidence plusieurs requis. Notamment le fait de pouvoir s'appuyer sur des informations concernant la disponibilité des fournisseurs, et plus particulièrement sur leurs interruptions.

Les accords de niveau de service permettent déjà d'exprimer des garanties concernant la disponibilité d'un service, généralement sous forme de ratio ou pourcentage pour les services hautement disponibles. Nous proposons ici un raffinement de la disponibilité afin de l'exprimer dans des SLAs destinés aux services intermittents. Ces SLAs pourront ainsi servir de base à notre politique de liaison s'appuyant sur les attentes et garanties des parties prestataires et consommatrices de service.

De plus un accord de niveau de service permet d'identifier les parties qui y sont engagées pendant la durée du contrat, ce qui répond à un autre besoin de la politique de liaison que nous proposons. Et ceci tout en gardant le découplage faible entre composants : avant les étapes de sélection et de négociation un consommateur de service ne sait pas à quel fournisseur il sera lié. Ce n'est qu'une fois un accord établi qu'il a conscience de son ou ses partenaires.

Les techniques de gestion du niveau de service (SLM) permettent également de suivre le respect de l'accord, et donc de suivre les interruptions de services, et de réagir en cas de dépassement des seuils fixés par l'accord.

Enfin les garanties fournies par les SLAs permettent d'assurer un certain niveau de stabilité aux applications orientées services, notamment pour les applications missions-critiques composée de services intermittents. Ces dernières ne peuvent en effet pas tolérer des reconfigurations architecturales intempestives [Touseau2007].

V.2.1 Objectifs de niveau de service sur l'intermittence

Dans l'optique de la conception d'une politique de liaison sensible aux services intermittents, il est nécessaire que le gestionnaire de liaisons ait accès aux informations relatives à la disponibilité des parties afin d'inférer un seuil de tolérance aux interruptions. Ces informations sont apportées par les objectifs de niveau de service (SLOs) de l'accord passé entre les parties. Nous y retrouvons donc les critères de qualité de service caractérisant la disponibilité des services intermittents décrits dans le chapitre précédent (section IV.2.3). La Figure 41 représente le méta-modèle de SLOs spécifiques aux services intermittents, étendant le modèle de SLO générique présenté dans l'état de l'art sur les accords de niveau de service.

On y retrouve donc :

- les informations génériques telles que le nom, la description ou la partie concernée par l'obligation,
- ainsi que les métriques (valeur et unité) associées à une relation d'ordre (inférieur, supérieur ou égal à),
- une condition de validité optionnelle (telle qu'une fenêtre temporelle par exemple),
- et une possibilité d'extension exprimées à l'aide de propriétés (ensemble de clés-valeurs). Cette extension peut selon les besoins donner un degré de précision (ou marge d'erreur) sur la valeur du SLO, indiquer une fenêtre temporelle, etc…)

Les trois critères retenus pour caractériser un service intermittent se retrouvent donc sous forme de SLOs. L'objectif *MaxTTR* désigne la durée maximum d'une interruption, *CumulTTR*, le temps d'interruption cumulé et *MinTBF*, la durée de fonctionnement minimale entre deux interruptions. Ces critères sont comme tous SLOs définis par une valeur, une unité, une précision, une relation d'ordre et des conditions de validité pouvant désigner une fenêtre temporelle.

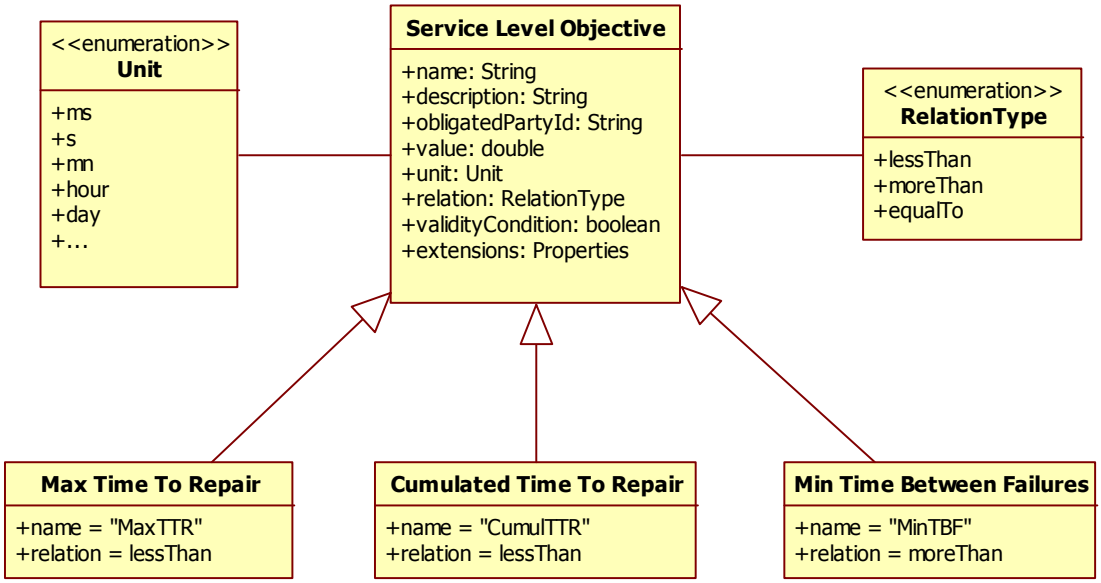

Figure 41. Méta-modèle des SLO portant sur la disponibilité des services intermittents

Notons que dans le contexte des services intermittents, les notions de *Repair* et *Failure* dans le nom des indicateurs ne font pas ici forcément référence aux pannes ou réparations mais plutôt à l'interruption d'un fournisseur de service et son retour à un état de disponibilité. Les noms ont été simplement choisis en référence aux métriques usuelles de disponibilité.

Enfin, ces SLOs ne représentent qu'un sous-ensemble des SLOs pouvant peupler un SLA permettant de mettre en œuvre notre politique de liaison pour services intermittents. Selon le même modèle de SLO générique il est alors possible de décrire des objectifs concernant d'autres critères de qualité de service tels que le temps de réponse, le nombre d'appels par client, par session ou encore la qualité d'image ou du flux dans un contexte de service vidéo.

V.2.2 Négociation

Les objectifs de niveau de service, ou en d'autres termes les obligations des parties, sont décrits dans un accord de niveau de service. Ils résultent par conséquent du processus de négociation débouchant sur l'accord. Comme décrit dans l'état de l'art (section III.2.2.2), le processus de négociation confronte les besoins et capacités des parties souhaitant coopérer. Il est donc nécessaire dans un premier temps que consommateurs et fournisseurs de service **identifient** et **expriment** leurs besoins et leurs capacités vis-à-vis des indices de disponibilité qui seront alors retranscrits dans les SLOs au terme d'une **négociation** réussie. Ces trois étapes sont schématisées sur la Figure 42.

L'analyse des besoins et capacités d'un composant (qu'il soit consommateur ou fournisseur de service) peut se faire durant sa phase de développement (a priori), ou à l'exécution (a posteriori).

Si elle faite a priori, il s'agit d'une analyse prévisionnelle des besoins. Cette dernière peut se faire à partir des spécifications du composant ou par observation de son comportement lors de tests de performance, par exemple.

Une analyse faite a posteriori, c'est-à-dire par observation des composants en fonctionnement puis par analyse des résultats de l'observation, peut être conduite par la partie propriétaire du composant ou par des parties tierces. Nous faisons l'hypothèse que sur une période assez longue d'observation d'un fournisseur de service il est possible d'inférer ses capacités en termes de disponibilité.

Une fois exprimés, ces indices caractéristiques des services intermittents deviennent alors au même titre que d'autres critères de qualité de service, objets de la négociation. Durant l'étape de négociation le choix des fournisseurs de service dépend donc de paramètres multiples. Il est alors courant d'avoir recours à des fonctions de coûts ou fonctions d'utilité [Bottaro2007] permettant de classer les différents prestataires afin de choisir le plus approprié.

La complexité du processus de négociation peut être variable (cf. Figure 25) comme le démontre l'état de l'art. Toutefois en ce qui nous concerne nous nous limiterons à une négociation du niveau de la simple « sélection ». C'est-à-dire que la négociation est réussie si et seulement si les besoins et capacités des parties correspondent. Néanmoins nous ne proposons pas de résoudre les problèmes de correspondance sémantique entre l'expression des capacités et besoins des parties qui font déjà l'objet d'autres travaux [Oldham2006].

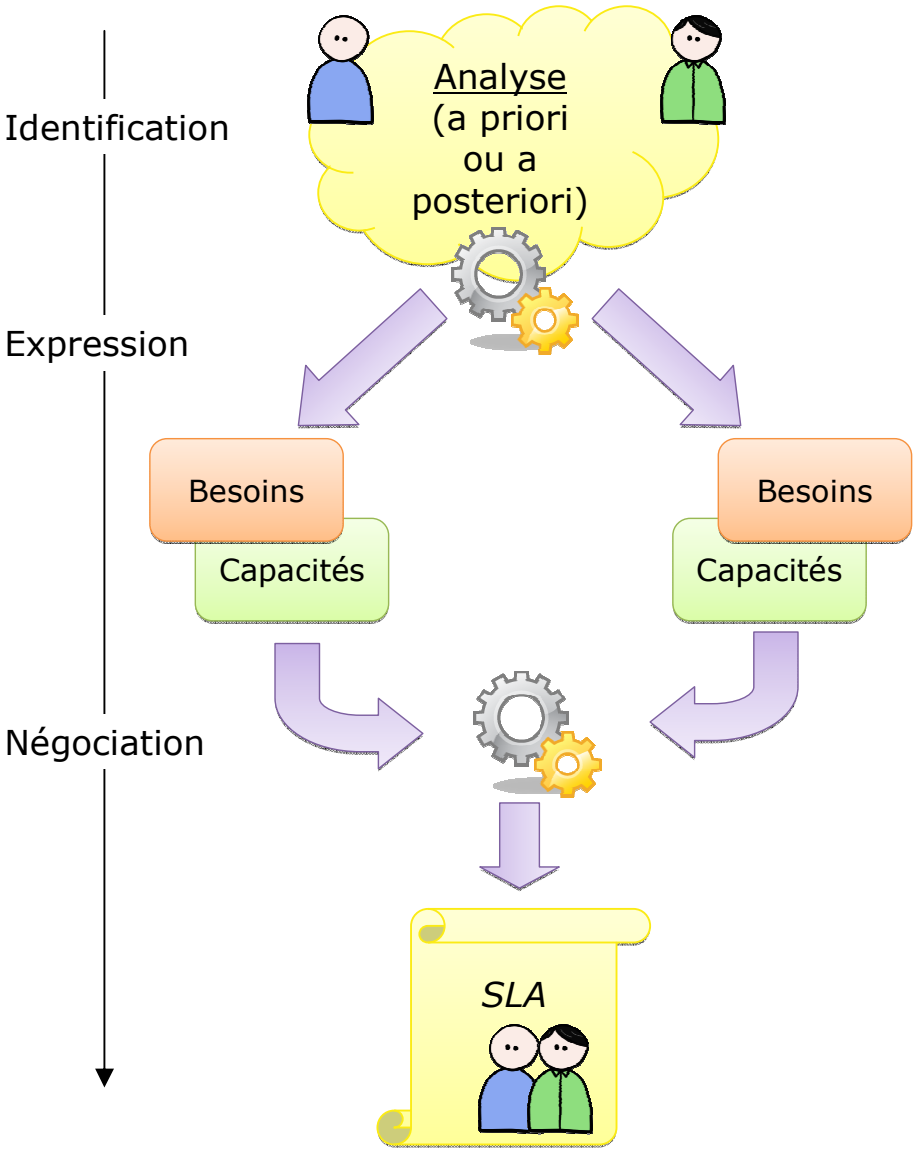

Figure 42. Etapes de négociation d'un SLA

V.2.3 Mécanismes de gestion des accords

Dans notre cas le mécanisme de gestion des accords (SLM) se décompose en trois parties : un moniteur/superviseur dont les résultats sont exploités par un évaluateur de conformité, et une partie chargée de l'application des politiques de liaison et des politiques de recours ou pénalités.

Le gestionnaire de niveau de service (SLM) présente une architecture analogue à celle d'un gestionnaire autonomique de type MAPE-K (cf. Figure 27). La Figure 43 situe les composants de notre SLM sous forme de gestionnaire autonomique.

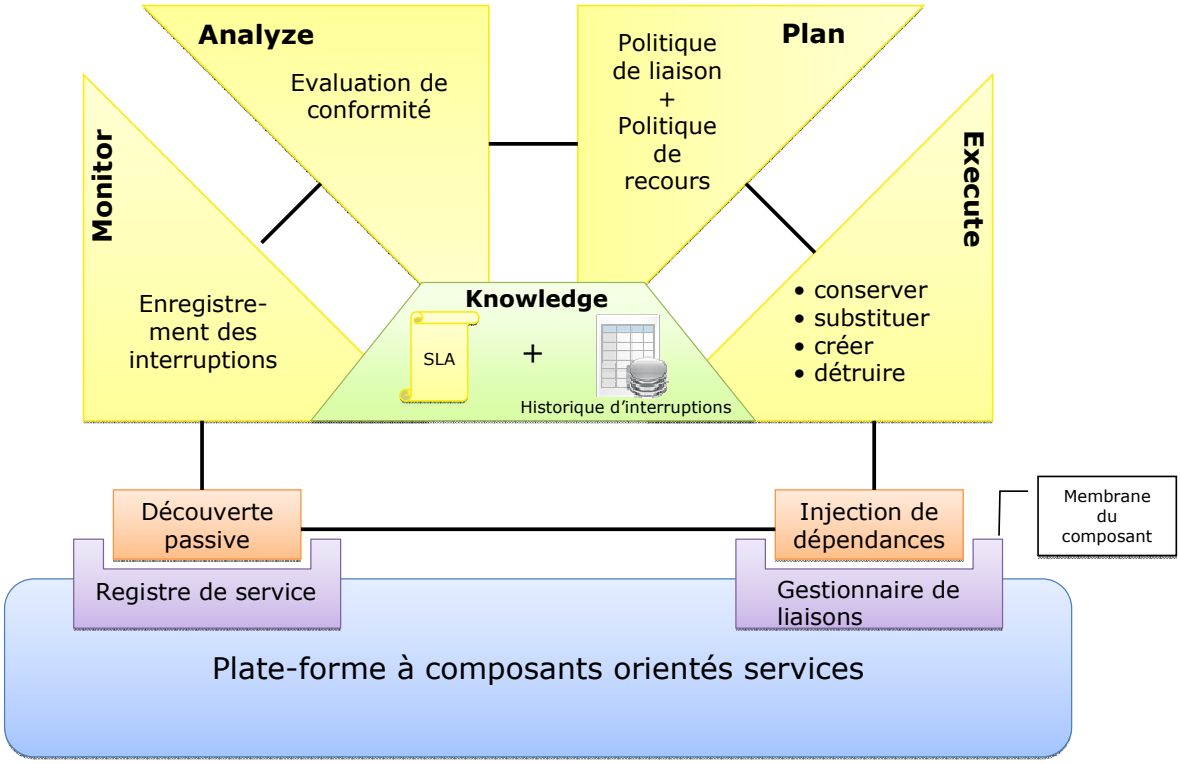

Figure 43. Représentation du gestionnaire DSLM sous forme d'un gestionnaire autonomique

Le superviseur utilise les informations glanées par des sondes pour mesurer les indicateurs de qualité de service –ici les indices d'intermittence de service- selon les obligations définies dans le SLA. Ensuite les informations relevées par le superviseur sont à la fois gardées en mémoire au niveau « connaissance » afin de conserver un historique des interruptions enregistrées, et peuvent être alors exploitées par le composant chargé d'évaluer le respect des SLOs. A partir de l'analyse des interruptions de services, le gestionnaire de niveau de service peut prendre plusieurs décisions. D'une part concernant la reconfiguration de l'architecture d'une application orientée service supervisée, et d'autre part concernant l'accord de niveau de service et l'application de politiques de recours (comme la terminaison du contrat). Ces décisions se traduisent alors par :

- des modifications de l'état de l'accord du niveau de service (état, validité) et l'application de pénalités si une obligation n'a pas été respectée;
- des changements au niveau du gestionnaire de liaisons, dans la membrane des composants orientés service. Une liaison peut ainsi être conservée suite à une interruption, ou bien détruite ou remplacée par une autre si l'interruption a dépassé les bornes fixées par les SLOs.

Les sondes utilisées dans notre cas -c'est-à-dire pour les SLOs portant sur les interruptions de service- doivent simplement surveiller et rendre compte des interruptions de service. En pratique, dans le cas de plates-formes dynamiques à services supportant la découverte passive, il s'agit de s'abonner au registre de service pour recevoir les notifications de publication et de retrait des fournisseurs, ou bien d'écouter

directement les annonces faites par les prestataires (e.g., multicast UPnP). Nous faisons donc l'hypothèse qu'une interruption de service, quelque soit sa nature, entraîne un retrait du fournisseur du registre. En pratique lorsque l'interruption est due à une panne le retrait du registre n'est pas toujours mis en œuvre.

Dans le contexte de capteurs et d'équipements sans fil, afin de détecter une interruption qui n'aurait pas entraîné un désenregistrement « propre » du service, il est courant d'utiliser des mécanismes de *ping* ou *heartbeat*. Nous supposons néanmoins que si un fournisseur de service est lié à un appareil, une interruption (perte de connexion avec l'appareil) se traduira par un retrait du service, ceci dans le but d'avertir les consommateurs liés à ce fournisseur.

Suite à l'évaluation du respect des SLOs, le SLM en plus d'adapter les liaisons de service doit veiller à l'application de politiques définies dans l'accord. En ce qui concerne l'application de politiques de recours les réactions possibles à une violation de SLO sont multiples. En voici une liste non-exhaustive :

- Rupture du contrat
- Mise sur liste noire (*blacklisting*) afin de ne plus interagir avec la partie fautive. Ce *blacklisting* peut-être permanent ou temporaire et disparaître avec le temps.
- Système de bonus et malus attribués aux parties afin d'influencer les futurs choix de partenaires
- Financières : surfacturation, surtaxation ou dédommagement (selon la partie fautive)
- Ou la simple journalisation de la rupture des clauses qui seront alors traitées ultérieurement par un opérateur humain
- …

Certaines pénalités peuvent en outre être assorties d'un changement de prestataire.

Quant aux décisions portant sur les reconfigurations dynamiques des applications supervisées, les « points d'accroche » (*touchpoints*) de la boucle autonomique correspondent, dans les cas de composants orientés services, à la partie du conteneur gérant les dépendances de service. Les dépendances peuvent ainsi être gelées (conservées), substituées ou invalidées en fonction de l'état de l'accord et fournisseurs de service disponibles.

Chapitre VI

Réalisation

“ Ce qui distingue d'emblée le pire architecte de l'abeille la plus experte, c'est qu'il a construit la cellule dans sa tête avant de la construire dans la ruche. ”

Karl Marx

Ce chapitre explique comment les concepts décrits précédemment ont été mis en œuvre. Il porte notamment sur les choix de conception architecturale. Le gestionnaire de liaison et l’infrastructure de gestion des accords de niveau de service (SLA) s’appuient sur les plates-formes dynamiques à service et tirent partie de l’approche des modèles à composants orientés service.

L’architecture proposée intervient à deux niveaux :

- au niveau du conteneur des modèles à composants afin de l’étendre avec les concepts de services intermittents et de modifier la gestion des liaisons de services.
- au niveau des services proposés par la plate-forme à service sous-jacente : les composants logiciels de gestion des SLAs présentés sous la forme de services utilisés par le conteneur pourront ainsi être réutilisés par d’autres applications ayant besoin de gérer des accords de niveau de service.

Une fois étendu, le conteneur des composants est capable d’appliquer la politique de liaison « DSLA » grâce aux services fournis par le gestionnaire d’accords de niveau de service. Ce dernier utilise le service fourni par une sonde traquant les fournisseurs sous contrat et enregistrant leurs interruptions pour surveiller le respect des obligations concernant la disponibilité des services intermittents.

VI.1 Architecture générale

Le schéma de la Figure 44 représente l'architecture générale retenue pour la mise en œuvre de notre politique DSLA de liaison basée sur les accords de niveau de service [Touseau2008b]. Le canevas que nous proposons se décompose en trois parties.

1. La gestion des liaisons de services et l'application de notre politique sensible aux interruptions se font au niveau du conteneur des composants. Dans ce conteneur la distinction est faite entre les rôles de consommateur et de fournisseur.
2. Le conteneur interagit ensuite avec le gestionnaire d'accords de niveau de service (*DSLA Manager*) pour :
   - Etablir la liaison entre deux composants suite à la négociation d'un accord ;
   - Utiliser les informations fournies par le gestionnaire de niveau de service (*DSLM*) afin de mettre en œuvre une politique de liaison dirigée par les SLAs.
3. Le gestionnaire d'accords de niveau de service a quant à lui besoin d'informations sur les interruptions de service. Il requiert donc un service dédié au suivi des interruptions, le *DisruptionLogger*.

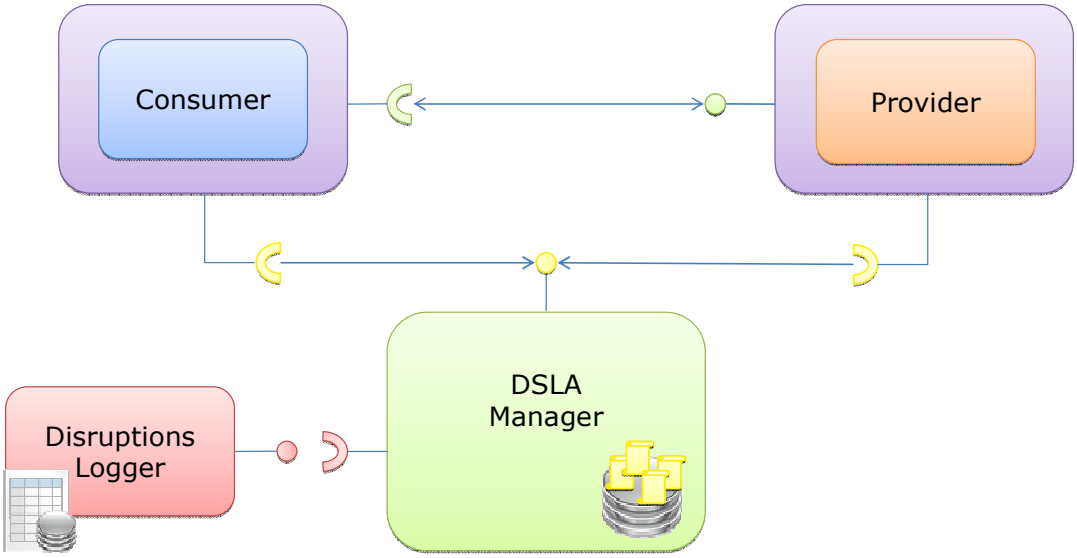

Figure 44. Architecture générale

Enfin pour répondre au besoin d'identification les parties d'un accord doivent être identifiables de manière unique et persistante. Certaines plates-formes à service intègrent de base cette gestion de l'identification unique. Par exemple, UPnP permet de définir des UUID (*Universally Unique IDentifier*) pour les services qui sont forgés par l'équipement ou bien générés à l'installation. OSGi définit une propriété spéciale service.pid (pour *peristant id*) qui peut être publiée avec le service. Toutefois les consommateurs ne fournissant pas de service doivent eux aussi pouvoir être identifiés.

VI.2 Supervision des interruptions de services

Le DisruptionLogger fait office de sonde en fournissant un service de mesure des interruptions de service. Contrairement aux sondes servant à la mesure d'autres critères/paramètres de qualité de service, celle-ci a l'avantage de ne pas être intrusive puisque la disponibilité d'un service et ses interruptions se constatent au niveau des mécanismes de publication ou de retrait (e.g., annuaire de service ou autre mécanisme de découverte passive).

Le DisruptionLogger propose un service de supervision des interruptions afin de pouvoir être utilisé par d'autres applications que notre DSLA Manager. Ce service offre la possibilité à un composant d'ajouter un prestataire à superviser et de s'abonner afin d'être notifié des interruptions et retours d'interruption de ce fournisseur via des méthodes de rappel (*callbacks*).

La Figure 45 représente un composant abonné (*DisruptionsListener*) et son interaction avec le service de journalisation des interruptions (*DisruptionsLogService*).

A chaque retour d'interruption d'un fournisseur les composants abonnés sont notifiés de la durée de l'interruption. Ils peuvent en outre utiliser le service pour obtenir l'historique des interruptions de ce fournisseur. Les historiques conservés par le DisruptionLogger sont persistants et ne sont pas perdus en cas d'arrêt de ce dernier.

Enfin le composant abonné peut se désabonner du *logger* à tout moment. Lorsqu'il ne reste plus d'abonnés pour un fournisseur donné, nous avons fait le choix d'arrêter le suivi de ce prestataire et de supprimer son historique.

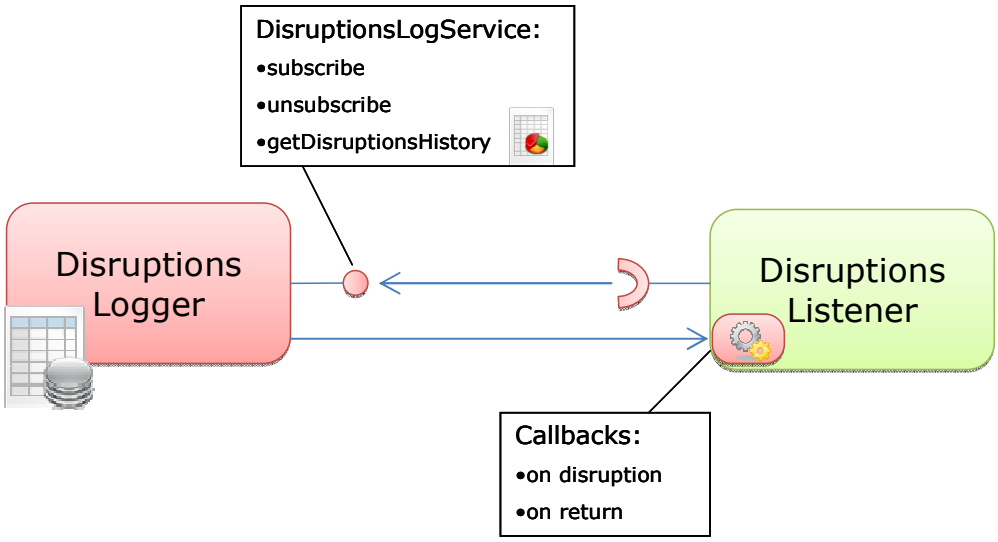

Figure 45. Utilisation du DisruptionsLogger

VI.3 Support des accords de niveau de service

Le DSLA Manager est le composant permettant la gestion des accords de niveau de service, depuis leur création jusqu'à leur terminaison. Il est donc responsable :

- de la création des accords,
- de leur conservation,
- de leur suivi et de l'évaluation du respect des objectifs de niveau de service (SLO) concernant les interruptions,
- et de leur terminaison de manière coordonnée afin que chaque partie, même absente, soit avertie.

Le schéma ci-dessous (Figure 46) représente le DSLA Manager. Ce gestionnaire composite est constitué d'un moniteur de niveau de service (DSLM) et d'un composant prenant en charge les négociations. Il est également fournisseur de deux services donnant accès à ses fonctionnalités listées au-dessus (négociation, accès aux accords passés, ...).

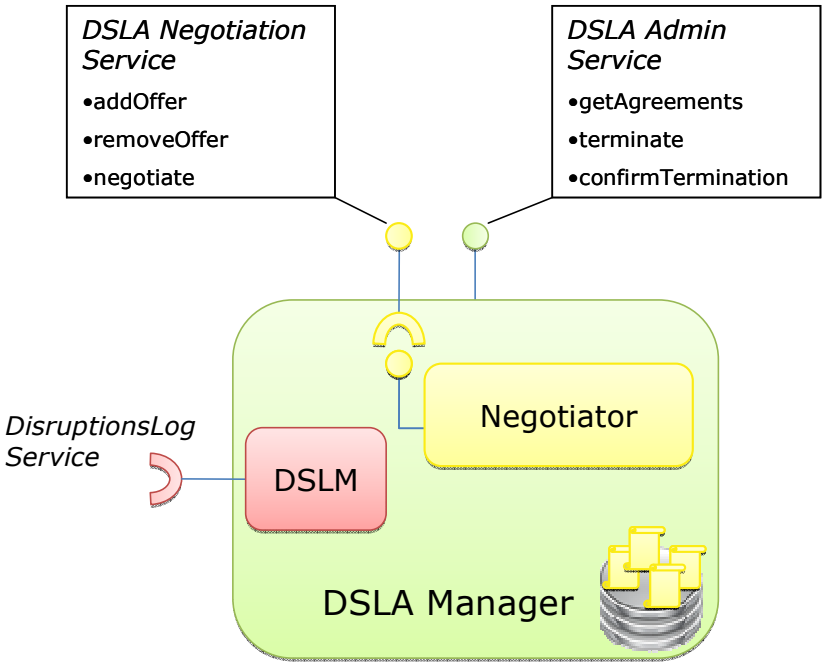

Figure 46. DSLA Manager

VI.3.1 Contenu d'un DSLA

Les accords de niveau de services gérés par le DSLA Manager ont une forme qui découle directement de l'analyse faite dans l'état de l'art. Le SLA (i.e., DSLA) se découpe en trois parties : les informations générales (nom, ID, description, service concerné), le contexte (dates de validité et parties impliquées) et les termes (principalement les objectifs de niveau de service).

Les SLOs sont de la forme décrite dans le chapitre précédent. Ils sont exprimés par un nom, une description, une valeur, une unité, une relation d'ordre en gardant une possibilité d'extension (ensembles de clés-valeurs) pour des informations telles que la précision de la mesure (marge d'erreur) ou les conditions de validité du SLO. Nous nous intéressons essentiellement aux critères de QdS caractéristiques de la disponibilité des services intermittents. Néanmoins d'autres objectifs peuvent être décrits via des paires clé-valeur. En revanche le système actuel ne fournit pas les mécanismes de mesure d'autres SLOs.

Par défaut la partie obligée est le fournisseur de service, nous ne proposons pas ici de résoudre le cas de l'intermittence des consommateurs qui nécessiterait une réflexion plus poussée sur la mesure de cette intermittence, sur les concepts de sessions et de contrôle d'admission. Néanmoins cette voie est présentée plus en détails dans les perspectives de cette thèse (chapitre IX.1).

VI.3.2 Négociation et établissement d'un SLA

Le DSLA Manager prend aussi en charge la négociation et donc la création d'un accord de niveau de service. L'accord se construit à partir des capacités et besoins des parties. En pratique le prestataire (ou plus rarement le consommateur [Andrieux et al 2004]) propose une offre servant de base à la négociation. Nous avons décidé de suivre ce mode de fonctionnement. Le schéma d'interaction Figure 47 montre les différentes étapes du processus de négociation entre un consommateur et un fournisseur de service.

Les fournisseurs de service publient une proposition d'accord en même temps que leur service. Les consommateurs en plus de trouver le service requis doivent alors confronter leurs besoins aux capacités (et vice versa) des fournisseurs découverts, activement ou passivement lorsque c'est possible, ceci afin de fixer les termes de l'accord (i.e., les SLOs). Cette offre est elle-même en quelque sorte tolérante aux interruptions puisqu'elle reste active même si le fournisseur est indisponible. Elle est automatiquement retirée par le gestionnaire de SLAs lorsque le fournisseur ne respecte plus ses SLOs. Toutefois ce dernier n'est pas mis sur liste noire et pourra proposer une autre offre ultérieurement.

Le service de négociation proposé par le DSLA Manager est fourni par un composant dédié à la gestion de la négociation. Il permet à un fournisseur d'enregistrer (ou retirer) une proposition de SLA au moment de la publication (ou du retrait) de son service, et à un consommateur de négocier une offre publiée. La négociation peut porter aussi bien sur les SLOs spécifiques aux interruptions de service que sur d'autres paramètres de qualité de service. Cependant, nous avons choisi de procéder par simple comparaison des valeurs des paramètres de qualité de service, ce qui fait que le composant de négociation reste du

niveau de la simple sélection. Pour qu'un accord soit conclu entre un consommateur et un fournisseur de service, il est également nécessaire que l'ensemble des besoins du consommateur soit un sous-ensemble de l'offre du fournisseur. Ce service de négociation pourra être raffiné par la suite pour supporter des protocoles plus complexes, mais ce n'est pas le but de cette thèse.

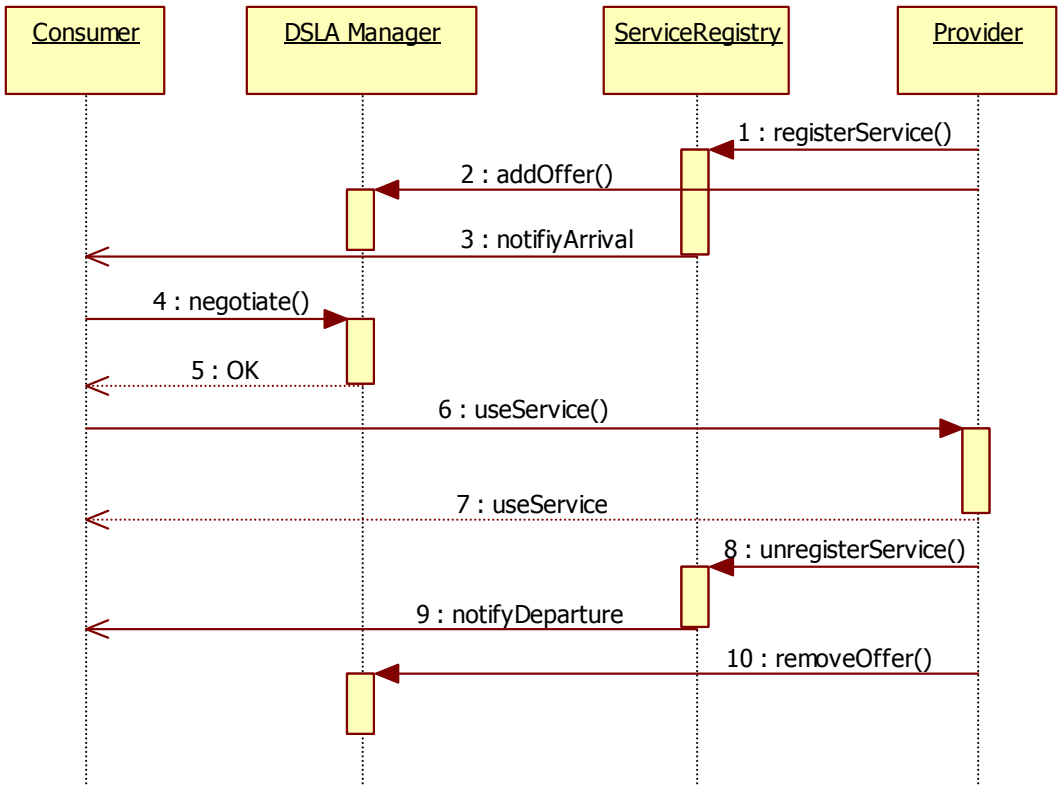

Figure 47. Diagramme de séquence UML du processus de négociation

VI.3.3 Suivi de la conformité

L'évaluation du respect des objectifs de niveau de service, et donc de la conformité à l'accord, est un des rôles du SLM (*Service Level Management*). Dans notre cas, elle est gérée par un composant appelé DSLM dont la tâche est de comparer les mesures faites par les sondes et de déclencher des actions lorsque ces mesures ne sont plus en accord avec les termes du SLA.

Le DSLM utilise le service du DisruptionsLogger comme sonde de mesure des interruptions de service. Lorsqu'un SLA entre en application, le DSLM fait appel au DisruptionsLogger pour surveiller un fournisseur. Il y enregistre un objet servant (*DisruptionsListener*, cf. Figure 45) qui est ensuite notifié en cas d'interruption et de retour via deux méthodes de rappel. Quand une interruption survient, le DSLM programme selon un mécanisme de chien de garde (*watchdog*) un rapport de violation à l'instant t où t correspond à l'échéance la plus proche dans le temps : le temps restant avant le dépassement de la durée maximum d'interruption, ou celui du seuil des durées

d'interruptions cumulées. Si le prestataire est de retour avant l'échéance alors la tache est simplement annulée. Un mécanisme similaire est utilisé au retour d'un service afin d'évaluer le troisième critère, c'est-à-dire le temps minimum de fonctionnement.

A la création d'un accord, le DSLA Manager crée une instance du composant DSLM chargé de la supervision du SLA. Cette instance est alors identifiée dans la partie de l'accord relative au contexte comme partie chargée de la supervision.

En tant qu'évaluateur de conformité, le DSLM est responsable du maintien de l'état de l'accord qu'il surveille. Il peut transmettre une violation de clause au DSLA Manager afin que ce dernier invalide l'accord et selon la politique de recours choisie y mette fin en conséquence.

Enfin le DSLM, qui supervise le respect des SLOs et rapporte les violations de clauses, applique les pénalités et déclenche les politiques de recours. Ici, nous ne souhaitons pas couvrir un ensemble conséquent de pénalités (cf. chapitre III.2.3.3). Seule la mise sur liste noire (*blacklisting*) a été validée dans l'implémentation et les expériences. Elle est déléguée au conteneur gérant les liaisons de services.

VI.3.4 Terminaison

N'importe quelle partie peut mettre fin à un accord. Que ce soit la partie tierce chargée de la supervision (SLM), le consommateur ou le fournisseur du service. Seulement les autres parties doivent alors être averties de la terminaison afin de pouvoir réagir en conséquence. Par exemple, un consommateur averti pourra alors librement changer de prestataire, ou un fournisseur pourra facturer son service au consommateur ayant rompu l'accord, voire appliquer les pénalités définies par le contrat. Enfin le gestionnaire de niveau de service pourra arrêter son activité de supervision une fois l'accord terminé.

VI.4 Gestion des liaisons par le conteneur

La gestion des liaisons de service et la communication avec la couche de gestion des accords de niveau de service se font au niveau du conteneur du modèle à composants. Nous partons du principe qu'un modèle à composant orienté services fournit au minimum les fonctionnalités suivantes.

- Pour un fournisseur de service :
  - Publication du service dans un annuaire ou d'une annonce de disponibilité.
  - Retrait du service de l'annuaire ou bien annonce d'indisponibilité.
  - La publication et le retrait de service sont exécutées respectivement à l'activation du composant et lorsque celui-ci est désactivé.
- Pour un consommateur de service :
  - Découverte passive des fournisseurs de service.
  - Injection des liaisons vers les services à consommer. Ces liaisons sont aussi appelées dépendances de services.
  - Gestion du cycle de vie du composant en fonction de la résolution de ses dépendances de service.

Sur cette base, notre proposition étend le conteneur sur les deux plans : au niveau de la livraison de service, et au niveau de la consommation de service. La Figure 48 décrit le fonctionnement de la couche SLA pour gérer les liaisons de service, ainsi que les connexions entre les parties du conteneur dédiées à la gestion des accords (enregistrement, négociation) et celles dédiées à la gestion des services (publication, découverte, liaison, retrait).

Le conteneur d'un composant fournisseur de service se charge, en plus de la publication et du retrait de service, de l'enregistrement d'une offre de SLA auprès du DSLA Manager via le service de négociation, et de son retrait. L'offre est en fait constituée des SLOs que le fournisseur peut potentiellement s'engager à respecter pour un service donné. Les capacités du fournisseur concernant sa disponibilité sont donc fournies au conteneur sous forme de méta-données. Le conteneur fait ensuite appel au service du DSLA Manager pour enregistrer cette offre si cela n'avait pas déjà été fait précédemment. Nous proposons également d'agir au niveau de la publication du service. Les capacités peuvent être elles-mêmes publiées avec le service en tant que propriétés extra-fonctionnelles. Le conteneur gère de surcroît la confirmation de terminaison des accords suite aux notifications du DSLA Manager.

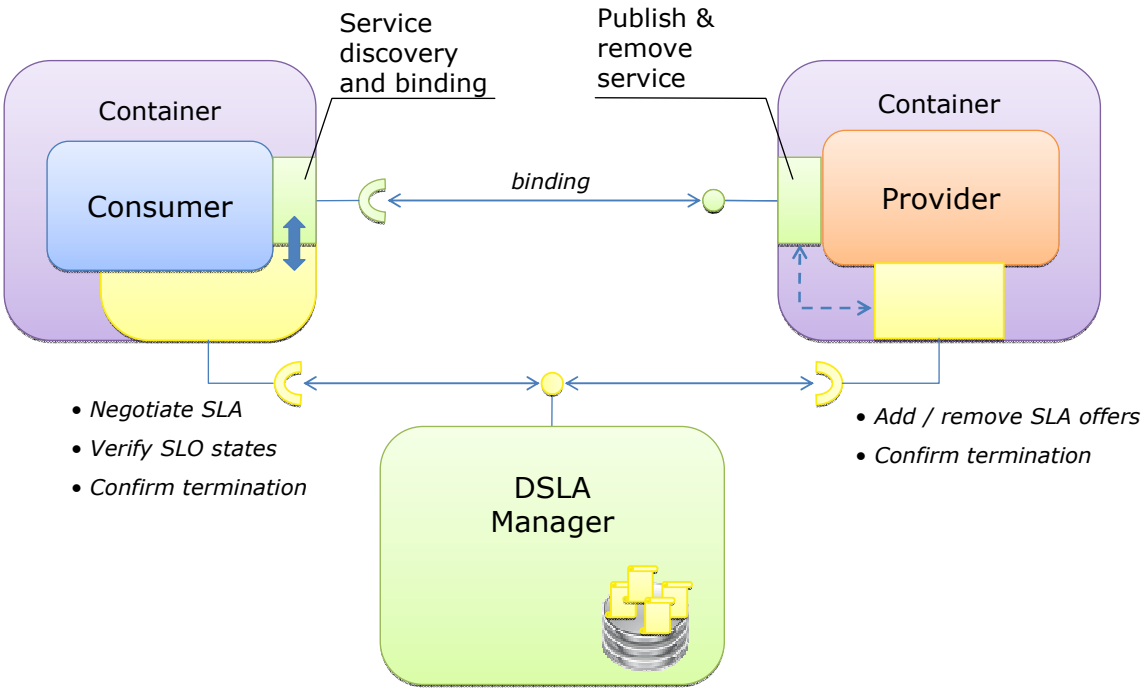

Figure 48. Utilisation du DSLA Manager par le conteneur pour la gestion des liaisons de service

Au niveau de la membrane d'un consommateur de service, la relation entre la gestion des liaisons et celle des SLAs est plus étroite. En effet, la sélection d'un fournisseur doit faire suite à la négociation d'un accord avec ce fournisseur. De plus les liaisons doivent être supprimées, substituées ou conservées selon l'état du respect des SLOs. La partie du conteneur concernant le rôle de consommateur utilise donc les services du DSLA Manager pour :

1. Filtrer les prestataires lors de la phase de découverte à partir de l'expression des besoins concernant l'intermittence de service, en plus du filtrage portant sur d'autres propriétés extra-fonctionnelles.
2. Négocier un accord de niveau de service suite à la découverte d'un fournisseur du service requis.
3. Prendre les décisions concernant la liaison de service à partir de l'état de validité du SLA. Tant que le contrat n'est pas rompu par une violation de SLO, la liaison vers le prestataire courant est conservée même si ce dernier est indisponible, et le composant n'est donc pas désactivé/stoppé. La liaison n'est détruite ou remplacée par une autre que lorsque les contraintes d'intermittence ne sont plus respectées.
4. Confirmer la terminaison de l'accord suite à une annonce du DSLA Manager.

Lorsqu'un accord n'est plus respecté par le prestataire, en fonction de la configuration du composant concernant la politique de recours à adopter, le conteneur peut en outre mettre le fournisseur responsable sur liste noire. En pratique son identifiant est ajouté au filtre de sélection afin de l'exclure des futures découvertes de service.

Troisième partie : Expérimentation et Résultats

Chapitre VII

Implémentation

“ Plus vous laissez de temps à un travail pour se réaliser, et plus il a tendance à prendre ce temps qui s'allonge pour se réaliser. ”

Cyril Northcote Parkinson

Afin de mettre en œuvre l’approche et l’architecture logicielle décrites dans le chapitre précédent, nous avons fait le choix de nous appuyer sur la plate-forme dynamique à services OSGi et le modèle à composants orientés service iPOJO.

Ces deux technologies fournissent les mécanismes nécessaires à la réalisation :

- un support des architectures dynamiques orientées services,
- et la possibilité d’étendre le conteneur des composants, notamment la partie chargée de la gestion des liaisons.

Cependant comme ces technologies ne spécifient rien en ce qui concerne les accords de niveau de service, il a fallu concevoir un module de gestion des SLAs en concentrant nos efforts sur les critères de disponibilités définis dans le chapitre IV.2.3. Cette forme de SLA est néanmoins extensible à d’autres domaines que la disponibilité, et son accessibilité convient à une vision qui se situe au niveau du composant.

Ce chapitre décrit ce qui a motivé le choix d’OSGi et iPOJO, puis comment les différentes couches constitutives de notre canevas ont été implémentées sur Apache Felix (une implémentation open-source d’OSGi R4) et Apache iPOJO.

VII.1 Choix de la plate-forme cible

Afin de mettre en œuvre une politique de liaison sensible aux interruptions, offrant un compromis entre architectures dynamiques et statiques, il est nécessaire de s'appuyer sur une plate-forme supportant l'ajout et le retrait dynamique de services. Il est aussi intéressant de réutiliser un modèle à composants existant plutôt que de redévelopper un conteneur pour l'externalisation de la gestion des liaisons.

L'équipe ADELE possède une forte expertise sur les technologies OSGi et iPOJO, ce qui a en partie motivé notre choix pour ces technologies. Bien qu'OSGi ne propose rien en ce qui concerne les SLAs, d'autres travaux se sont intéressés à la collaboration et la cohabitation multi-organisations, notamment dans le cadre de passerelle domestique multi-service [Royon2007b]. Malgré l'absence de solutions pour les SLAs, OSGi et iPOJO offrent une excellente gestion du dynamisme, ce qui aurait été compliqué à mettre en place avec des services web par exemple, en dépit de l'existence de standards pour la gestion des SLAs.

VII.1.1 OSGi

Le choix de la plate-forme à services s'est porté sur OSGi pour deux raisons principales. OSGi propose une architecture orientée service dynamique qui répond à nos besoins. La plate-forme OSGi supporte le retrait de service : les consommateurs de services peuvent s'abonner à l'annuaire (registre) afin que ce dernier les notifie des arrivées et départs des fournisseurs de service dont ils ont demandé le suivi.

OSGi présente également l'avantage d'être une plate-forme centralisée (les services sont co-localisés sur la même machine virtuelle Java), ce qui permet d'éviter les problèmes liés à la distribution dans un premier temps. En fait par la suite des équipements ou services externes pourront être représentés sur la plate-forme, soit par réification, soit par un mécanisme de ponts vers d'autres technologies à services. Dans le premier cas, un bundle fournit sous forme de service les fonctionnalités d'un équipement connecté, avec ou sans fil, à la plate-forme (par Zigbee [Girolami et al 2008], Bluetooth, USB, etc). Dans le second cas, les services exposés via d'autres technologies sont traduits localement sous forme de services OSGi (services Web, UPnP, DPWS, JINI). La distribution devra donc être gérée au niveau de la couche applicative ou par des surcouches à OSGi (ROSE[6], Apache CFX DOSGi [Bosschaert2009], R-OSGi [Rellermeyer2007], Extended-ServiceBinder [Bottaro2006]).

La seconde raison réside dans le fait que la plupart des modèles à composants orientés services (abrégé en SOCM pour *Service-Oriented Component Model*) présentés dans la section II.4.3 ciblent la plate-forme OSGi. Le but d'un SOCM est de faciliter le développement de composants orientés services et permet de s'appuyer sur un mécanisme déjà existant de gestion des liaisons dynamiques.

[6] ROSE est hébergé sur OW2 Chameleon, http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Rose

VII.1.2 Apache Felix iPOJO

Le modèle à composants orienté service retenu est Apache Felix iPOJO. Il reprend les principes de *ServiceBinder* et *Declarative Services* en les combinant à des mécanismes d'injection de dépendances très efficaces. iPOJO propose un conteneur extensible par un mécanisme de *handlers* qui peuvent être eux-mêmes des composants iPOJO. La gestion des liaisons est elle-même extensible. Il est donc aisé d'y ajouter nos préoccupations concernant les interruptions de services intermittents et la gestion de leur niveau de service via de nouveaux handlers.

L'implémentation de notre proposition consiste donc en :

- un service OSGi DisruptionLogService et un bundle implémentant le service sous la forme d'un composant iPOJO.
- un composite iPOJO chargé de la gestion des accords de niveau de service. Il est composé d'un composant de négociation, un composant de SLM et il fournit un service de gestion des SLAs ainsi qu'un service de négociation.
- deux handlers qui étendent le conteneur iPOJO en s'appuyant sur le service de gestion des SLAs. Dans le jargon d'iPOJO les handlers que nous proposons sont considérés comme des handlers externes. Ils appartiennent au même espace de nommage (*namespace*) DSLA (*fr.liglab.adele.ipojo.handlers.dsla*).

Tous ces éléments sont décrits dans les deux sections suivantes. Les détails des bundles et composants iPOJO constitutifs de notre canevas sont donnés en Annexe A.

VII.2 Gestion des accords de niveau de service

La gestion des accords de niveau de service est assurée par le DSLAAdmin. Afin que ce composant puisse créer les accords et fixer les objectifs de niveau de service, les parties identifiées de manière unique doivent lui fournir les informations nécessaires, celles-ci étant définies par le développeur du composant. Et pour être en mesure de superviser le respect de ces accords, le DSLAAdmin exploite l'historique des interruptions relevées par le DisruptionLogger.

VII.2.1 Identification unique et persistante

En premier lieu les fournisseurs de service sont identifiés par un PID (*Persistent IDentifier*) pour répondre au besoin d'identification unique et persistante des parties. La spécification OSGi définit dans ce but une propriété *service.pid* associée à une chaîne de caractères qui peut être renseignée pour chaque composant fournisseur de service. Lorsque le développeur spécifie un identifiant pour une partie prestataire de service, cet identifiant sert de PID au service publié faisant l'objet de l'accord. Par exemple, dans le cas de services UPnPDevice, de la spécification UPnPBaseDriver d'OSGi, c'est la propriété GUID qui est utilisée, et dans le cas d'équipements Bluetooth, le PID pourra être forgé à partir d'une adresse MAC.

VII.2.2 Des méta-données aux objectifs de niveau de service

Les SLOs sont générés par le composant de négociation à partir de l'intersection de l'offre d'un fournisseur et des besoins d'un consommateur. Ces derniers sont injectés dans le conteneur des composants consommateurs et fournisseurs de service à partir de leurs méta-données de configuration iPOJO.

Les méta-données d'un fournisseur vont servir à construire l'offre d'accord tandis que celles des consommateurs serviront à la négociation des offres. Les méta-données consistent en :

- des informations sur les parties : identifiant, nom et description ;
- la spécification de service (le nom de l'interface de service fournie) pour un fournisseur ;
- le champ de la dépendance de service pour un consommateur, avec un filtre optionnel pour la sélection ;
- la durée de l'accord ;
- les objectifs de niveau de service définis par les attributs suivants : nom, valeur, description, unité, relation d'ordre et extensions ;

A partir de ces informations, d'autres seront générées pour construire l'objet représentant l'accord. Par exemple la date de début de l'accord est fixée à l'issue de la négociation et la date de fin calculée d'après la durée ; l'identifiant de l'accord combine ceux des parties, etc.

VII.2.3 DSLA Admin

Le gestionnaire d'accords de niveau de service DSLAAdmin (qui remplit le rôle du DSLAManager présenté dans le chapitre précédent) est un composite iPOJO. Il est composé d'un composant singleton fournissant un service de négociation et d'un autre composant (DSLM) chargé de la supervision du niveau de service concernant les interruptions de service.

Le composant de négociation fournit un service permettant :

- d'enregistrer une offre composée d'informations sur le prestataire, de la spécification de service, de la durée de l'accord et d'une proposition de SLOs;
- de retirer une offre en précisant l'identifiant du fournisseur de service dont il faut supprimer l'annonce ;
- de négocier un accord en passant les besoins en termes de SLOs, le nom de la spécification de service concernée, une liste optionnelle de fournisseurs (leurs identifiants) avec lesquels le consommateur souhaiterait collaborer, et les informations du consommateur (nom, identifiant, description) afin de remplir le SLA en cas de négociation réussie.

Les accords négociés sont ensuite gérés par le DSLAAdmin. Les objets SLAs sont sauvegardés après la négociation. Ainsi, les accords déjà établis peuvent être chargés au démarrage du DSLAAdmin. Ensuite, l'état de l'accord est maintenu à jour selon le respect des SLOs. Et lorsqu'une partie (consommateur, fournisseur ou SLM) demande la terminaison de l'accord le DSLAAdmin passe le SLA dans l'état « terminé » et envoie une notification de terminaison via le service d'EventAdmin[7] aux parties impliquées. L'accord n'est effectivement supprimé que lorsque le DSLAAdmin a reçu confirmation de la terminaison de la part de toutes les parties.

Le composant de supervision du respect des SLOs est assigné au suivi d'un SLA en particulier. Il y a en effet une instance de composant DSLM par accord. Les instances sont créées configurées et gérées par le DSLAAdmin. A la fin de la phase de négociation d'un accord, le composant de supervision est instancié via sa fabrique et ses informations sont intégrées dans la partie du SLA relative au contexte et à la partie tierce.

Ce composant requiert le service du DisruptionsLogger puisque durant toute la durée de l'accord, le DSLM utilise le DisruptionLogService pour suivre le fournisseur de service engagé dans l'accord. A partir de l'historique des interruptions du fournisseur, il est capable de détecter la violation des trois clauses « *temps maximum d'interruption* », « *temps d'interruption cumulé maximum* », et « *temps de fonctionnement minimum entre interruptions* ».

[7] Dans la spécification R4 d'OSGi, l'EventAdmin définit un médiateur d'événements de type *publish-subscribe*.

Une fois le SLA terminé, quelque soit la raison, le DSLAAdmin détruit l'instance de DSLM attachée à l'accord.

VII.2.4 DisruptionsLogger

Le DisruptionsLogger est un composant iPOJO fournissant le service de suivi des interruptions. Il permet aux objets implémentant l'interface *DisruptionListener*:

- de s'abonner afin de demander le suivi d'un fournisseur et d'être notifié des interruptions et retours d'interruption de ce dernier ;
- de se désabonner ;
- d'accéder à l'historique des interruptions d'un fournisseur de service identifié d'après son PID.

A l'instar des autres composants proposés ici, le service (la spécification) et son implémentation sont livrées dans deux bundles indépendants.

L'historique des interruptions d'un fournisseur se présente sous la forme d'une liste dont chaque entrée est un tableau contenant les dates de début et de fin d'une interruption, par ordre chronologique. De la même manière que le DSLAAdmin rend les accords persistants, le DisruptionsLogger sauvegarde les historiques d'interruptions, et peut pour ceci reposer sur des services de journalisation standards (*LogService*) ou de persistance fournis par la plate-forme hôte. La sauvegarde des interruptions permet de prendre en compte les interruptions passées d'un fournisseur malgré les arrêts possibles de la plate-forme OSGi.

VII.3 Gestion des liaisons

Dans le cas du composant fournisseur de service, le handler iPOJO *dsla :provide* est intégré au conteneur. Il est configuré par des informations sur le fournisseur (nom, id, description), l'interface du service publié et une proposition de SLOs. Ces informations lui permettent de publier la proposition de SLOs auprès du DSLAAdmin une fois que le service correspondant à l'interface est enregistré sur la plate-forme.

Du côté du consommateur de service la gestion des liaisons se décompose en plusieurs activités.

1. Les phases de découverte et de sélection précèdent la création de la « liaison »
2. Ensuite, une nouvelle liaison peut être créée pour utiliser un nouveau prestataire.
3. Enfin, une liaison est détruite lorsque le fournisseur n'est plus disponible ou requis.

Néanmoins il faut garder à l'esprit qu'une liaison à un service peut revêtir différentes formes selon le motif d'interaction. Nous nous sommes concentrés sur la plus courante, qui est de type requête-réponse. Cependant, il existe dans la spécification OSGi d'autres patrons d'interaction, notamment pour les connexions producteur-consommateur de données et publication-souscription avec respectivement les services OSGi *WireAdmin* et *EventAdmin*.

VII.3.1 Requête-réponse

Le modèle requête-réponse est le patron d'interaction de base des services OSGi. Il correspond à des appels de méthodes Java sur une interface de service. Le connecteur n'est pas explicite. La liaison est en fait représentée par un champ du consommateur appelé dépendance de service, référençant le ou les fournisseurs de service. La valeur de ce champ est donc une référence Java vers les objets fournissant le service requis.

Les modèles à composants tels qu'iPOJO injectent la dépendance de service dans ce champ. Par un mécanisme de manipulation de bytecode, l'injection est faite à chaque fois que ce champ est utilisé pour appeler une méthode du service.

Un champ vide dénote une dépendance non résolue (il n'y a pas de liaison) et provoque l'invalidation du handler de dépendances (*DependencyHandler*) et donc du composant consommateur.

iPOJO permet d'étendre la gestion des liaisons à travers une abstraction du modèle de dépendances, le *DependencyModel*. Le handler que nous proposons à la place du

DependencyHandler utilise ce modèle pour définir un type de dépendance sensible aux interruptions basée sur les accords de niveau de service, *DSLADependency*.

Une des principales modifications apportées se situe au niveau de la gestion du cycle de vie. Ce dernier n'est plus conditionné par la disponibilité des fournisseurs de service mais par les accords de niveau de service qu'un consommateur passe avec ses fournisseurs et de l'état de ses SLAs. Ainsi notre handler *dsla:require* n'est plus invalidé quand un service est indisponible mais quand un SLO sur la disponibilité n'est plus respecté.

L'autre changement majeur concerne l'injection de dépendance. Quand le consommateur tente d'accéder à une dépendance de service, le handler intercepte l'accès grâce au mécanisme de manipulation de bytecode d'iPOJO. Il renvoie alors la valeur correspondant au fournisseur lié (i.e., ayant passé un accord avec le consommateur) s'il y en a un de disponible. Cependant lorsque la liaison est en attente du retour d'interruption (le prestataire est momentanément indisponible), le fil d'exécution est suspendu jusqu'au retour du service. Dans le cas où le SLO n'est pas respecté, soit un autre prestataire est retourné au POJO consommateur si un accord a pu être négocié, soit l'instance du composant est invalidée.

VII.3.2 Producteur-consommateur de données

Le modèle producteur-consommateur est particulièrement adapté dans le cas d'applications à base de capteurs [Marin2005]. Les capteurs produisent continuellement des données à un rythme différent de celui auquel une application pourrait les consommer. Pour cette raison, l'approche classique d'un service de type requête-réponse (*pull*) n'est pas optimale pour modéliser un capteur (qui a plutôt un comportement de type *push*).

Les producteurs et consommateurs de données s'échangent des flots de données par l'intermédiaire de connecteurs. Le producteur transmet ses données au connecteur depuis lequel un consommateur peut récupérer les données produites.

De plus, cette forme de service nécessite une approche différente de la notion de contrat de service. La partie fonctionnelle du contrat (par opposition à ses propriétés extra-fonctionnelles qui ne varient pas de l'approche plus classique) n'est plus une interface composée de méthodes. Le pivot correspond dans ce cas au type de données.

VII.3.2.1 OSGi WireAdmin

La spécification OSGi définit un modèle d'interactions producteur-consommateur, WireAdmin [OSGi2005], permettant de connecter des producteurs et consommateurs de données. Dans la terminologie de WireAdmin les connecteurs sont appelés *wires*. WireAdmin s'appuie lui-même sur des services OSGi « classiques », c'est-à-dire des services de type requête-réponse, pour modéliser les comportements de producteurs et consommateurs. Ces services identifiés de manière unique à travers leur PID sont utilisés par WireAdmin pour la transmission de données entre un producteur et un

consommateur connectés par un *wire*. Un wire est lui aussi identifié de manière unique et persistante. Il est caractérisé par les identifiants des producteurs et consommateurs qu'il relie ainsi que par un ensemble de propriétés.

Chaque composant peut produire ou consommer des données de manière active ou passive. Lorsqu'un producteur produit une donnée sur le wire, ce dernier appelle une méthode de rappel (*callback*) du consommateur en lui passant la donnée produite (*updated*). Inversement un consommateur peut demander explicitement une donnée au wire qui appellera alors une méthode de rappel du producteur (*polled*). La Figure 49 décrit la transmission de données entre un producteur et un consommateur par l'intermédiaire du wire, suivant les deux modes *push* et *pull*.

Le WireAdmin fournit un service permettant de créer, détruire, récupérer et mettre à jour des liaisons (wires) entre producteurs et consommateurs. Lorsqu'un wire est créé ou détruit, une troisième méthode de rappel est invoquée sur les producteurs et consommateurs. Elle permet d'injecter les liaisons en leur passant en paramètre la liste mise à jour des wires auxquels ils sont connectés. Lors de leur publication dans l'annuaire d'OSGi les services *Producer* et *Consumer* doivent aussi enregistrer en tant que propriété la liste de types de données qu'ils souhaitent produire/consommer. Cette liste appelée « *flavors* » permet de vérifier la compatibilité des données échangées entre un producteur et un consommateur. A cet effet, la spécification OSGi fournit des classes pour représenter diverses mesures physiques et positions (*org.osgi.util.measurement.Measurement* et *org.osgi.util.position.Position*).

Du fait de la persistance des wires, tant que ces derniers ne sont pas explicitement détruits par le WireAdmin, consommateurs et producteurs restent connectés malgré leurs possibles interruptions.

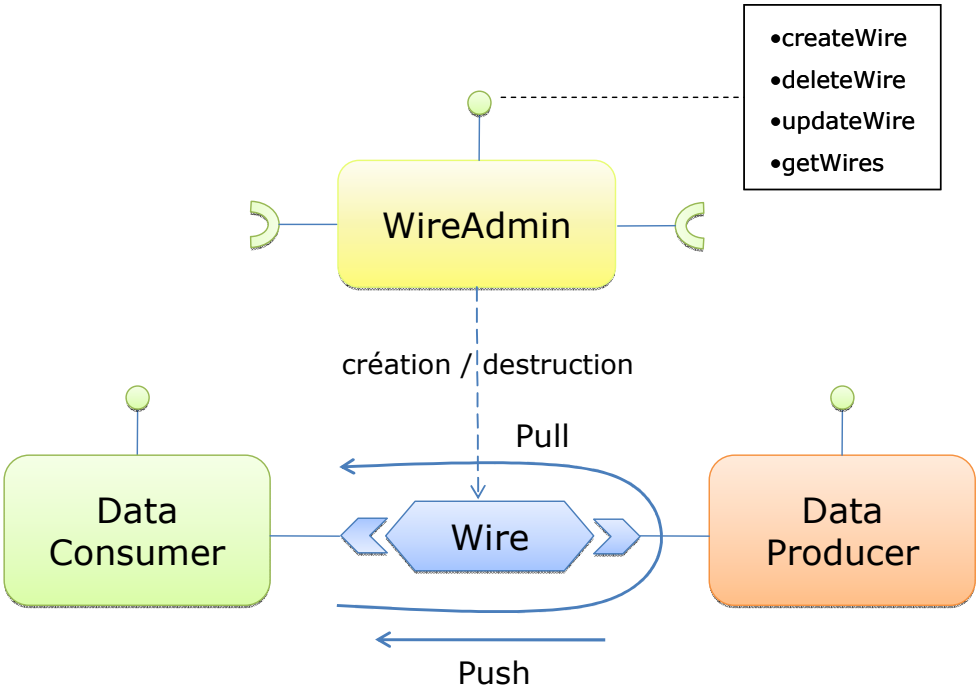

Figure 49. OSGi WireAdmin

VII.3.2.2 WireAdmin Binder

Le WireAdmin s'utilise via le service qu'il publie. La gestion des liaisons doit donc se faire de manière programmatique en utilisant ce service, ce qui détériore la lisibilité et la maintenabilité du code métier des applications. Le *WireAdminBinder* permet d'externaliser la gestion des liaisons par la définition d'un langage déclaratif de description d'architecture (ADL) dynamique, W-ADL [Touseau2008a].

W-ADL définit la notion de WireApp pour décrire le modèle d'une application constituée d'ensembles de wires (*WireSet*). Du point de vue du déploiement, une WireApp, et donc son descripteur W-ADL, est embarquée dans un bundle représentant l'application. Le descripteur est interprété par le WireAdminBinder qui pilote alors la création et destruction de liaisons (wires) via le WireAdmin. Les connecteurs sont créés et détruits dynamiquement selon les arrivées et départs des producteurs et consommateurs de données à l'exécution conformément aux WireSets.

La Figure 50 décrit le langage W-ADL. Un ensemble de wires (*WireSet*) se définit par des filtres de sélection des producteurs et consommateurs, un ensemble de propriétés, et enfin l'élément qui nous intéresse, une politique de gestion de liaison (*removePolicy*) qui indique à quel moment un wire peut être détruit.

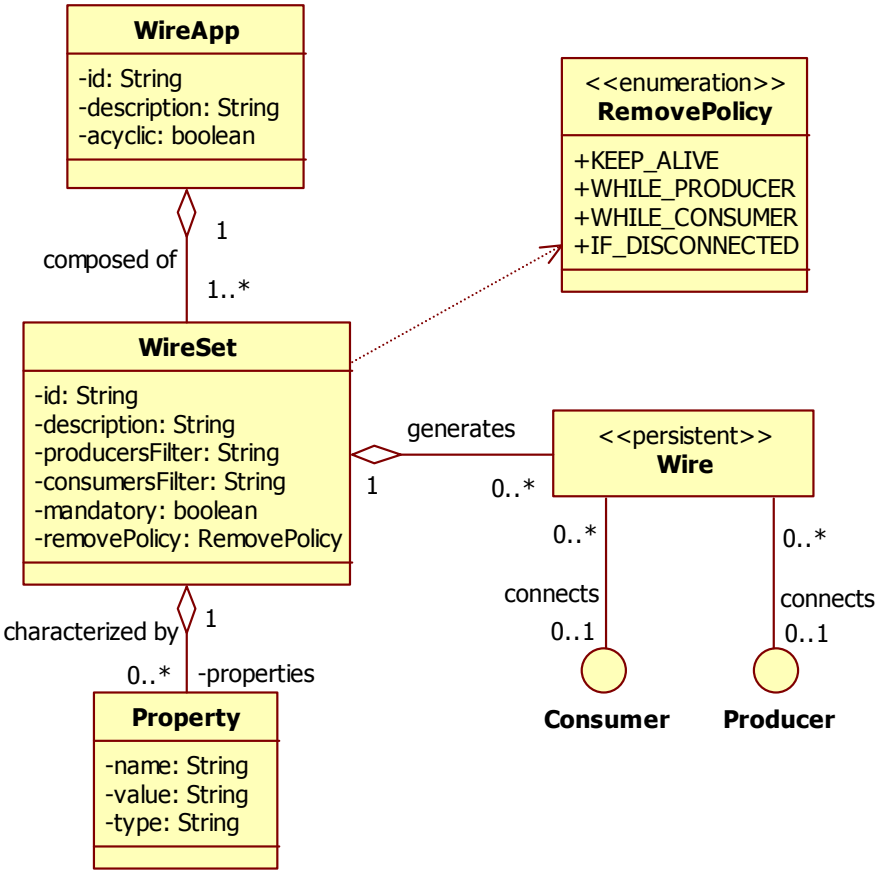

Figure 50. Langage WADL pour le WireAdminBinder

Les quatre politiques proposées ont les effets suivants :

- KEEP_ALIVE : la liaison est persistante et doit être détruite par un administrateur.
- WHILE_PRODUCER : tant qu'un producteur est connecté à un wire défini par ce WireSet, la liaison est conservée.
- WHILE_CONSUMER : idem, si un consommateur est encore connecté au wire.
- IF_DISCONNECTED : si un producteur ou un consommateur est interrompu alors la liaison est détruite.

La mise en œuvre de notre proposition pour les services intermittents de type « production de données » peut donc s'appuyer sur le WireAdminBinder et définir une cinquième politique de liaison pour le langage W-ADL. Celle-ci s'appuie alors sur les SLOs passés dans les propriétés du WireSet, et sur les informations fournies par le DSLAAdmin. Ainsi une liaison n'est détruite que lorsque le consommateur ou le producteur, suite à une interruption, ne respecte pas ses obligations définies par l'accord attaché à la liaison.

Dans le cas de liaisons de type producteur-consommateur, la gestion de la liaison passe par le WireAdmin et est ainsi indépendante des composants consommateurs. C'est ce découplage qui permet d'exprimer des contraintes bilatérales et de faire du contrôle d'admission au niveau des fournisseurs de service. Cette question de recherche est par ailleurs abordée en perspective de cette thèse.

VII.4 Outil de supervision

Afin de disposer d'un moyen de suivre les accords de niveau de service DSLA, nous proposons un outil d'administration des accords de niveau de service.

Les composants de gestion des accords de niveau de service (DSLAAdmin et DSLM) ont été rendus administrables à distance via JMX (Java Management eXtension) grâce au handler JMX pour iPOJO. La Java VisualVM[8] (anciennement jConsole) est un outil graphique permettant de superviser à distance l'exécution d'une machine virtuelle Java (JVM) et de suivre sa consommation mémoire et CPU, l'activité des threads, etc. Elle permet également de communiquer avec les applications utilisant JMX qui sont exécutées sur une JVM.

Nous proposons donc un greffon pour la JVisualVM qui permet à un opérateur de suivre à distance, grâce à JMX, l'état des différents accords signés sur une plate-forme OSGi et d'y mettre fin via un appel de la méthode *terminate*. Un second onglet permet aussi de visualiser les offres publiées par différents fournisseurs et de visualiser les interruptions de ces derniers. La Figure 51 montre deux captures d'écran donnant un aperçu de ces deux vues.

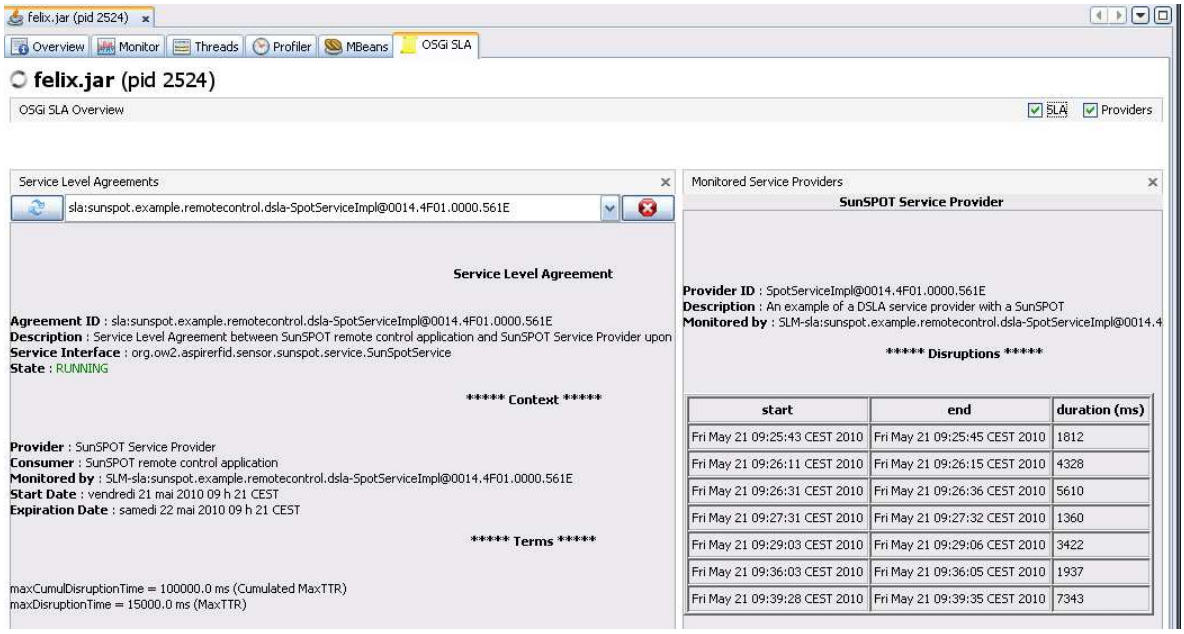

Figure 51. Captures d'écran du greffon DSLA pour la jVisualVM

[8] jVisualVM, https://visualvm.dev.java.net/

Chapitre VIII

Validation

"Tout le monde fait des erreurs, c'est pour ça qu'il y a des gommes au bout des crayons."

Lenny, dans « Les Simpson » de Matt Groening

Pour des raisons de commodité, notre proposition a été validée sur des applications utilisant déjà OSGi et iPOJO, ce qui nous a permis d'une part d'avoir des résultats rapides, et d'autre part de pouvoir comparer les performances entre l'utilisation de liaisons adaptives et de notre politique.

La validation s'est faite dans deux contextes distincts : l'informatique ubiquitaire et l'informatique à la demande.

Dans le premier cas, une expérimentation a été menée dans le cadre du projet européen Aspire ciblant l'Internet des Choses, dans un scénario de chaîne d'approvisionnement à température maîtrisée. Dans le même contexte d'informatique ubiquitaire mais dans le domaine de la domotique, nous avons validé notre approche sur la plate-forme domestique H-Omega, bâtie sur OSGi et iPOJO, et développée au sein de l'équipe Adèle dans le cadre du projet ANSO. Ce second scénario met en scène une application médicale de Maintien à Domicile (MAD).

Dans le second contexte, nous nous intéressons au serveur modulaire d'application JOnAS construit sur la plate-forme OSGi et dont la dernière version utilise iPOJO afin de fournir une offre « à la carte » et dynamique. Nous avons ainsi pu tester notre politique de liaison sur un ensemble de services techniques du serveur. Sur le serveur était déployée une application simple d'e-commerce.

Ces trois scénarios impliquent des services intermittents subissant des interruptions pour diverses raisons (mises à jour de services, coupure de courant, perte de connectivité due aux communications sans fil, ...). Les résultats tirés de ces expérimentations ont permis de valider la politique DSLA, et ainsi notre approche.

VIII.1 Méthodologie de migration

Cette première section décrit la démarche de migration de composants iPOJO vers des composants utilisant notre politique de liaison. Migrer d'une gestion dynamique des liaisons par iPOJO à notre politique DSLA s'est en effet révélée être relativement simple compte tenu de la conception d'iPOJO et de son extensibilité.

Il faut dans un premier temps déployer sur la plate-forme OSGi les bundles constituant notre canevas, à savoir: le DisruptionsLogger (spécifications et implémentation), le DSLAAdmin (spécifications et implémentation), nos deux handlers; et, s'assurer que les bundles suivants soient présents: support des composites iPOJO, handlers JMX et EventAdmin. Ensuite, il suffit de remplacer, dans les méta-données iPOJO des composants consommateurs de service, l'injection de service dans un champ identifiée par la balise requires par le *dsla:requires* pris en charge par notre handler et d'y renseigner les attributs en gras sur l'exemple suivant (les éléments soulignés sont obligatoires tandis que les autres sont optionnels) :

```
<ipojo xmlns:dsla="fr.liglab.adele.ipojo.handlers.dsla"
     xmlns:jmx="org.apache.felix.ipojo.handlers.jmx">

<component classname=
             " fr.liglab.adele.dsla.examples.app.coldchain.TemperatureProducer"
            name="coldchain_app" immediate="true">
   [...]

   <dsla:requires pid="org.ow2.aspirerfid.app.coldchain"
            name="Cold chain DSLA consumer"
            description="Cold chain application that regulates temperature">
     <require field="tempProducer"filter="(service.pid=spot-temp-prod*)">
            <slo  name="maxDisruptionTime"  value="1800000"  description="MaxTTR"
                  unit="ms" relation.order="lessThan">
                  <extension name="value.delta" value="1000"/>
            </slo>
            <slo name="maxCumulDisruptionTime" value="2400000"
                  description="Cumulated MaxTTR over one hour"
                  unit="ms" relation.order="lessThan"/>
     </require>
    </dsla:requires>

   [...]

</component>
<instance component=" coldchain_app "name="cold-chain-app-instance"/>
</ipojo>
```

La version complète des méta-données de l'application de contrôle du froid et du fournisseur du service fictif de réfrigération est présentée dans l'annexe B.

Du côté des composants fournisseurs de service, il faut ajouter une déclaration *dsla:provides* et spécifier : l'interface du service fourni (publication et retrait en fonction du cycle de vie sont laissées à la charge du ProvidedServiceHandler fourni par iPOJO), les informations sur le fournisseur (pid, nom, description) et un ensemble de slos, comme pour le consommateur.

Ce sont les seules étapes nécessaires à la migration d'applications iPOJO vers notre gestion des liaisons. Par ailleurs, migrer depuis des applications purement OSGi n'est pas plus complexe qu'une migration de ces applications vers iPOJO, seule change une partie des meta-données de configuration à renseigner.

Notre approche présente donc l'avantage d'une mise en œuvre aisée pour les développeurs iPOJO et d'un apprentissage rapide pour les développeurs OSGi.

VIII.2 Serveur d'application JOnAS

JOnAS est un serveur d'application JEE (Java Enterprise Edition) open-source. Depuis les travaux de Mikael Désertot [Desertot2007], JOnAS est construit au dessus de la plate-forme à services OSGi (Apache Felix) pour fournir une offre modulaire de serveur à la carte capable de répondre à différents besoins. Les composants techniques du serveur (gestion de la persistance, des transactions, de la sécurité, ...) sont fournis sous la forme de services. Depuis peu, JOnAS utilise en outre le modèle à composants iPOJO pour fournir et consommer ces services.

Cependant, dans la dernière version de JOnAS disponible (5.2.0_M1), toutes les dépendances de services ne sont pas gérées par iPOJO, une partie l'est directement dans le code des composants, ce qui pose problème dans notre expérimentation. En effet, même si les dépendances de service dynamiques d'iPOJO sont remplacées par des dépendances de type DSLA, si un service publié par un composant DSLA est interrompu, certains consommateurs purement OSGi seront arrêtés, ce qui impactera le socle du serveur qui ne pourra plus fonctionner. Par exemple, le service ServerProperties est utilisé par un grand nombre de composants mais le fournisseur n'est pas un composant iPOJO.

L'utilisation de politiques DSLA (ou temporelles) présente un intérêt non négligeable dans le cas de mises à jour de services techniques du serveur. Bien qu'une mise à jour ne dure généralement que quelques dizaines de millisecondes selon le service impacté, l'arrêt puis le redémarrage du service déclenchent la désactivation en cascade d'autres modules. Or lorsque certains services techniques ne sont pas disponibles les applications en cours d'exécution sur le serveur sont arrêtées, ce qui n'est pas souhaitable, en particulier pour des applications mission-critique qui doivent rester disponibles. Par exemple, le module EAR qui permet l'exécution d'applications requiert le conteneur EasyBeans qui dépend lui-même du service JMX. Le service JMX, qui de plus est requis par la plupart des composants de JOnAS, dépend du serveur de MBeans. Donc, si ce serveur est arrêté momentanément, c'est tout le serveur JOnAS et les applications qui y sont déployées qui sont impactées.

Cependant comme tous les modules ne sont pas sous forme de composants iPOJO, nous ne pouvons que supposer l'intérêt et l'utilité de notre approche dans ce cas. Et bien que nous n'ayons pas pu vérifier l'efficacité de notre politique vis-à-vis des autres existantes, nous avons néanmoins pu évaluer l'impact de notre approche sur la performance d'un logiciel de grande taille tel que JOnAS. D'autres cas d'application présentés dans les sections suivantes nous permettent de juger de la pertinence de notre approche. C'est pourquoi il n'était pas particulièrement nécessaire d'effectuer la tâche de réécriture des modules de JOnAS n'utilisant pas iPOJO.

Nous avons retenu deux composants dont les dépendances sont schématisées sur la Figure 52 :

- le DataBaseManager qui fournit le DataBaseService et requiert les services TransactionService, RegistryService, *JmxService* et ServerProperties ;
- le JonasTransactionManager qui fournit le TransactionService et requiert les services *RegistryService*, *JmxService* et *ServerProperties*.

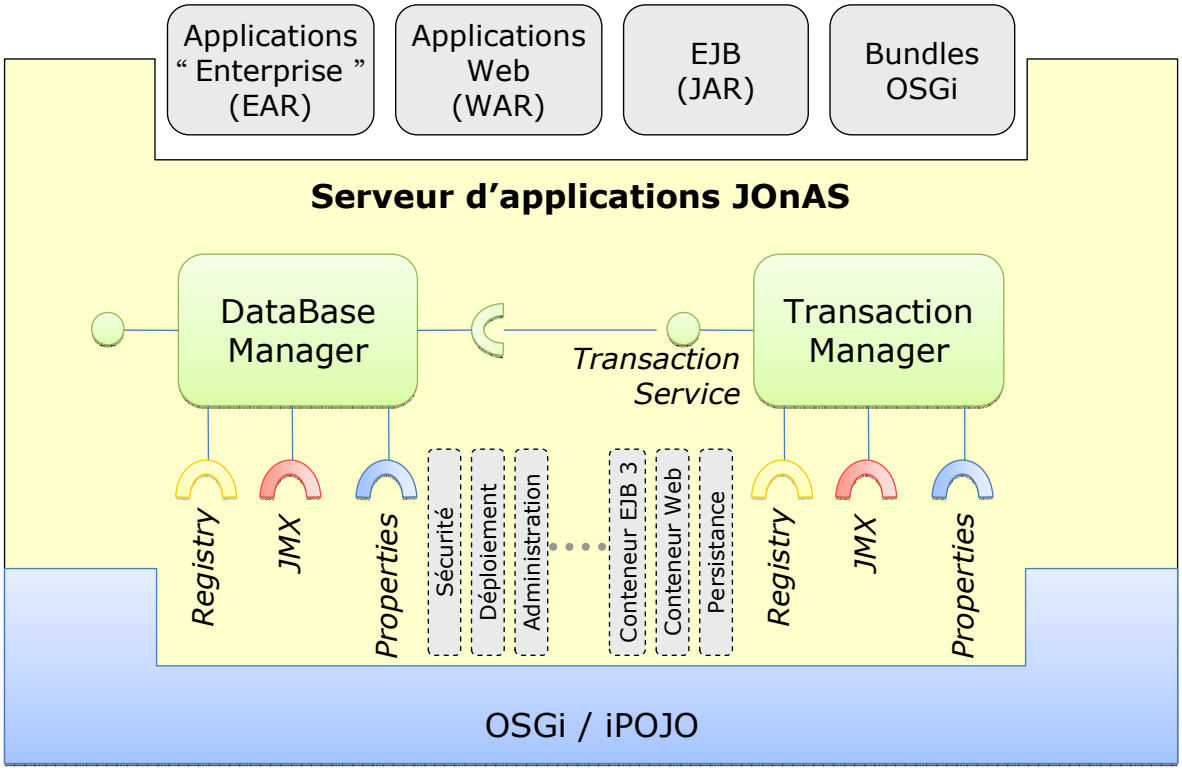

Figure 52. Deux services techniques de JOnAS : Database manager et Transaction manager

Nous avons testé trois configurations différentes de JOnAS. La première correspond à un JOnAS non modifié utilisant la politique de liaison dynamique d’iPOJO. La seconde configuration utilise une dépendance temporelle (entre le *DataBaseManager* et le *TransactionManager*), et la troisième une dépendance de type DSLA. La machine utilisée pour l’expérience possède un processeur Intel Pentium Centrino 1.73Ghz, 1Go de mémoire vive, une JVM HotSpot (version 1.6.0_15) tournant sur le système d’exploitation Windows XP Pro.

L'application témoin utilisée est une application d'e-commerce issue de la base d'exemples de Jasmine[9]. Cette application web permet à un utilisateur d’ajouter/retirer un produit à/de son panier. Nous utilisons Apache JMeter[10] pour simuler la montée en charge à raison d’un nouvel utilisateur toutes les 30 secondes jusqu’à un total de 10 utilisateurs (charge faible). Chaque utilisateur effectue 6 requêtes, 500 fois de suite (ajout d’un CD dans le panier, ajout d’un DVD, retrait du DVD, ajout d’un second CD puis retrait des deux CD), pour un total de 30000 échantillons.

Dans un premier temps, nous avons observé des temps de démarrage légèrement supérieurs avec la 3ème configuration (cf. Tableau 8). Ce délai supplémentaire est dû à l’installation et au démarrage de ressources supplémentaires (support des composites iPOJO, JMX handler, EventAdmin handler, handlers DSLA, DisruptionsLogger et DSLAAdmin), ainsi qu’au temps de négociation et de création de l’accord de niveau de service correspondant à la liaison. La différence est toutefois minime : le délai est

[9] Outil d’administration OW2 JASMINe, http://wiki.jasmine.ow2.org/
[10] Apache Jakarta JMeter, http://jakarta.apache.org/jmeter/

relativement petit devant le temps de démarrage. La seconde configuration (temporelle) résulte en un démarrage 1,0472 fois plus lent (surcoût de 4,72%), tandis que dans la troisième configuration (DSLA) le démarrage est 1,0846 fois plus lent (surcoût de 8,46%).

En revanche, la différence de temps de réponse (troisième colonne du Tableau 8) n'est pas significative. Le mécanisme d'injection de dépendance de service étant le même dans les trois configurations (i.e., interception du champ de service par iPOJO), les résultats obtenus sont similaires.

Configuration	**Temps de démarrage** (moyenne sur 10 démarrages)	**Temps de réponse** (sur 30000 échantillons)
Normale (dynamique)	18,857 secondes	moyen : 15 ms médian : 13 ms max : 63 ms min : 9 ms
Temporelle	19,748 secondes	moyen : 16 ms médian : 13 ms max : 90 ms min : 9 ms
DSLA	20,453 secondes	moyen : 15 ms médian : 14 ms max : 64 ms min : 9 ms

Tableau 8. Temps de démarrage et de réponse du serveur JOnAS selon la politique de liaison utilisée

VIII.3 AspireRFID ~ Chaine d'approvisionnement à température maitrisée

Le projet ASPIRE[11] (*Advanced Sensors and lightweight Programmable middleware for Innovative Rfid Enterprise applications*) est un projet de recherche FP7 financé par l'Union Européenne dans le but de promouvoir l'adoption de la technologie RFID par les PMEs. Dans ce but, Aspire propose une plate-forme logicielle composée d'un intergiciel open-source léger, adaptable, conforme aux standards définis par des consortiums comme EPCglobal[12] (EPC signifiant *Electronic Product Code*), NFC Forum[13], JCP et l'OSGi Alliance. Aspire propose en outre divers outils pour faciliter le développement, le déploiement et l'administration d'applications RFID et à base de capteurs.

Dans cette validation nous nous intéressons à l'intergiciel AspireRFID qui intègre la Suite RFID développée par l'équipe Adèle depuis 2005. La Figure 53 présente l'architecture d'AspireRFID dans un contexte de chaîne d'approvisionnement. Les données sur les objets et les mesures qui leurs sont associées sont collectées et filtrées au niveau des edges. Un edge fait office de passerelle entre les objets physiques qui produisent des données (principalement des lecteurs RFID et des capteurs) et les systèmes d'information traitant ces données : les EPCIS (EPC Information Services). Entre ces deux couches, l'intergiciel peut optionnellement intégrer des serveurs intermédiaires, appelés *premises*, servant également à collecter et filtrer les données de plusieurs edges afin de diminuer leur charge. Les premises reçoivent les données et les retransmettent sous forme de rapports ALE (*Application Level Event*, un standard défini par EPCglobal). En fin de chaîne, les systèmes d'information EPCIS stockent les données et fournissent des services d'administration permettant de suivre les objets ou les biens identifiés, et les mesures qui leur ont été associées (GPS pour la géo-localisation, température pour le suivi de la chaîne du froid, etc). Enfin, les différents EPCIS peuvent partager des informations via un ONS (*Object Naming Service*), un équivalent du DNS pour les objets. Chaque couche de l'intergiciel (edge-premise-epcis) est administrable via JMX, et bien que les lecteurs RFID soient configurables, l'administration de capteurs n'est pas spécifiée par la Suite et peut être traitée par d'autres intergiciels [Gürgen et al 2008].

D'un point de vue technique, les edges sont généralement de petites machines capables d'héberger une plate-forme OSGi tandis que les EPCIS sont déployés sur des serveurs JEE JOnAS. Pour la validation de notre approche nous nous plaçons au niveau de l'edge. Basé sur OSGi et iPOJO, un edge peut héberger des applications tierces en plus de l'intergiciel. En effet, la cohabitation de l'intergiciel et d'applications tierces est rendue possible grâce au paradigme de l'approche orientée service mise en œuvre par OSGi. La Figure 54 représente les composants d'un edge dédiés aux tâches de l'intergiciel, et, en pointillés, les applications tierces indépendantes de l'intergiciel ayant des tâches spécifiques, et tirant partie des données produites par les capteurs et lecteurs.

La Suite RFID a déjà été validée dans divers scenarios dont la gestion de chaine d'approvisionnement pour suivre le transport de produits [Kefalakis et al. 2008], le suivi d'un ballon sonde [Jean et al 2009], l'enrichissement de l'expérience de visite d'un musée en utilisant un téléphone NFC [Rudametkin et al 2010].

[11] Projet européen ASPIRE, http://www.fp7-aspire.eu
[12] EPCglobal, http://www.epcglobalinc.org
[13] NFC Forum, http://www.nfc-forum.org

Nous avons choisi de valider notre approche dans un scenario de chaîne d'approvisionnement à température maîtrisée. Il s'avère en effet nécessaire d'avoir une température maîtrisée dans certains cas. Notamment dans le cas de conservation de produits à basse température par exemple, où une rupture de la chaîne du froid entraîne le développement de bactéries et ainsi la perte des produits alimentaires ou pharmaceutiques ; ou encore dans le cas de la conservation et du transport de vin qui a besoin de conditions de température et de luminosité optimales pour développer ses qualités gustatives. C'est typiquement le genre d'application tierce qui peut être déployée au niveau edge d'AspireRFID.

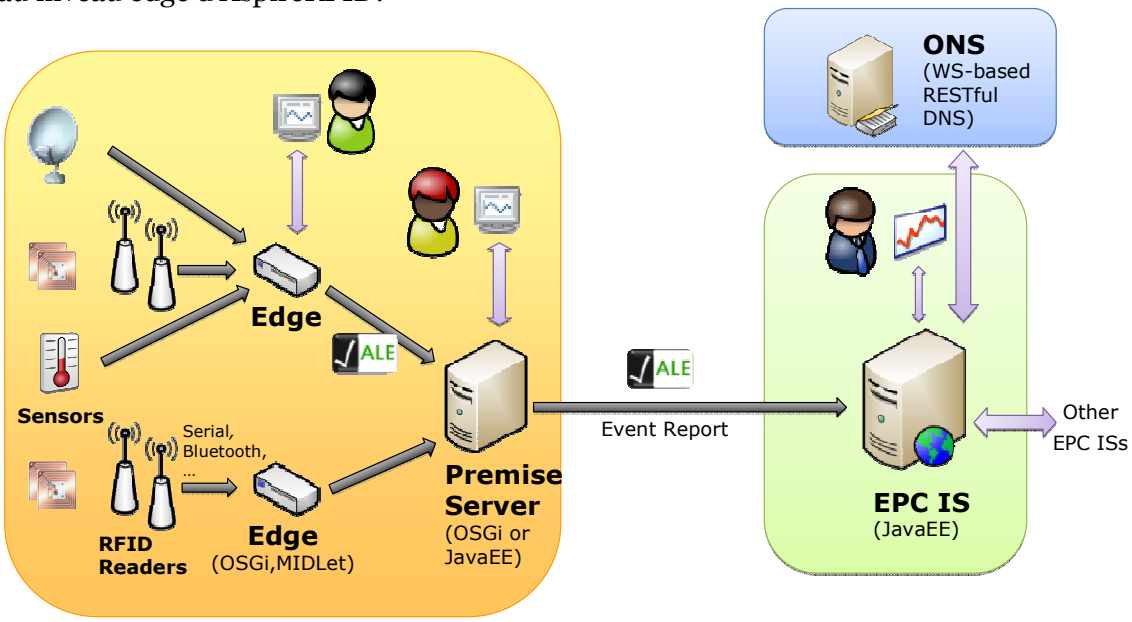

Figure 53. Architecture de l'intergiciel AspireRFID pour une chaîne d'approvisionnement [Donsez2009b]

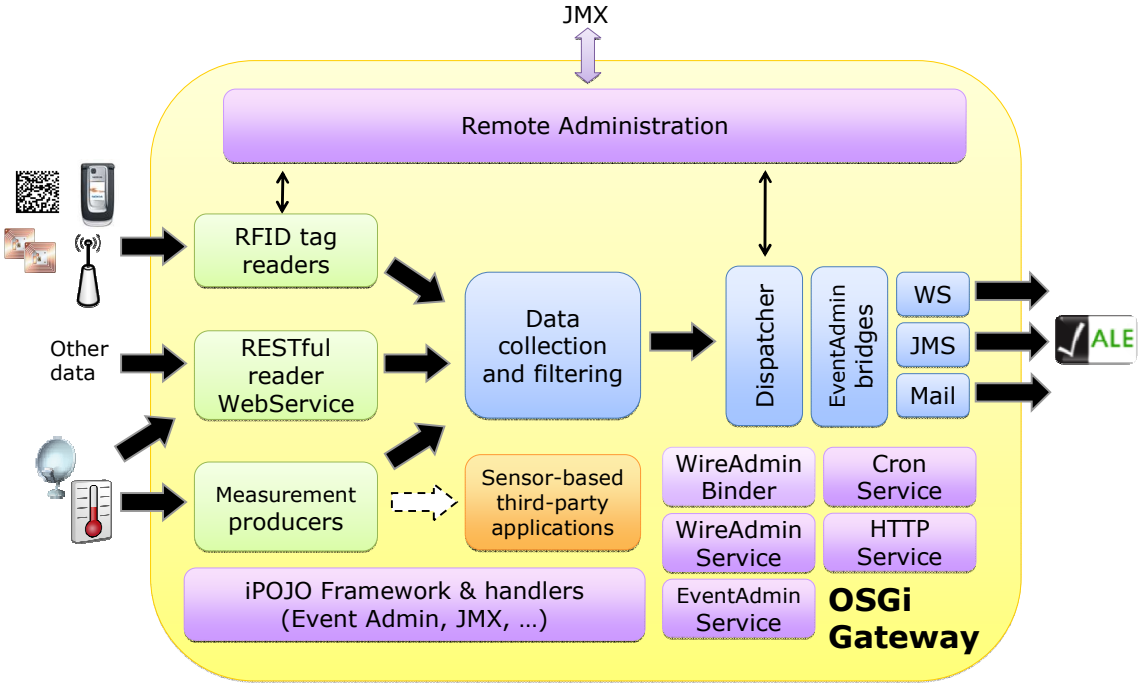

Figure 54. Composants d'un edge AspireRFID déployé sur une passerelle OSGi

Pour expérimenter notre approche, nous nous plaçons dans le contexte de conservation de produits pharmaceutiques thermosensibles (vaccins, médicaments, produits sanguins, organes). Notre application utilise des capteurs de température pour réguler la température d'une enceinte thermostatique, et pour assurer le suivi de la température afin de prouver le respect de la chaîne du froid. Dans ce cas, il est possible que le système de réfrigération subisse une coupure de courant temporaire à la suite de laquelle les produits sont généralement éliminés par précaution. Or le réchauffement des produits n'est pas instantané, l'interruption du système doit donc pouvoir être tolérée si elle est de courte durée, ce qui justifie l'intérêt de notre politique.

La Figure 55 décrit l'architecture de l'application de contrôle du froid. En plus de contrôler et d'enregistrer la température de l'enceinte grâce à la Suite RFID, l'application joue le rôle de thermostat grâce à un dispositif de régulation active. Un capteur de température fournit un service de mesure. Ici nous avons utilisé le capteur d'un SunSPOT. Un SunSPOT[14] est un petit objet programmable en Java pouvant communiquer par le protocole ZigBee, et qui embarque des capteurs de température (allant de 0 à 45°C en fonctionnement), de luminosité, un accéléromètre à 3 axes, deux interrupteurs et 8 LED RVB. Le service de mesure est utilisé chaque minute par l'application thermostat afin de réguler la température du dispositif de réfrigération (qui se présente également sous forme de service) pour adapter la température de l'enceinte et la maintenir entre +2° et +8°C. Pour les besoins de notre application, le service de réfrigération est fictif et le capteur de température a été placé dans le réfrigérateur de la cafétéria. L'application se comporte de plus comme un producteur de mesures de température afin de les injecter dans l'intergiciel AspireRFID. L'intégration avec l'intergiciel se fait par publication d'un service producteur WireAdmin et par la déclaration d'une WireApp pour WireAdminBinder. Le descripteur de cette WireApp est fourni en annexe C.

La température est affichée sous forme de code par les LED d'un SunSPOT placé à l'extérieur de l'enceinte frigorifique. Le nombre de LED allumées en vert indique la température de l'enceinte. Si les seuils de température maximale (+8°C) ou minimale (+2°C) sont dépassés, les LEDs deviennent orange puis clignotent en rouge lorsque la température atteint le seuil critique de +10°C (ou 0°C). L'inertie thermique implique un certain temps avant que les produits ne se réchauffent réellement, la montée de température n'est pas immédiate. Nous supposons que ce temps est de 30 minutes, et le traduisons par un temps d'interruption maximum de 1800000 millisecondes dans les SLOs.

Lorsque l'application de contrôle de la chaîne du froid est arrêtée (suite à l'indisponibilité du service de température ou du système de réfrigération, par exemple), un service d'alerte est utilisé pour avertir que la température de l'enceinte n'est plus assurée et que les produits qui y sont stockés doivent être éliminés. Ce service d'alerte est fourni par une lampe d'alerte[15] qui passe du vert au rouge en cas d'alerte.

Le premier scénario consiste à simuler une coupure de courant au niveau du système de réfrigération, ce qui résulte en une interruption de ce service, tandis que dans le second scénario, le capteur de température alimenté de façon autonome tombe à court d'énergie. Sans données sur la température de la chambre froide, le principe de précaution préconise l'élimination des biens qui y sont stockés. Cependant, toujours en raison de l'inertie

[14] SunSPOT, http://www.sunspotworld.com/

[15] Dans notre expérience le service d'alerte est fourni par un avatar i-Buddy (http://www.i-buddy.com) dont la tête change de couleur.

thermique, l'absence de mesures peut-être tolérée temporairement. Dans les deux scénarios, nous simulons une première interruption de durée limitée (5 minutes), et une seconde de durée critique (supérieure à 30 minutes).

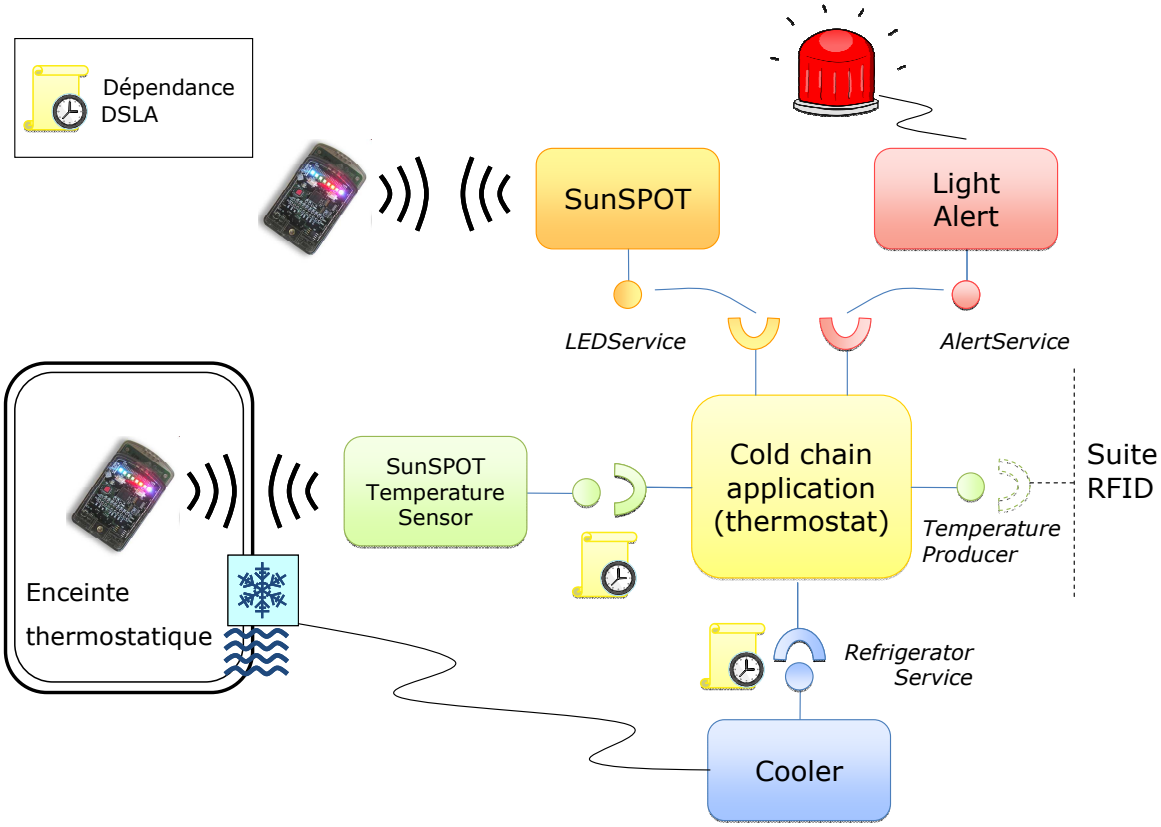

Figure 55. Application de contrôle du froid

L'expérimentation consiste en une observation du comportement de l'application selon la politique de liaison utilisée entre l'application de contrôle du froid et les services utilisés. Les résultats sont présentés dans le Tableau 9.

Dans le cas de la politique optionnelle, l'application continue à fonctionner, et ce même après une interruption du système de réfrigération. Lorsque le service de mesure de température lié au SunSPOT n'est plus disponible, c'est la dernière mesure relevée qui est envoyée à la Suite RFID. Cependant comme l'application continue à fonctionner même lorsque ses dépendances ne sont pas résolues, aucune alerte n'est déclenchée. De plus, utiliser des dépendances optionnelles pour la tâche principale de l'application (i.e., la régulation de la température) n'est pas cohérent.

La politique statique ne permet pas le retour du service de réfrigération une fois que celui-ci a été interrompu. L'alerte est donc déclenchée à la première interruption d'un des services, et l'application est arrêtée et devra être redémarrée par un administrateur.

Dans le cas de dépendances de services gérées par une politique de liaison dynamique, l'alerte est émise dès l'interruption du SunSPOT ou du service de réfrigération qui provoquent l'arrêt de l'application. En plus de l'émission d'une alerte possiblement prématurée, la désactivation de l'application implique également l'arrêt de production de mesures pour la suite RFID.

En revanche, en utilisant notre politique DSLA, les interruptions de ces services sont tolérées et l'application n'est pas désactivée. Par conséquent elle ne déclenche pas d'alerte et continue à transmettre les températures à la Suite RFID si les temps d'interruptions ne dépassent pas les bornes définies par le contrat.

Dépendance	Alerte	Suivi de la température
Optionnelle	Aucune, car l'application n'est pas arrêtée	Scénario 1 : ininterrompu Scénario 2 : envoie la dernière valeur relevée en cas d'interruption
Statique	Dès la première interruption d'un des services. L'application est ensuite arrêtée définitivement	Arrêté à la 1ère interruption d'un des services
Dynamique	A chaque interruption de service	Arrêté lorsqu'un des services est interrompu
DSLA	Après le dépassement du seuil temporel fixé par le contrat	Scénario 1 : ininterrompu Scénario 2 : envoie la dernière valeur avant interruption Arrêté lorsqu'un SLO est enfreint

Tableau 9. Observation de différentes politiques de liaison dans une application de contrôle du froid

Des résultats similaires sont obtenus avec la politique temporelle, excepté dans le cas où les interruptions sont fréquentes. Par exemple, si le service de réfrigération subit trois coupures de courant de 20 minutes espacées de 2 minutes chacune, le cumul des durées d'interruptions n'est pas pris en compte alors que le système dysfonctionne. Notre politique DSLA s'appuie également sur des SLO portant sur le cumul, et permet ainsi de parer à ce genre de situation.

L'utilisation d'accords de niveau de service par rapport à une autre politique de liaison temporisée n'a pas joué un rôle primordial ici, mais elle peut se révéler intéressante d'un point de vue juridique dans le cas où la solution de contrôle de la température est assurée par une organisation tierce. La rupture de l'accord et l'enregistrement des dépassements servent en effet de preuve du non-respect de la chaîne du froid.

Les résultats de l'expérience sont également visibles sur les diagrammes a et b de la Figure 56, issues de la console d'administration de l'intergiciel d'Aspire. Dans le premier cas, une politique de liaison dynamique a été utilisée. On observe alors une absence de mesures pendant l'interruption du service de réfrigération. En revanche, sur le second diagramme correspondant à une liaison « DSLA », l'émission des mesures de températures à la Suite RFID n'a pas été perturbée par l'arrêt du réfrigérateur, et on constate bien une hausse progressive de la température de l'enceinte thermique.

Du point de vue des contraintes techniques, la communication avec le SunSPOT se fait par les airs et à travers les parois solides du réfrigérateur, et a ainsi causé des pertes de messages d'où des interruptions de service fréquentes. Ce phénomène caractéristique des services intermittents renforce d'autant plus l'intérêt d'utiliser une politique tolérante aux interruptions. De plus, pour simuler l'arrêt du service de réfrigération, simplement sortir le SunSPOT du réfrigérateur augmentait trop rapidement sa température, il a donc fallu le placer dans un sac isotherme.

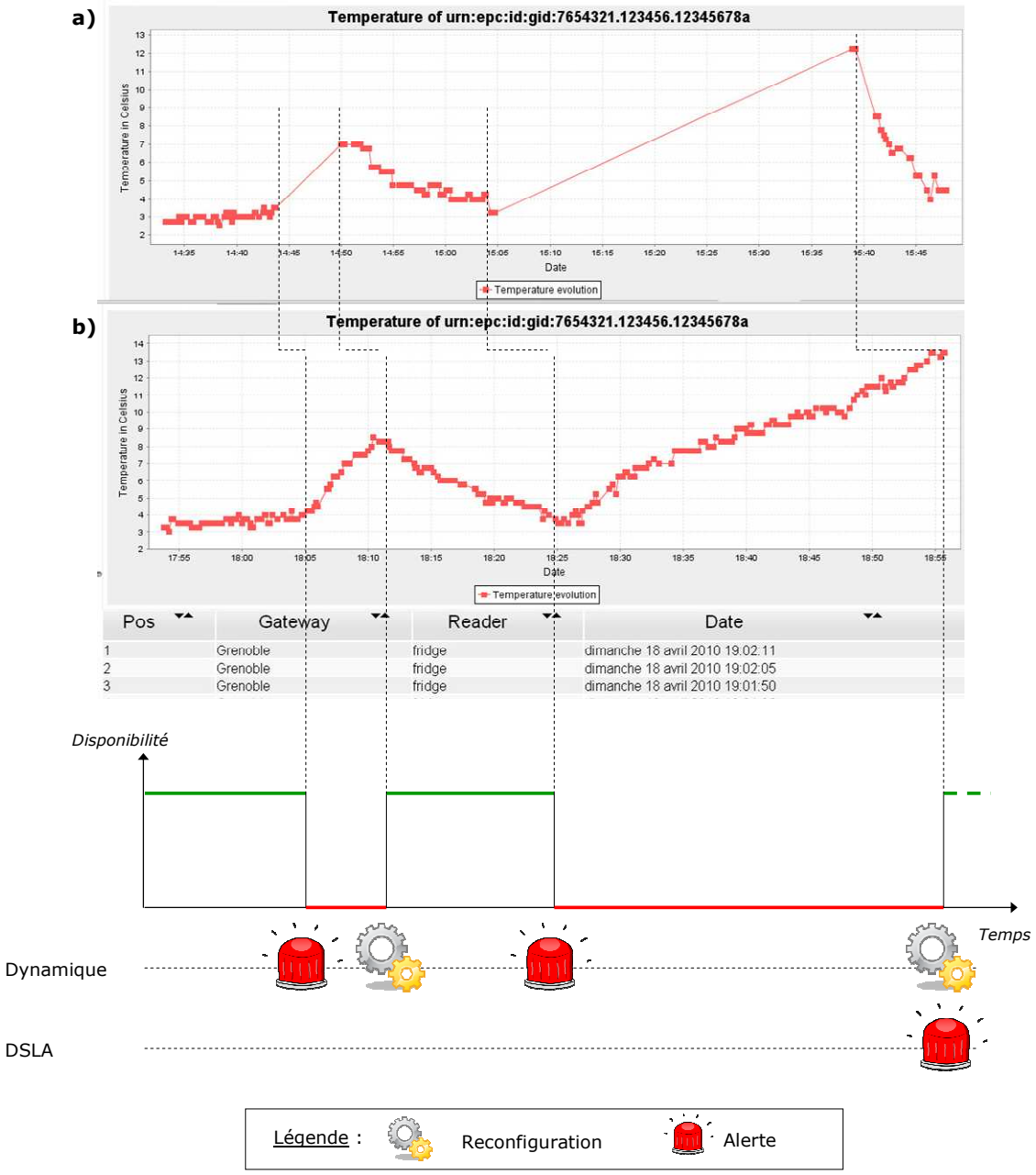

Figure 56. Courbes de suivi de la température

VIII.4 Passerelle domestique H-Omega

La passerelle résidentielle H-Omega [Bourcier2007], qui est issue du projet européen ANSO[16], a pour objectif la livraison de services domotiques dans la maison. Un des intérêts principaux d'H-Omega réside dans sa capacité à intégrer des services et dispositifs hétérogènes (cf.

Figure 57). Pour cela, la passerelle utilise un composant appelé Remote Service Manager, devenu aujourd'hui le gestionnaire de services distribués ROSE. Le serveur d'applications domestique H-Omega est construit sur la plate-forme OSGi et le modèle à composants iPOJO.

Les applications déployées sur H-Omega peuvent concerner aussi bien le contrôle de la maison (réglage de la lumière, ouverture et fermeture des volets, verrouillage des portes, etc) que des services externes (service web météo agrégé sur la passerelle) ou des applications spécifiques à un domaine tel que le divertissement multimédia (utilisant des équipements UPnP AV par exemple) ou l'hospitalisation et le maintien à domicile (HAD et MAD).

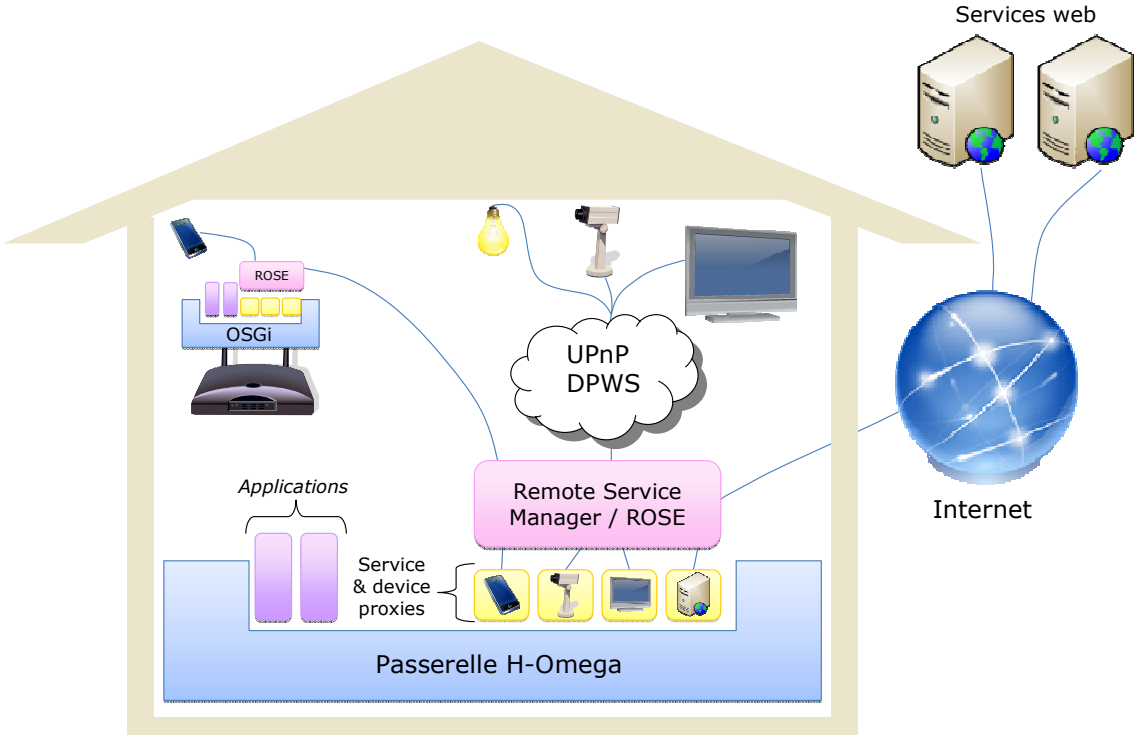

Figure 57. H-Omega agrège des services distribués grâce au Remote Service Manager.

[16] Projet européen ITEA ANSO, http://anso.vtt.fi/

VIII.4.1Remote Service Manager

Le *Remote Service Manager* de H-Omega utilise un composant appelé *Proxy Selector* qui définit la stratégie de sélection des mandataires (*proxy*) découverts. Si l'administrateur souhaite modifier la stratégie, il doit modifier le composant *Proxy Selector* et mettre à jour le bundle correspondant. Par exemple s'il est nécessaire d'économiser l'espace disponible sur la passerelle, il est préférable de suivre une stratégie qui prône la réutilisation de mandataires existants déjà présents sur la passerelle.

Toutefois, l'interruption du *Proxy Selector* déclenche l'arrêt du *Remote Service Manager*, puisque ce dernier requiert le service du *Proxy Selector* auquel il est lié par une politique de liaison dynamique. Or si le *Remote Service Manager* est arrêté, le composant gérant les instances de service « mandataires » est lui aussi désactivé et détruit les instances de mandataires. Par conséquent, les applications utilisant des services distants (externes à la passerelle) seront elles-aussi temporairement désactivé, le temps que le *Remote Service Manager* soit de nouveau disponible et que les mandataires soient recréés et republient leurs services. Dans cette situation, l'utilisation d'une politique tolérante aux interruptions (temporelle ou DSLA) est particulièrement recommandée.

VIII.4.2 Application médicale

Nous avons également validé notre approche dans un scénario multi-fournisseurs où la substitution n'est pas souhaitable en cas d'interruption. Une application médicale d'HAD (Hospitalisation à Domicile) ou de MAD (Maintien à Domicile) permet d'assurer le suivi des patients et d'envoyer des rapports périodiques au médecin en charge du patient. L'application décrite Figure 58 utilise les données issues de capteurs médicaux (montre cardiogramme-podomètre, glucomètre, accéléromètre de SunSPOT pour le suivi d'activité, etc) pour générer un rapport médical. Les données sont sauvegardées à intervalles réguliers, puis intégrées au rapport qui est transmis périodiquement à l'hôpital (toutes les 12 heures).

Pour cela, la passerelle H-Omega propose un gestionnaire de persistance, le *Persistence Manager* basé sur le canevas Apache Cayenne. Ce gestionnaire instancie (en utilisant le patron de conception *Extender Model* d'OSGi) des services de persistance *ObjectContext* propres à une classe/modèle de données, ici une classe de représentation d'un rapport médical. Ces services peuvent être liés à différentes sources de données. Dans l'exemple traité, deux instances de service de persistance propre au rapport médical sont disponibles. L'une est connectée à une base de données MySQL distante, tandis que l'autre utilise le mécanisme de sauvegarde locale *EmbeddedDriver* fourni par Apache Derby.

Par souci de cohérence il est préférable de stocker toutes les données dans la même base, puisque les informations sont récupérées au moment de la génération du rapport sont. Donc si la base de données distante n'est plus disponible, plutôt que d'effectuer une substitution et sauvegarder les informations sur l'autre source de données, notre politique DSLA privilégie la relation client-fournisseur et conserve la liaison avec BDD distante. La sauvegarde est simplement « mise en pause », et une fois que la BDD redevient disponible, les données sont persistées. Toutefois, si le service de persistance lié à la BDD distante ne revient pas dans un délai raisonnable, l'application utilisera le second service. De plus, cette stratégie permet d'économiser l'espace disponible sur la passerelle, qui est souvent une ressource limitée.

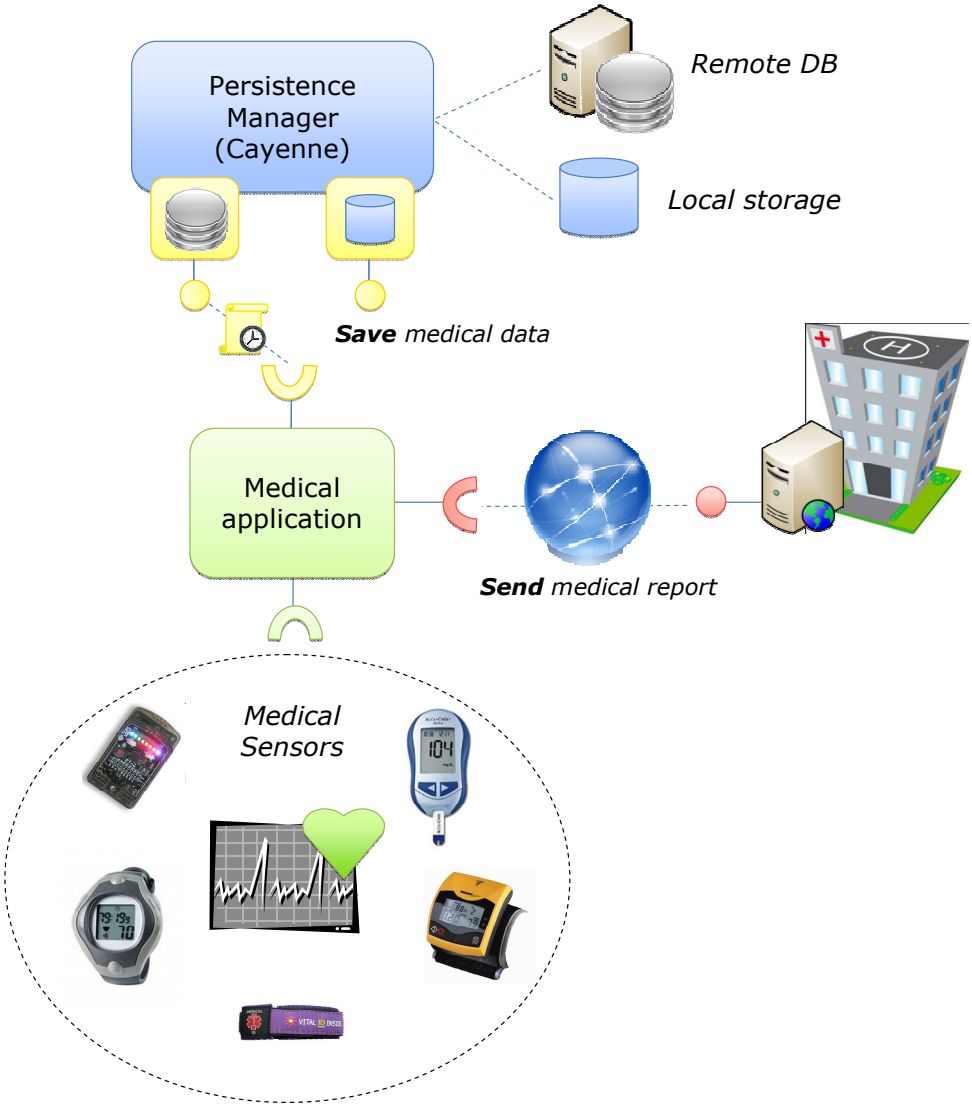

Figure 58. Application médicale utilisant la persistance d'H-Omega

Notre politique est également intéressante au niveau du service utilisé pour l'envoi des rapports médicaux. Selon l'implémentation du service, l'envoi peut se faire par mail, ou en utilisant un service web de l'hôpital pour que les données soient intégrées automatiquement au système d'information de ce dernier. Or le service web utilisé pour l'envoi de rapports est lié à un médecin, un service ou un hôpital particulier. Par conséquent, si un médecin n'est pas disponible, le rapport peut être envoyé à un de ses collègues, ou à un autre service/hôpital, c'est-à-dire à un autre fournisseur de service. C'est effectivement ce qui se passerait avec les politiques dynamiques ou temporelles d'iPOJO qui reconfigureraient les liaisons de l'application médicale. Cependant, comme le rapport n'est émis qu'à des intervalles de temps espacés de 12 heures, une interruption entre ces intervalles n'est pas gênante et il est donc inutile de modifier le destinataire des rapports.

Chapitre IX

Perspectives et Conclusion

"*L'échec est la voie du succès; chaque erreur nous apprend quelque chose.*"

Morihei Ueshiba, fondateur de l'Aïkido

Dans ce document nous avons présenté une approche basée sur les accords de niveau de service pour la mise en œuvre d'une politique d'adaptation tolérante aux interruptions de services. L'idée directrice de cette proposition est d'offrir un compromis entre la stabilité de l'approche statique et la capacité d'adaptation offerte par l'approche dynamique.

Ce chapitre de conclusion est l'occasion de revenir sur le travail mené durant cette thèse et d'y porter un regard rétrospectif. Avec le recul, les points forts, mais aussi les limites, de ce travail apparaissent plus clairement, et plusieurs perspectives de recherche se dessinent. Ces perspectives se placent aussi bien dans la continuité directe de cette thèse que sur des thèmes connexes abordés au fil des réflexions.

Il faut également garder à l'esprit qu'au commencement de cette thèse des éléments décrits dans ce manuscrit n'existaient pas encore et sont apparus en cours de route, voire à la fin. C'est le cas des politiques de liaisons temporelles qui poursuivent un objectif similaire au notre, ou encore des contextes applicatifs tels que le *cloud computing* ou l'*informatique verte* qui ont confirmé la réalité de la problématique abordée, et donc l'intérêt de notre proposition.

IX.1 Perspectives

Au terme de cette thèse, il reste plusieurs points que nous avons abordés mais qui peuvent encore être approfondis, et des voies que nous n'avons pas pu nous permettre de suivre mais qui mériteraient d'être étudiées par la suite.

IX.1.1 Dans la continuité

La première catégorie de perspectives concerne les extensions possibles de nos travaux. Certains points ont été peu ou pas traités par manque de temps ou de ressources. Par exemple, l'expression des objectifs de niveau de service aurait pu être traitée plus finement, mais comme ce n'était pas le point principal de notre contribution, nous n'avons retenu que trois critères qui nous ont permis de valider l'approche.

IX.1.1.1 Modélisation des accords

La modélisation des objectifs de niveau de service portant sur la disponibilité est en effet un point qui pourrait être amélioré.

Premièrement, il est des cas où les trois critères que nous avons retenus ne suffisent pas. Bien que ce ne soit pas caractéristique des services intermittents, il est possible qu'un service ne soit utilisé qu'une seule fois par une application, lors d'une phase d'initialisation par exemple. Dans ce cas, une fois le service utilisé, l'interruption du service ne devrait pas déclencher de reconfiguration. De plus, même si nous avons abordé la nécessité de fenêtres temporelles pour qualifier des temps de fonctionnement, l'implémentation proposée ne permet pas de prendre ce type de paramètre en compte.

Deuxièmement, la sémantique des objectifs de niveau de service n'a pas été définie, et pour le moment l'appariement de consommateurs et de fournisseurs ne peut se faire que si les deux utilisent la même sémantique. Par exemple, un consommateur ayant besoin d'un service disponible « 20 heures par jour » devrait pouvoir utiliser celui d'un prestataire annonçant des interruptions « inférieures à une heure sur 24 heure ».

Enfin, bien que la fonctionnalité ne soit pas encore implémentée, il est également envisagé de traduire les accords générés par le DSLAAdmin vers le langage WS-Agreements afin de rendre les DSLA compatibles avec ce standard.

IX.1.1.2 Temps-réel

Les mesures de temps effectuées par notre gestionnaire de service ne sont pas faites en temps-réel. Leur exactitude ne peut donc pas être garantie puisque le temps-réel est le

seul moyen d'obtenir des mesures fiables. Cependant le portage de notre solution sur une machine virtuelle Java temps-réel (implémentant RTSJ[17]) demeure une perspective à long-terme.

Par ailleurs, le principe de seuil de tolérance temporel nous rapproche des problématiques du temps-réel et du concept d'échéance (*deadline*). Nous pensons qu'il serait en effet intéressant d'appliquer notre approche à des applications temps-réels critiques. Le principe consisterait à tolérer des interruptions de service (ce qui est généralement prohibé dans ce genre d'applications) tant que les objectifs de niveau de service annoncés restent compatibles avec l'échéance de la tâche à exécuter.

IX.1.1.3 Logique temporelle

Lorsqu'il sera nécessaire d'exprimer des contraintes temporelles plus complexes, il sera sans doute plus intéressant d'utiliser des règles de logique temporelle au niveau du gestionnaire de niveau de service ayant la charge d'évaluer le respect des objectifs. Le terme de logique temporelle couvre les approches de représentation des informations temporelles au sein d'un cadre logique. Il existe deux approches principales à la logique temporelle : la logique modale et la logique à base de prédicats. La première repose sur l'utilisation de modificateurs désignant la probabilité d'occurrence d'un événement (e.g., « si *P* est vérifiée il est possible que *Q* soit vraie à un certain moment »). L'intermittence de service, c'est-à-dire l'hypothèse qu'un service interrompu redeviendra disponible s'exprimerait de la manière suivante :

Soit *d*, la proposition « le prestataire de service P est disponible », et *i*, « le prestataire de service P est indisponible ». L'axiome *F* (comme Futur) signifie : « à un certain moment il sera vrai que… ». L'intermittence se traduirait donc par :

$$i \rightarrow F_d \text{ et } d \rightarrow F_i$$

La seconde approche utilise le calcul des prédicats basée sur une logique de premier ordre. *L'Event Calculus* [Shanahan1999] est un exemple de calcul des prédicats de premier ordre. Un des prédicats, *Happens(a,t)*, signifie qu'une action *a* se produit à l'instant *t*. La proposition ci-dessus serait donc exprimée ainsi :

$$\exists t_1, t_2 \forall t [t_1 < t < t_2 \Rightarrow Happens(i, t_1) \wedge \neg Happens(d, t) \wedge Happens(d, t_2)]$$

Une fois les SLO exprimés dans des règles décrites par un tel formalisme, l'utilisation d'un moteur de règles logiques permettrait d'assurer la vérification des SLOs et de détecter les infractions à l'exécution. Un des avantages de cette approche logique est son extensibilité. A partir du formalisme, il est possible de définir différentes règles pour des SLOs additionnels, sans avoir à changer le moteur de règles. Alors qu'avec notre prototype il est nécessaire de modifier le code du SLM pour prendre en compte de nouveaux SLOs « temporels ».

[17] Real-Time Specification for Java (JSR-1), https://rtsj.dev.java.net/

IX.1.1.4 Approche prédictive

Dans notre proposition, l'offre est enregistrée par le fournisseur auprès du DSLAAdmin et n'est pas retirée lorsque le fournisseur est arrêté, contrairement à la publication/retrait de service. Il est alors possible qu'un accord soit conclu avec un fournisseur indisponible.

Cette approche innovante de la négociation, et donc de la sélection de service, consisterait à considérer les fournisseurs de service qui ne sont pas disponibles sur la plate-forme au moment de la sélection, mais qui, d'après leurs SLOs sont susceptibles de revenir dans un délai plus ou moins court. Il faudrait alors mettre en place un classement (*ranking*) basé sur la disponibilité. Ainsi, un consommateur peut choisir d'attendre le retour d'un fournisseur présentant de meilleures caractéristiques plutôt qu'en utiliser un déjà présent.

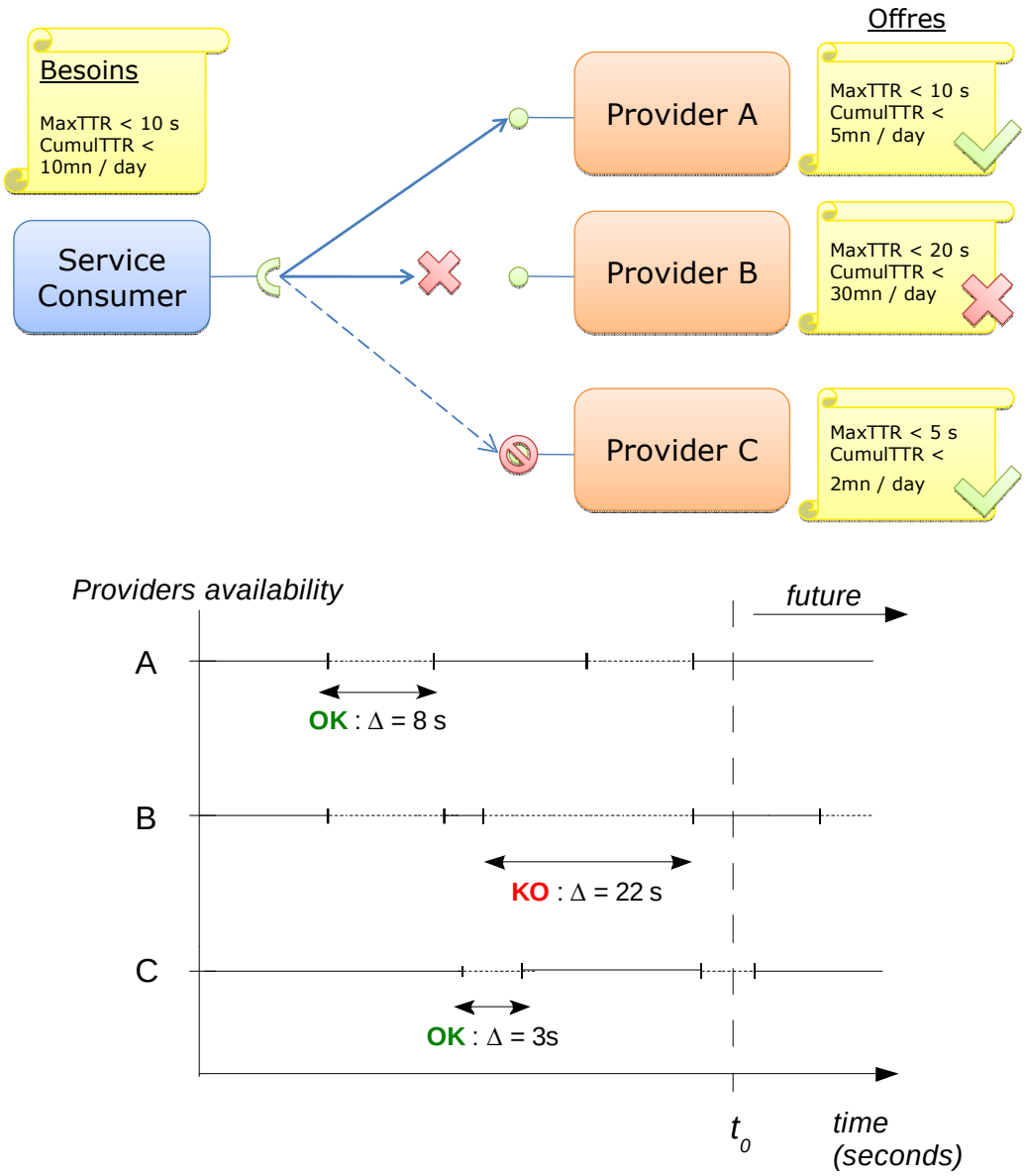

Figure 59. Sélection de service selon l'historique des interruptions et la disponibilité garantie.

L'exemple décrit Figure 59 met en scène un consommateur, trois fournisseurs (A, B, et C), et l'historique de leurs interruptions. Ici seules deux critères sont considérés : *MaxTTR* (la durée maximum d'une interruption) et *CumulTTR* (la durée maximum des temps d'interruptions cumulés). Le consommateur ne peut pas tolérer une interruption plus longue que 10 secondes et plus de 10 minutes d'interruption par jour. Les trois fournisseurs proposent différentes garanties. Seuls les prestataires A et C correspondent aux besoins du consommateur puisque B ne peut assurer une disponibilité suffisante comme montré sur le chronogramme. A t_0, au moment où le consommateur s'apprête à utiliser le service, seuls A et B sont disponibles. Bien qu'il puisse se lier au prestataire A, le consommateur pourrait également attendre le fournisseur C compte tenu de ses garanties légèrement meilleures.

Il existe également des approches probabilistes de sélection de service pour les réseaux ad-hoc mobiles et déconnectés, qui prennent en compte la probabilité d'interruption des services pour choisir les fournisseurs auxquels se lier [LeSommer2007]. Cette approche pourrait être complétée par l'approche prédictive décrite ci-dessus qui considère la probabilité de retour d'interruption. La sélection de service prendrait alors en considération, à la fois la probabilité d'interruption d'un fournisseur inférée d'après l'historique de ses interruptions passées et les garanties qu'il annonce, et la probabilité de retour d'un fournisseur calculée à partir des mêmes informations.

IX.1.1.5 DSLA pour le modèle publication - souscription

Dans cette thèse nous avons traité le cas des services communicant via le modèle classique « requête-réponse » et via le patron producteur-consommateur utile pour les applications utilisant des capteurs.

Le connecteur de type publication-souscription présente un couplage encore plus faible puisque les consommateurs ou puits d'événements (les composants abonnés) écoutent un topic géré par un médiateur sur lequel d'autres composants publient des événements. On peut donc imaginer que le lien « contrat de service » englobe le topic (canal de communication) et le type des événements. Dans ce mode de communication asynchrone, la présence des parties n'est pas supposée, alors qu'un consommateur a parfois besoin de recevoir des événements à intervalles réguliers (dans le cas d'un hopital recevant des rapports de patients par exemple).

Il est donc envisageable d'appliquer notre politique DSLA à ce type de services, en agissant au niveau du médiateur (l'EventAdmin d'OSGi par exemple). Les informations nécessaires à l'établissent des SLO et de l'accord seraient issues de la configuration des composants souscripteurs et publieurs. Il faudrait alors étendre les handlers iPOJO de gestion des événements existants et en proposer une version DSLA.

IX.1.1.6 Accords bilatéraux et contrôle d'admission

A travers ce document, la bilatéralité des accords a été évoquée à plusieurs reprises. Cette idée d'accord bilatéral part du principe que le fournisseur doit pouvoir aussi choisir ses partenaires, c'est-à-dire les consommateurs du service qu'il fournit.

Toutefois, mettre en œuvre cette bilatéralité pose plusieurs verrous à lever. Tout d'abord, le fournisseur doit pouvoir exprimer ses besoins, et le client ses capacités, puis les enregistrer (auprès du DSLAAdmin par exemple).

Ensuite, il est nécessaire de pouvoir suivre et surveiller la disponibilité des cconsommateurs de service. Dans le cas des producteurs-consommateurs de données, le consommateur enregistre un service *WireAdmin Consumer* et un PID dans le registre OSGi, par conséquent la supervision de sa disponibilité est immédiate. Toutefois, l'approche classique des services (de type requête-réponse) pose problème puisque les consommateurs ne se signalent pas. Toutefois, ceci change peu à peu avec l'arrivée de mécanismes tels que les *ServiceHooks*[18] dans OSGi R4.2.

Les fournisseurs devraient aussi pouvoir filter les consommateurs qui utilisent leurs services afin de réaliser un contrôle d'admission. Dans un contexte d'intelligence ambiante où un service représente un dispositif physique, il peut s'avérer nécessaire de contrôler les utilisateurs du service, en particulier si l'objet ne peut servir qu'un utilisateur à la fois. Le contrôle d'admission peut se faire sur des critères propres au fournisseur (sa charge actuelle par exemple), mais aussi par rapport aux caractéristiques des consommateurs souhaitant utiliser ses services. Par exemple il peut choisir de ne servir que des consommateurs ayant un compte premium, n'effectuant pas plus de 10 appels par minute, ou encore, dans le cas qui nous intéresse, les consommateurs garantissant leur disponibilité et leurs temps d'interruption. Ces garanties peuvent en effet inciter le fournisseur à maintenir la session d'un consommateur temporairement déconnecté, ou simplement à conserver le contrat qui les lie, et ne pas servir d'autres clients si le nombre de consommateurs est limité.

Une solution envisagée consisterait à modéliser les liaisons de service, qui sont généralement implicites (dans OSGi il s'agit d'une référence vers un objet Java), en s'inspirant du patron proposé par la spécification WireAdmin d'OSGi qui réifie les connexions. Cependant ce point n'étant pas directement lié à cette thèse, il est abordé dans la section suivante relative aux possibles pistes de recherches ouvertes par ces travaux.

IX.1.2 Pistes connexes

Parmi les perspectives de recherche que nous avons dégagées suite aux travaux présentés dans ce document, les pistes suivantes ne s'inscrivent pas directement dans la continuité. En revanche il s'agit de questions de recherches intéssantes qui mériteraient d'être approfondies selon la direction que prendront mes futurs travaux.

IX.1.2.1 Réification des liaisons

A la fin de la section précédente, nous avons parlé de modélisation et de réification de la liaison consommateur-fournisseur. Cette approche inspirée par le WireAdmin consisterait

[18] Communiqué de Peter Kriens sur les Service Hooks sur le blog de l'OSGi Alliance (18 Février 2009), http://www.osgi.org/blog/2009/02/osgi-service-hooks.html

à externaliser la gestion de la liaison de service : le connecteur ne serait plus créé/détruit par le composant consommateur, mais par un gestionnaire externe. La liaison se comporterait comme un mandataire du service auquel elle est connectée. En dehors d'une possible perte de performances, expliciter les liaisons permettrait :

- Un découplage des composants et de leurs connecteurs. La liaison aurait un cycle de vie indépendant de celui du consommateur ou du fournisseur. Elle pourrait donc exister sans les composants qu'elle lie.
- Une symétrie dans les rôles de consommateurs et fournisseurs, chaque entité étant consciente des liens auxquels elle est liée, et donc de ses partenaires. Cette approche faciliterait alors la mise en œuvre du contrôle d'admission, puisque les liaisons (et possiblement les SLAs correspondants) pourraient être créées (suite à une négociation) en confrontant les besoins ET capacités de chaque partie, y compris ceux du fournisseur de service. Et une fois lié à un consommateur, un fournisseur pourrait choisir de le servir ou pas.
- D'effectuer certains traitements au niveau de la liaison. En explicitant la liaison il est aussi possible de lui assigner des tâches supplémentaires telles que le filtrage, le gel d'appels de méthodes si le service n'est pas présent, la supervision de critères de qualité de service tels que le temps de réponse en y ajoutant des sondes.
- D'attacher un accord de niveau de service à une liaison.

IX.1.2.2 Négociation passive

A l'instar des phases de découverte et de sélection, la négociation pourrait également être un processus passif. C'est-à-dire qu'un consommateur n'aurait pas besoin de découvrir un fournisseur de service et de démarrer explicitement le processus de négociation. En enregistrant leurs besoins et leurs capacités, ainsi que leurs stratégies de négociation auprès du composant négociateur, consommateurs et fournisseurs pourraient laisser ce composant se charger de manière autonome de la négociation, et ainsi de leur liaison.

Le principe est le même que celui de la découverte passive. Une piste consisterait à étendre l'annuaire de service avec des fonctionnalités de négociateur. La création de liaison et l'établissement d'accords de niveau de service seraient ainsi gérés par la même entité, un annuaire de service étendu à la gestion des accords de niveau de service.

IX.1.2.3 Accords multi-parties dans une composition

Pendant cette thèse, s'est également posée la question des compositions de services et l'impact qu'elles peuvent avoir sur les accords de niveau de service. Le choix a été fait d'avoir un SLA par liaison de service, et de ne considérer que la qualité de service des parties deux à deux. C'est-à-dire que si un composant A requiert un service B qui utilise un service C, l'accord entre A et B ne fait pas référence aux capacités de C. C'est au composant fournissant B d'exprimer ses contraintes vis-à-vis de C et de prévoir des

politiques de recours permettant de compenser les impacts possible sur les liaisons établies avec ses consommateurs directs.

Nous avons néanmoins envisagé l'éventualité d'accords multi-parties établis entre un consommateur et plusieurs fournisseurs. Il existe en effet des cas où une application repose sur la coopération et la coordination de plusieurs services pour atteindre son but. Par exemple le système de réservation d'une agence de voyage doit coopérer avec les services de plusieurs compagnies de transport (ferroviaires ou aériennes) et de logement, ainsi que d'autres services (banque, etc).

IX.1.2.4 Déploiement d'applications mission-critique orientées service

L'approche décrite dans cette thèse peut servir d'inspiration au déploiement, et plus particulièrement aux mises à jour de systèmes critiques. En effet, les applications mission-critiques ont des contraintes de disponibilité importantes. Les mises à jour peuvent être de natures diverses (correctif, critique, non-critique, évolutive) mais résultent en une interruption du composant mis à jour.

Dans certains cas, il est donc préférable de retarder les opérations de mises à jour afin de respecter les contraintes de disponibilité qui peuvent être exprimées dans des accords de niveau de service. Par exemple, dans le cas d'applications hébergées, il y a des périodes où l'interruption n'est pas tolérable. Les ventes privées ou « flash » à durée limitée attirent un grand nombre d'utilisateurs sur une courte période, et les mises à jour ne peuvent évidemment pas être effectuées sur cette période. Il en est de même pour les sites de résultats (du baccalauréat, de concours, ou résultats sportifs) ou gouvernementaux (déclaration d'impôts en ligne, élections) qui peuvent identifier aisément les périodes de grande affluence et exprimer leurs contraintes sur les interruptions en conséquence.

Il s'agirait donc d'utiliser les informations fournies par les SLOs des divers composants pour piloter les opérations de déploiement et savoir si un service peut être arrêté ou pas.

IX.1.2.5 Cycle de vie des composants iPOJO

Au contact du modèle à composants iPOJO, nous avons remarqué qu'il n'était pas toujours idéal de lier le cycle de vie à la disponibilité des dépendances de services. D'une part parce que, comme nous avons pu le voir dans la validation sur AspireRFID, il est plus immédiat de lier l'activation ou la désactivation d'une application à une autre donnée telle que la température, plutôt qu'à des données temporelles. D'ailleurs le modèle iPOJO permet déjà de contrôler le cycle de vie en le liant à des conditions grâce au handler *LifeCycleController*.

D'autre part, ce choix d'arrêter l'application lorsqu'une dépendance n'est plus résolue n'est pas toujours justifié. Nous avons vu qu'il existe des cas où une application peut utiliser des services de manière sporadique, voire une seule fois. C'est pourquoi, il serait plus efficace d'agir sur le cycle de vie du composant uniquement au moment de l'utilisation effective d'une dépendance, c'est-à-dire à l'instant où le fil d'exécution accède au champ dans lequel le service est injecté.

IX.2 Conclusion

Cette thèse fait suite au travail effectué pendant mon Master Recherche [Touseau2006]. Durant ce master, la problématique des accords de niveau de service appliqués aux applications M2M sur des plates-formes dynamique à services avait été mise en évidence.

IX.2.1 Synthèse

L'évolution des technologies de l'information conduit peu à peu à la « dématérialisation » de l'informatique avec, d'une part les internets d'objets qui contribue à mettre en œuvre une intelligence ambiante, et d'autre part l'Internet des services qui propose un nouveau mode de consommation « à la demande » des applications, mais aussi des infrastructures logicielles et matérielles. De plus, sur cette toile de fond, les préoccupations écologiques poussent à l'économie d'énergie. et à la création de nouvelles applications utilisant l'informatique pour gérer plus intelligemment la consommation énergétique.

Les applications issues de ces domaines émergents ont la particularité d'avoir une disponibilité fluctuante dépendant de divers facteurs (mise en veille, perte de connectivité avec un équipement sans fil, mises à jour, etc). Par conséquent leur fonctionnement intermittent implique des interruptions répétées des services qu'elles fournissent.

L'approche décrite dans cette thèse se place dans le contexte de la programmation orientée service et des architectures dynamiques. La programmation à base de composants appliquée à la construction d'applications orientées service permet une gestion intrinsèque de la disponibilité dynamique des services. La logique d'adaptation des composants est intégrée au conteneur des composants et n'est plus à la charge des développeurs. A partir du constat sur les contraintes de disponibilité des nouvelles applications, nous avons jugé nécessaire de revoir les politiques d'adaptation mis en œuvre par ces modèles à composants orientés service.

En effet, suite à une interruption de service, les politiques usuelles suivent soit une approche statique et ne tolèrent pas l'interruption, soit une approche dynamique et reconfigurent les liaisons entre l'application et le service. Cependant, malgré le potentiel de l'approche dynamique, reconfigurer dynamiquement l'architecture des applications au moindre changement d'état de leurs dépendances de service n'est pas toujours une bonne décision. Lorsque des services sont interrompus temporairement du fait de leur fonctionnement intermittent, adopter une politique de liaison dynamique n'est pas toujours une solution optimum et peut même s'avérer contre-productif.

La proposition faite ici consiste à contractualiser les liaisons de service par des accords de niveau de service (SLA) portant sur la disponibilité et les interruptions de service. La politique de liaison baptisée DSLA s'appuie sur les SLA négociés entre consommateurs et fournisseurs de services pour décider de la création, de la destruction ou bien de la

conservation d'une liaison. L'utilisation d'accords de niveau de service permet à la fois de qualifier la disponibilité des composants, mais aussi de garantir cette disponibilité et de prévoir des recours au cas où l'accord serait enfreint, ce qui peut s'avérer nécessaire dans le cas d'applications critiques.

Cette proposition a été mise en œuvre sur la plate-forme OSGi qui permet de développer des applications dynamiques à base de services. Nous avons étendu le modèle à composants iPOJO afin qu'il supporte notre politique de liaison. L'approche a été validée sur les composants du serveur d'application JOnAS, dans une application construite sur l'intergiciel de médiation de données issues de capteurs AspireRFID, et sur les services techniques de gestion de la persistance et de la distribution utilisés par la passerelle résidentielle H-Omega.

IX.2.2 Bilan

Au cours de cette thèse nous avons pu constater que les travaux menés portent sur une problématique d'actualité. Le principe de gestion des dépendances est relativement proche des solutions basées sur des liaisons temporisées qui ont vu le jour pendant cette thèse. A la différence que ces approches se limitent à une temporisation fixe, dépendant uniquement des besoins du consommateur de service, tandis que la notre permet de piloter la gestion des liaisons de manière plus fine et contractualise les reconfigurations à travers des accords de niveau de service validés à la fois par le consommateur, et le fournisseur. La gestion des interruptions est plus fine car les paramètres caractéristiques des services intermittents ne se limitent pas au temps d'interruption et peuvent même prendre en compte des conditions temporelles de validité, sous forme de fenêtres temporelles de fonctionnement par exemple.

De même, au début de la thèse, la problématique des services intermittents nous était apparue essentiellement à travers le contexte des applications ubiquitaires et pervasives, ainsi qu'au niveau des serveurs d'applications et autres logiciels de taille importante construits sur l'approche orientée service. Ce n'est que très récemment que de nouveaux contextes applicatifs tels que le *cloud computing* ou le *greenIT* nous sont apparus. Ces domaines émergents ont néanmoins conforté la position de nos travaux. Ils confirment le caractère intermittent des nouvelles applications où l'interruption de service représente un état normal du fonctionnement des systèmes informatiques, par exemple dans le cas d'une mise en hibernation d'une partie des ressources pour diminuer la dépense énergétique, tout en adaptant l'offre à la demande.

L'approche proposée par cette thèse s'applique essentiellement aux services qualifiés d'intermittents. Les politiques d'adaptation dynamique conviennent à la plupart des cas d'utilisation.

En effet, une première difficulté consiste à évaluer les besoins et les capacités des consommateurs et fournisseurs de service du point de vue de leurs interruptions. Ces informations ne sont pas toujours renseignées lors de l'écriture des spécifications techniques d'un composant, et peuvent de plus dépendre du contexte dans lequel le composant est utilisé. Nous considérons que cette tâche revient à l'assembleur d'applications, ou bien qu'elle soit automatisée par analyse du comportement des composants.

Enfin, si notre politique DSLA est plus avantageuse dans certains scénarios, elle n'est pas efficace dans les cas défavorables suivants :

- Lorsque le fournisseur interrompu ne revient pas, l'attente s'avère vaine.
- De même, si le fournisseur est interrompu le consommateur se trouve dans un état d'attente avant de se lier à un autre fournisseur. Or si ce second fournisseur devient lui aussi indisponible et que le premier ne revient toujours pas, alors la substitution aurait été une option plus intéressante.

Dans ces situations l'indisponibilité n'est effectivement constatée qu'après un délai, lorsque l'interruption dépasse le seuil de tolérance, et non immédiatement comme dans l'approche dynamique. La reconfiguration est donc retardée par rapport aux politiques adaptatives qui auraient immédiatement détruit la liaison et possiblement créé une nouvelle avec un autre fournisseur.

Toutefois, l'utilisation d'accords de niveau de service et de politiques de recours permet de parer à ces cas défavorables, d'une part en sanctionnant une indisponibilité dépassant les limites fixées dans l'accord et donc en obligeant les partie engagées, et d'autre part par la possibilité de fournir une compensation à la partie lésée.

Chapitre X

Bibliographie

[Adam1986] Daniel A. Adam, Daniel L. Sumner, "Service Level Management: A Strategy for Comprehensive Information Systems Management", in proceedings of 12th International Computer Measurement Group Conference, pp.550-555, Las Vegas, NV, USA, December 8-12, 1986

[Aiello2005] Aiello, M., Frankova, G., and Malfatti, D., "What's in an Agreement? An Analysis and an Extension of WS-Agreement", in Proceedings of 3rd International Conference On Service Oriented Computing (ICSOC'05), Amsterdam, The Netherlands, December 12-15, 2005

[Amazon2007] Amazon S3 : Simple Storage Service, 2007 http://aws.amazon.com/s3/ et http://aws.amazon.com/s3-sla/

[Ananda1992] Ananda, A. L., Tay, B. H., and Koh, E. K. 1992. "A survey of asynchronous remote procedure calls", ACM SIGOPS Operating Systems Review Vol.26, Issue 2, pp.92-109, April 1992.
DOI= http://doi.acm.org/10.1145/142111.142121

[Andrieux et al 2004] A. Andrieux, C. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu., "Web Services Agreement Specification (WS-Agreement)", Version 1.1, Draft 20, June 6th 2004

[Arenas et al 2008] Arenas, A., Wilson, M., Crompton, S., Cojocarasu, D., Mahler, T., and Schubert, L., "Bridging the Gap between Legal and Technical Contracts", IEEE Internet Computing, Volume 12, Issue 2, pp. 13-19, Mars 2008.
DOI= http://dx.doi.org/10.1109/MIC.2008.28, 2008

[Asawa1998] Manjari Asawa, "Measuring and Analyzing Service Levels: A Scalable Passive Approach", in Proceedings of 6th IEEE/IFIP International Workshop on Quality of Service, Napa, CA, USA, pages 3–12. IEEE/IFIP, May 18–20 1998

[Avizienis2001] Avizienis, A., Laprie, J.C., Randell, B., "Fundamental concepts of dependability", Research Report No 1145, LAAS-CNRS, April 2001

[Beisiegel et al 2007] M. Beisiegel, et al., "SCA Service Component Architecture: Assembly Model Specification", Open Service Oriented Architecture, 2007; http://www.osoa.org/display/Main/Service+Component+Architecture+Home.

[Bertolino et al 2009] Bertolino, A., Angelis, G. D., Frantzen, L., and Polini, A., "The PLASTIC Framework and Tools for Testing Service-Oriented Applications", in Software Engineering: International Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures, pages 106–139, 2009

[Beugnard1999] Beugnard, A., Jézéquel, J., Plouzeau, N., and Watkins, D., "Making Components Contract Aware", Computer 32, 7 (July 1999), pp.38-45, 1999.
DOI= http://dx.doi.org/10.1109/2.774917

[Bianco2008] Bianco, P.; Lewis, G. A. & Merson, P., "Service Level Agreements in Service-Oriented Architecture Environments", Techreport, Software Engineering Institute of The Carnegie Mellon University, 2008

[Bieber2002] Bieber, G. and Carpenter, J. , "Introduction to Service-Oriented Programming", Rev 2.1, http://www.openwings.org, 2002

[BMC2010] BMC Performance Manager, http://www.bmc.com/products/product-listing/53174792-132703-1311.html, 2010

[Boschi et al 2006] Elisa Boschi, Spyros Denazis, Tanja Zseby, "A measurement framework for inter-domain SLA validation", Computer Communications, pp.703-716, Volume 29, Issue 6, 31 March 2006

[Bosschaert2009] David Bosschaert and Tim Diekmann , "Distributed OSGi", presentation at EclipseCon 2009, Supporting Technology - OSGi DevCon, Santa Clara, California, USA, 23-26 March 2009

[Bottaro2006] 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, April 2006

[Bottaro2007] André Bottaro, Richard S. Hall, "Dynamic Contextual Service Ranking", in proceedings of 6th International Symposium on Software Composition (SC 2007), Braga, Portugal, March 2007

[Bourcier2007] J. Bourcier, C. Escoffier, and P. Lalanda, "Implementing Home-Control Applications on Service Platform", in proceedings of 4th Consumer Communications and Networking Conference (CCNC'07), Las Vegas, Nevada, USA, 11-13 January 2007.

[Bruneton et al 2004] E. Bruneton, T. Coupaye, M. Leclercq, V. Quema, J.-B. Stefani. "An open component model and its support in Java", 7th International Symposium on Component-Based Software Engineering (CBSE), LNCS 3054, pp 7-22, May 2004.

[Buco et al 2004] Buco, M. J., Chang, R. N., Luan, L. Z., Ward, C., Wolf, J. L., and Yu, P. S., "Utility computing SLA management based upon business objectives", IBM Systems Journal, Volume 43, Issue 1, pp.159-178, January 2004

[Campbell1994] Campbell, A., Coulson, G., and Hutchison, D., "A quality of service architecture", SIGCOMM Computer Communication Review, Volume 24, Issue 2, pp.6-27, April 1994.
DOI= http://doi.acm.org/10.1145/185595.185648, 1994

[CCA2004] Common Component Architecture (CCA) Forum, http://www.cca-forum.org/, 2004

[Cervantes2002] H. Cervantes and R. S. Hall, "Beanome : A Component Model for the OSGi Framework", proceedings of the international workshop on Software Infrastructures for Component-Based Applications on Consumer Devices, Lausanne, Switzerland, September 2002

[Cervantes2004a] H. Cervantes and R. S. Hall, "Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model", International Conference on Software Engineering (ICSE), Edinburgh, Scotland, May 2004

[Cervantes2004b] Humberto Cervantes , "Vers un modèle à composants orienté services pour supporter la disponibilité dynamique", thèse de doctorat de l'Université Joseph Fourier, 3 Mars 2004

[Channabasavaiah2003] Kishore Channabasavaiah and Kerrie Holley and Edward Tuggle, Jr., "Migrating to a service-oriented architecture", IBM DeveloperWorks White Paper, December 2003

[Chappell2004] David A Chappell , "Enterprise Service Bus", published by O'Reilly Media, 288 pages, ISBN: 978-0-596-00675-4 | ISBN 10: 0-596-00675-6, June 2004

[Chappell2007] David Chappell, "Introducing SCA", http://www.davidchappell.com/articles/Introducing_SCA.pdf, 2007

[Cloud2010] Cloud computing use case discussion group, "Cloud Computing Use Case White Paper V3", Open cloud manifesto, http://opencloudmanifesto.org/resources.htm, 2010

[Collet2005] Collet P. et Rousseau R., "ConFract : un système pour contractualiser des composants logiciels hiérarchiques", paru dans L'Objet, ISSN 1262-1137. Langages et Modèles à Objets (LMO), Conférence N°11, Berne, SUISSE, vol. 11, no 1-2, pp.223-238, 2005. DOI:10.3166/objet.11.1-2.223-238

[Colyer et al 2007] A. Colyer, et al., "Spring Dynamic Modules for OSGi 1.0.1", http://static.springframework.org/osgi/docs/1.0.1/reference/html/ , 2007.

[Dan et al 2004] Asit Dan, Doug Davis, Robert Kearney, Alexander Keller, Richard P. King, Dietmar Kuebler, Heiko Ludwig, Mike Polan, Mike Spreitzer and Alaa Youssef, "Web services on demand: WSLA-driven automated management", IBM Systems Journal, Volume 43, 2004

[Desertot2006] Mikael Desertot, Didier Donsez and Philippe Lalanda, "A Dynamic Service-Oriented Implementation for Java EE Servers", in proceedings of the 3th IEEE International Conference on Service Computing (SCC'06), Chicago, USA, 18-22 September 2006, Publisher: IEEE Computer Society, ISBN: 0-7695-2670-5, DOI: 10.1109/SCC.2006.4

[Desertot2007] Mikaël Desertot, "Une architecture adaptable et dynamique pour les serveurs d'applications", thèse de doctorat de l'Université Joseph Fourier, Grenoble, 27 Mai 2007

[DiModica2007] G. Di Modica, V. Regalbuto, O. Tomarchio, and L. Vita, "Enabling re-negotiations of SLA by extending the WSAgreement Specification", in proceedings of 4th IEEE International Conference on Services Computing (SCC'07), pages 248–251, Salt Lake City, Utah, USA, July 9-13, 2007

[Dixit2001] S Dixit, Y Guo, Z Antoniou, “Resource management and quality of service in third generation wireless networks”, IEEE Communications Magazine, Vol.39, February 2001

[Dodani2004] Mahesh H. Dodani, “From objects to services : A journey in search of component reuse nirvana”, Journal of Object Technology, 3(8) :49–54, 2004.

[Donsez2006] Didier Donsez, “Objets, composants et services : intégration de propriétés non fonctionnelles”, Habilitation à Diriger des Recherches en Informatique de l’Université Joseph Fourier (Grenoble 1), 11 décembre 2006

[Donsez2009a] Didier Donsez, Kiev Gama and Walter Rudametkin, “Developing Adaptable Components using Dynamic Languages”, in proceedings of 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEEA), Patras, Greece, 27 Août 2009

[Donsez2009b] Didier Donsez, Lionel Touseau and Kiev Gama, “Experimenting with the OSGi platform in the Aspire RFID middleware”, OSGiDevCon 2009, ETHZ, Zurich, June 22, 2009, http://www.osgi.org/DevConEurope2009/Schedule

[Duclos2002] Duclos, F., Estublier, J., and Morat, P., “Describing and using non functional aspects in component based applications”, in Proceedings of the 1st international Conference on Aspect-Oriented Software Development (AOSD’02), Enschede, The Netherlands, April 22 - 26, 2002, DOI= http://doi.acm.org/10.1145/508386.508394

[EJB2010] Oracle - Sun Microsystems, “Enterprise Java Beans”, http://java.sun.com/products/ejb/, 2010

[Escoffier2008] Clément Escoffier, “iPOJO : Un modèle à composant à service flexible pour les systèmes dynamiques”, thèse de doctorat de l’Université Joseph Fourier, 3 Décembre 2008

[Escoffier2009] Clément Escoffier, “iPOJO Temporal Dependency documentation”, http://felix.apache.org/site/temporal-service-dependency.html, 2009

[Eskelin1999] Philip Eskelin, “Component Interaction Patterns”, in proceedings of Pattern Languages of Programs (PLoP’99), 1999

[Eugster2003] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A., “The many faces of publish/subscribe”, ACM Computing Surveys, Vol.35, issue 2, pp.114-131, June 2003. DOI= http://doi.acm.org/10.1145/857076.857078

[Fabry1976] R.S. Fabry, "How to design a system in which modules can be changed on the fly", in the proceedings of the International Conference on Software Engineering (ICSE), pp 470-476, 1976

[FAROS2006] Projet RNTL FAROS, "Livrable F-1.1 - Etat de l'art sur la contractualisation et la composition", Aout 2006

[Fielding2000] Fielding, Roy Thomas, "Architectural Styles and the Design of Network-based Software Architectures", doctoral dissertation, University of California, Irvine, 2000 http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[Fouquet2010] Guy Fouquet (Alcatel-Lucent Business Development Director), "Solutions d'alternative energies et de sustainable power", pour les matinées thématiques du LIG sur le GreenIT, 26 Janvier 2010.

[Frénot2008] Frénot Stéphane, Royon Yvan, Parrend Pierre, Beras Denis, "Monitoring Scheduling for Home Gateways", in IEEE/IFP network operations and management symposium, pp.411-416, 7 Avril 2008, DOI: 10.1109/NOMS.2008.4575162

[Gama2008] Kiev Gama and Didier Donsez , "A Practical Approach for Finding Stale References in a Dynamic Service Platform", in proceedings of 11th International Symposium on Component Based Software Engineering (CBSE-2008), Karlsruhe, Germany, 14 Octobre 2008

[Gamma et al 1995] E. Gamma, et al., "Design patterns: Elements of reusable object-oriented software", 1995, ISBN 0-201-63361-2

[Garlan1993] D. Garlan and M. Shaw, "An introduction to software architecture", published in V. Ambriola and G. Tortora (ed.), Advances in Software Engineering and Knowledge Engineering, Series on Software Engineering and Knowledge Engineering, Vol 2, World Scientific Publishing Company, Singapore, pp. 1-39, 1993

[Gimpel2003] Gimpel H., Ludwig H., Dan A., Kearney B., "PANDA: Specifying policies for automated negotiations of service contracts", in proceedings of 1st international conference on service-oriented computing (ICSOC'03), Trento, Italy, 15-18 December 2003

[Girolami et al 2008] Michele Girolami, Stefano Lenzi, Francesco Furfari, Stefano Chessa, "SAIL: A Sensor Abstraction and Integration Layer for Context Awareness", Software Engineering and Advanced Applications, 34th Euromicro Conference (SEAA'08), pp. 374-381, Parma, Italy, 3-5 September 2008

[Glassfish2009] Sun Glassfish Enterprise Server, http://www.sun.com/software/products/appsrvr/index.jsp, 2009

[Goland et al 1999] Y. Goland, et al., "Simple Service Discovery Protocol v1.0", IETF, 1999.

[Gray1991] Gray, J., Siewiorek, D.P., "High-availability computer systems", Computer Voume 24, Issue 9, pp.39-48, September 1991, DOI= http://dx.doi.org/10.1109/2.84898

[Gruber et al 2005] O. Gruber, et al., "The Eclipse 3.0 platform: adopting OSGi technology," IBM System Journal, vol. 44, no. 2, pp. 289-299, 2005

[Guice2009] Google-Guice project, http://code.google.com/p/google-guice/, 2009

[Gürgen et al 2008] Gurgen, L., Roncancio, C., Labbé, C., Bottaro, A., and Olive, V., "SStreaMWare: a service oriented middleware for heterogeneous sensor data management", In Proceedings of the 5th international Conference on Pervasive Services (ICPS'08), Sorrento, Italy, July 06 - 10, 2008 DOI= http://doi.acm.org/10.1145/1387269.1387290

[Harrop2009] Peter Harrop and Raghu Das, "Active RFID and Sensor Networks 2009-2019", rapport d'IDTechEx, http://www.idtechex.com/research/reports/active_rfid_and_sensor_networks_2009_2019_000166.asp, 2009

[Hashimi2003] Sayed Hashimi, "Service-oriented architecture explained", O'reilly on dot net, August 2003, http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html

[Herssens2008] Herssens, C., Faulkner, S., and Jureta, I. J., "Context-Driven Autonomic Adaptation of SLA", in Proceedings of the 6th international Conference on Service-Oriented Computing (ICSOC'08), Sydney, Australia, December 01 - 05, 2008, DOI= http://dx.doi.org/10.1007/978-3-540-89652-4_28

[Hovestadt et al 2006] Matthias Hovestadt, Odej Kao, Kerstin Voß, "Multiagent Policy Architecture for Virtual Business Organizations", in proceedings of 2006 IEEE International Conference on Services Computing (SCC'06), Chicago, Illinois, USA, 18-22 September 2006

[Huang1993] Huang, Y., Kintala, C., "Software implemented fault tolerance: Technologies and experience", Fault-Tolerant Computing, 1993. FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing, June 1993

[Huhns2005] Huhns, M.N. and Singh, M.P., "Service-oriented computing: key concepts and principles", in IEEE Internet Computing, Volume 9, Issue 1, pp.75-81, Jan-Feb 2005, ISSN: 1089-7801, DOI: 10.1109/MIC.2005.21

[ITIL2008] Information Technology Infrastructure Library (ITIL), "SLA Toolkit", http://www.service-level-agreement.net/, 2008

[ITU2005] International Telecommunication Union, "The Internet of Things", ITU Internet Reports 2005

[Jammes et al 2005] François Jammes, Antoine Mensch and Harm Smit, "Serviceoriented device communications using the devices profile for web services", MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, 2005

[Jean et al 2009] Sébastien Jean, Kiev Gama, Didier Donsez and André Lagrèze, "Towards a Monitoring System for High Altitude Objects", in Proceedings of the 6th international Conference on Mobile Technology, Applications, and Systems (ACM Mobility Conference 2009), Nice, France, September 2-4, 2009.

[JOnAS2007] Objectweb, "Jonas: Java Open Application Server", http://wiki.jonas.objectweb.org/xwiki/bin/view/Main/WebHome, 2007

[Jones2005] Steve Jones, "Towards an acceptable definition of service", in IEEE Software, Volume 22, Issue 3, pp.87-93, May-June 2005

[Jordan2007] D. Jordan and J. Evdemon, "Web Services Business Process Execution Language Version 2.0", OASIS, 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[Jurca2005] Radu Jurca , Boi Faltings, "Reputation-Based Service Level Agreements for Web Services", in Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC'05), Amsterdam, The Netherlands, December 12-15, 2005

[Karten2001] Naomi Karten, "How to establish Service Level Agreement", Handook, http://www.nkarten.com/, 2001

[Kefalakis et al. 2008] Kefalakis N, Leontiadis N, Soldatos J, Gama K, Donsez D., "Supply chain management and NFC picking demonstrations using the AspireRd middleware platform", in Companion proceedings of ACM/IFIP/USENIX 9th International Middleware Conference (Middleware'08), Leuven, Belgium, December 1-5, 2008

[Keller2002] A. Keller and H. Ludwig, "Defining and Monitoring Service Level Agreements for dynamic e-Business", in Proceedings of the 16th USENIX Conference on System Administration, Philadelphia, PA, November 03 - 08, 2002

[Kephart2003] J. Kephart and D. Chess, "The Vision of Autonomic Computing" IEEE Computer, vol. 36, pp. 41-50, 2003

[Ketfi2002] M. Ketfi, N. Belkhatir, P.Y. Cunin, "Dynamic updating of component-based applications", International Conference on Software Engineering Research and Practice (SERP'02) PDPTA, Las Vegas, Nevada, USA, June 2002

[Keynote2010] Keynote, "The Mobile and Internet Performance Authority", Keynote Systems, Inc., http://www.keynote.com, 2010

[Kilov1990] Kilov, Haim, "From semantic to object-oriented data modelling", First International Conference on System Integration, pp.385 - 393, 1990.

[Krakowiak2009] Sacha Krakowiak, "Middleware Principles and Basic Patterns", Chapter 2 of e-book on Middleware Architecture, © 2003-2009 S. Krakowiak, Creative Commons license, available on http://sardes.inrialpes.fr/~krakowia/MW-Book/Chapters/Basic/basic.html

[Lamanna2003] D. Davide Lamanna, James Skene, Wolfgang Emmerich, SLAng, "A Language for Defining Service Level Agreements". In proceedings of 9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003), San Juan, Puerto Rico, 28-30 May 2003. IEEE Computer Society 2003, ISBN 0-7695-1910-5

[Leblanc et al 2002] Leblanc, S., Merle, P. & Geib, J.-M., "Vers des composants de courtage avec TORBA", L'OBJET, Volume 8, pp. 185-201, 2002

[Lee et al 2003] J. Lee, et al., "Enterprise integration with ERP and EAI", Communication of the ACM, vol. 46, no. 2, pp. 54-60, 2003

[LeSommer2007] Nicolas Le Sommer, "A Framework for Service Provision in Intermittently Connected Mobile Ad hoc Networks", in 8th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM 2007), Helsinki, Finland, June 2007.

[Luckham2002] Luckham, D. C., "The Power of Events: an Introduction to Complex Event Processing in Distributed Enterprise Systems", Addison-Wesley Longman Publishing Co., Inc, ISBN:0201727897, 2002

[Ludwig et al 2003] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification" , http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf , January 2003

[Ludwig2004] Ludwig, H., Dan, A., and Kearney, R., "Cremona: an architecture and library for creation and monitoring of WS-agreements", in Proceedings of the 2nd international Conference on Service Oriented Computing (ICSOC'04), New York, NY, USA, November 15 - 19, 2004.
ACM, New York, NY, pp.65-74.
DOI= http://doi.acm.org/10.1145/1035167.1035178

[MacKenzie et al 2006] Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz, "Reference Model for Service Oriented Architecture 1.0", Technical Report wd-soa-rm-cd1, OASIS, February 2006.

[Mahler2006] T. Mahler and F. Vraalsen, "Legal Risk Management for an E-Learning Web Services Collaboration," Legal, Privacy and Security Issues in Information Technology, S.M. Kierkegaard, ed., Unipub/Complex, pp.503–523, 2006

[Malloy2006] Brian A. Malloy, Nicholas A. Kraft, Jason O. Hallstrom and Jeffrey M. Voas, "Improving the Predictable Assembly of Service-Oriented Architectures", IEEE Software, Volume 23, IEEE Computer Society Press, 2006

[Mandarax2007] Mandarax Project, http://mandarax.sourceforge.net/, 2007

[Marin2005] C. Marin, M. Desertot, “SensorBean: A Component Platform for Sensor-Based Services”, proceedings of the International Worshop of Middleware for Pervasive and Ad-Hoc Compouting (MPAC), Grenoble, France, 28-29 November 2005

[Marples2004] D. Marples, S. Moyer, “Home Networking and Appliances”, in Diane Cook, Sajal Das, Smart Environments: Technologies, Protocols and Applications, Wiley, 2004.

[Medvidovic2000] N. Medvidovic, R.N. Taylor, “A Classification and Comparison Framework for Software Architecture Description Languages”, IEEE Transactions on Software Engineering, Vol. 26, No. 1, (pp. 70-96), January 2000, DOI= http://dx.doi.org/10.1109/32.825767

[Mehta2000] Mehta, N. R., Medvidovic, N., and Phadke, S., “Towards a taxonomy of software connectors”, in Proceedings of the 22nd international Conference on Software Engineering (ICSE '00), Limerick, Ireland, June 04 - 11, 2000.
ACM, New York, NY, pp.178-187.
DOI= http://doi.acm.org/10.1145/337180.337201

[Meyer1992] Meyer, B., “Applying "Design by Contract" ”, Computer Volume 25, Issue 10, pp. 40-51, October 1992.
DOI= http://dx.doi.org/10.1109/2.161279

[Michahelles et al 2007] Michahelles, F., Thiesse, F., Schmidt, A. and Williams, J.R., “Pervasive RFID and Near Field Communication Technology”, Pervasive Computing, IEEE, Volume 6, no.3, pp.94-96, c3, July-September 2007

[Miller1987] George W. (Bill) Miller, “Service Level Agreements: Good Fences Make Good Neighbors”, in proceedings of 13th International Computer Measurement Group Conference (CMG’87), pp.553-558, Orlando, FL, USA, December 7-11, 1987

[Motahari2006] Motahari Nezhad, H.R., Benatallah, B., Casati, F. and Toumani, F., “Web Services Interoperability Specifications”, IEEE Computer, Volume 39, 2006

[Murugesan2008] San Murugesan, “Harnessing Green IT: Principles and Practices”, IT Professional, vol. 10, no. 1, pp. 24-33, Jan./Feb. 2008, doi:10.1109/MITP.2008.10

[Nagios2009] Nagios homepage, www.nagios.org, 2009

[Nelson1990] Nelson, V., “Fault-tolerant computing: Fundamental concepts”, IEEE Computer 23(7), pp. 19–25, July 1990

[Newton2009] Newton, "Newton: Component Model", http://newton.codecauldron.org/site/concept/ComponentModel.html, 2009

[Nichols et al 1998] Nichols, et. al., "DiffServ RFC 2474, Definition of the differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", http://tools.ietf.org/html/rfc2474, December 1998

[NimSoft2009] NimSoft, "Nimsoft Monitoring System", http://www.nimsoft.com/solutions/sla-monitoring.php, 2009

[OECD2009] OECD Broadband portal, "Total Broadband Subscribers by country" , Juin 2009 http://www.oecd.org/document/54/0,3343,en_2649_34225_38690102_1_1_1_1,00.html

[Offermans2003] Marcel Offermans, "Apache Felix Dependency Manager", http://felix.apache.org/site/apache-felix-dependency-manager.html, 2003

[OGF2008] OpenGridForum, http://www.gridforum.org/ and GRAAP working group, http://www.ogf.org/gf/group_info/view.php?group=graap-wg , 2008

[Oldham2006] Nicole Oldham, Kunal Verma, Amit P.Sheth and Farshad Hakimpour, "Semantic WS-agreement partner selection", in Proceedings of the 15th international Conference on World Wide Web (WWW'06), Edinburgh, Scotland, May 23 - 26, 2006. ACM, New York, NY, pp.697-706. DOI= http://doi.acm.org/10.1145/1135777.1135879

[OMG2001] OMG, "CORBA Component Model Specification", Version 4.0, http://www.omg.org/docs/formal/06-04-01.pdf, 2001

[OMG2010] Object Management Group, "UML: Unified Modeling Language", http://www.uml.org/, 2010

[OSGi2005] OSGi Alliance, "OSGi Service Platform Core Specification", http://www.osgi.org/, 2005

[Papazoglou et al 2007] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", IEEE Computer, vol. 40, p. 8, 2007.

[Parrend2008] Pierre Parrend, “Software Security Models for Service-Oriented Programming (SOP) Platforms”, thèse de Doctorat d’Informatique de l'Institut National des Sciences Appliquées de Lyon (INSA), Décembre 2008

[Parnas1975] D.L. Parnas, “On the criteria to be used in decomposing systems into modules”, Communication of the ACM, vol. 15, no. 12, pp. 1053–1058, 1972.

[Paschke2005] Paschke, A., “RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML”, in Proceedings of the international Conference on Computational intelligence For Modelling, Control and Automation and international Conference on intelligent Agents, Web Technologies and internet Commerce (Cimca-Iawtic'06), Volume 02, pp.308-314, IEEE Computer Society, ISBN 0-7695-2504-0, Vienna, Austria, November 28 - 30, 2005

[Paschke2006] Adrian Paschke, “Rule-based Knowledge Representation for Service Level Agreement”, The Computing Research Repository (CoRR), September 2006

[Paschke2008] Adrian Paschke, Martin Bichler, “Knowledge representation concepts for automated SLA management”, Decision Support Systems, Volume 46, Number 1, pp.187-205, December 2008

[Peaberry2009] Peaberry project, http://code.google.com/p/peaberry/, 2009

[Pedraza2009a] Gabriel Pedraza, “FOCAS : un canevas extensible pour la construction d’applications orientées procédé”, thèse de doctorat, Université Joseph Fourier, Grenoble, 12 Novembre 2009

[Pedraza2009b] Gabriel Pedraza and Jacky Estublier, “Distributed Orchestration versus Choreography: The FOCAS Approach”, in Proceedings of International Conference on Software Process (ICSP'09), Vancouver, Canada, 16 Mai 2009

[Peltz2003] C. Peltz, “Web services orchestration and choreography,” Computer, vol. 36, no. 10, October, 2003, pp. 46-52.

[Pixar2008] Pixar Animation Studios, “WALL-E”, film d’animation distribué par Walt Disney Pictures, Juin 2008

[Propst1985] John W. Propst, “Developing Effective Service Level Agreements”, in the proceedings of the 11th International Computer Measurement Group Conference, Dallas, TX, USA, December 9-13, 1985

[Qiu et al 2007] Qiu, Z., Zhao, X., Cai, C., and Yang, H., "Towards the theoretical foundation of choreography", in Proceedings of the 16th international Conference on World Wide Web (WWW'07), Banff, Alberta, Canada, May 08 - 12, 2007. ACM, New York, NY, pp.973-982. DOI= http://doi.acm.org/10.1145/1242572.1242704

[Quan2006] Quan, D. M. and Hsu, D. F. , "Network-based resource allocation for Grid Computing within an SLA context", in Proceedings of the Fifth international Conference on Grid and Cooperative Computing, (GCC'06), October 21 - 23, 2006. IEEE Computer Society, Washington, DC, 274-280. DOI= http://dx.doi.org/10.1109/GCC.2006.67

[Racoon 1997] L. B. S. Raccoon, "Fifty years of progress in software engineering", ACM SIGSOFT Software Engineering Notes, v.22 n.1, p.88-104, Jan. 1997, doi: 10.1145/251759.251878

[Rellermeyer2007] J. S. Rellermeyer, G. Alonso, and T. Roscoe, "R-OSGi: Distributed Applications Through Software Modularization", in Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference (Middleware'07), Newport beach, California, USA, 26-30 November 2007

[Rellermeyer et al 2008] Rellermeyer, J.S. et al, "The Software Fabric for the Internet of Things", in proceedings of The Internet of Things, 1st International Conference (IOT'08), Zurich, Switzerland, March 26-28, 2008

[Richardson2006] C. Richardson, "POJOs in Action: Developing Enterprise Applications with Lightweight Frameworks", Manning Publications Co, 2006

[Richardson2007] Leonard Richardson, Sam Ruby, "RESTful Web Services, Web services for the real world", Pages: 446, May 2007, ISBN 10: 0-596-52926-0 | ISBN 13: 9780596529260

[Rio2010] Rio Project, http://www.rio-project.org/, 2010

[Royon2007a] Yvan Royon and Stéphane Frénot, "Multiservice Home Gateways: Business Model, Execution Environment, Management Infrastructure", IEEE Communications Magazine, vol. 45, no. 10, October 2007

[Royon2007b] Yvan Royon, "Environnements d'exécution pour passerelles domestiques", Thèse de Doctorat d'Informatique de l'Institut National des Sciences Appliquées (INSA) de Lyon, Décembre 2007

[Rudametkin et al 2010] Rudametkin, W., Gama, K., Touseau, L., Donsez, D., "Towards a Dynamic and Extensible Middleware for Enhancing Exhibits", in proceedings of IEEE Consumer Communications and Networking Conference (CCNC'10), Las Vegas, Nevada, USA, January 9-12, 2010

[RuleML2010] RuleML, "Rule Markup Language", http://www.csw.inf.fu-berlin.de/ruleml2010/, 2010

[Sahai2001] Sahai, A., D. A., and V. Machiraju, "Towards Automated SLA Management for Web Services", HPL-2001-310R1, HP Labs, Palo Alto, California, 2001

[Saito2010] Nobuo Saito and Kazuhiro Kitagawa, "Special Session on the Integrated and Intelligent Management and Platform for Ecological Home Network", Panel Session for 7th IEEE Consumer Communications & Networking Conference (CCNC'10), Las Vegas, Nevada, January 9-12 2010

[Schmidt et al 2005] M.-T. Schmidt, B. Hutchison, P. Lambros, R. Phippen, "The Enterprise Service Bus: Making service-oriented architecture real", IBM journal, Volume 44, Number 4, Page 781 (2005)

[Seinturier et al 2009] Seinturier, L., Merle, P., Fournier, D., Dolet, N., Schiavoni, V., and Stefani, J-B., "Reconfigurable SCA Applications with the FraSCAti Platform", in Proceedings of the 2009 IEEE international Conference on Services Computing (SCC'09), Symposium on Compiler Construction, September 21 - 25, 2009. IEEE Computer Society, Washington, DC, pp.268-275. DOI= http://dx.doi.org/10.1109/SCC.2009.27

[Shanahan1999] Shanahan, M., "The event calculus explained", In Wooldridge, M.J., Veloso, M.M. (Eds.), Lecture Notes in Computer Science, vol. 1600, pp. 409-430, Springer, Berlin, 1999, ISBN 3-540-66428-9

[Shelby2010] Shelby, Z., Frank, B., "Constrained Application Protocol (CoAP) draft-shelby-core-coap-00", IETF Network Working Group, Internet Draft, March 2010, http://tools.ietf.org/html/draft-shelby-core-coap-00

[Sheng2008] Sheng, Q. Z., Li, X., and Zeadally, S. 2008, "Enabling Next-Generation RFID Applications: Solutions and Challenges", Computer 41, Vol. 9, pp. 21-28, September 2008. DOI=http://dx.doi.org/10.1109/MC.2008.386

[Sherkow1986] Alan M. Sherkow, "Using Service Level Agreements", in proceedings of 12th International Computer Measurement Group Conference (CMG'86), pp. 623-628, Las Vegas, NV, USA, December 8-12, 1986

[Sierra1997] C. Sierra and P. Faratin and N. Jennings, "A Service-Oriented Negotiation Model between Autonomous Agents", in proceedings of the 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-97), 1997

[Skene2009] James Skene, Franco Raimondi, Wolfgang Emmerich, "Service-Level Agreements for Electronic Services", IEEE Transactions on Software Engineering, 17 August 2009. IEEE computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/TSE.2009.55

[Snoonian2003] D. Snoonian, "Smart Building", IEEE Spectrum, Volume 40, Issue 8, pp. 18- 23, August 2003

[Spillner2009] Josef Spillner, Jan Hoyer, "SLA-driven Service Marketplace Monitoring with Grand SLAM", in Proceedings of the 4th International Conference on Software and Data Technologies (ICSOFT 2009), Volume 2, pp. 71-74, Sofia, Bulgaria, July 26-29, 2009. INSTICC Press 2009, ISBN 978-989-674-010-8

[Spring2010] Spring Source, "Spring Framework", http://www.springframework.org, 2010

[Sprint2008] Sprint, "Sprint Titan", http://developer.sprint.com/getDocument.do?docId=98336, 2008

[Szyperski1998] Clemens Szyperski, "Component Software – Beyond Object-Oriented Programming", Addison-Wesley and ACM Press, ISBN 0-201-17888-5, 1998

[TECЛA2009] TECЛA service platform, http://www.serviceplatform.org/, TU Dresden, 2009

[TIBCO2007] TIBCO Software, "Event-Driven SOA: A Better Way to SOA", white paper, http://www.tibco.com/multimedia/wp-event-driven-soa_tcm8-803.pdf, March 2007

[Tivoli2008] IBM, "Tivoli Service Level Advisor", http://www-01.ibm.com/software/tivoli/products/service-level-advisor/, 2008

[Tosic et al 2003] Tosic, V., et al., “Management Applications of the Web Service Offerings Language (WSOL)”, in proceedings of 15th International Conference on Advanced Information Systems Engineering (CaiSE03), Velden, Austria, 2003

[Touseau2006] Lionel Touseau, “Accords de niveau de service dans les plateformes dynamiques à services”, rapport de Master Recherche de l’Univesité Joseph Fourier, Juin 2006

[Touseau2007] Lionel Touseau, “How to Guarantee Service Cooperation in Dynamic Environments?”, in Proceedings of 2007 International Conference on Feature Interactions in Software and Communication Systems (ICFI’07), Grenoble, France, Septembre 2007

[Touseau2008a] Lionel Touseau, Humberto Cervantes and Didier Donsez, “An Architecture Description Language for Dynamic Sensor-Based Applications”, in proceedings of 5th International IEEE Consumer Communications & Networking Conference (CCNC'08), Las Vegas, Nevada, USA, January 2008

[Touseau2008b] Lionel Touseau, Walter Rudametkin and Didier Donsez, “Towards a SLA-based Approach to Handle Service Disruptions”, in proceedings of IEEE International Conference on Services Computing (SCC'08), pp.415-422, Honolulu, Hawaii, USA, 8-11 July 2008

[UPnP1999] Universal Plug and Play Forum, “About the Universal Plug and Play Forum”, http://www.upnp.org/forum/default.htm , 1999

[UPnPAV2008] UPnP MediaServer:3 standardized service and device descriptions, http://www.upnp.org/specs/av/av3.asp, approved on November 28th, 2008

[UPnPQoS2008] UPnP Quality of Service v3, http://www.upnp.org/specs/qos/qos3.asp, standard approved on November, 28th, 2008

[Vinoski2003] S. Vinoski, “Integration with Web Services”, IEEE Internet Computing, vol. 7, no. 6, pp. 75-77, 2003.

[Vouk2008] M. A. Vouk, “Cloud computing: Issues, research and implementations”, in proceedings of 30th International Conference on Information Technology Interfaces (ITI 2008), Cavtat, Croatia, June 23-26, 2008

[Waldner2007] J.-B. Waldner, “Nano-informatique et intelligence ambiante”, 2007

[Waldo1999] J. Waldo, "The Jini architecture for network-centric computing", Communications of the ACM, vol. 42, no. 7, pp. 76-82, 1999.

[Weiser1991] M. Weiser, "The computer for the 21st century", Scientific American, vol. 265, no. 3, pp. 66-75, 1991

[Weiser1993] M. Weiser, "Ubiquitous Computing", IEEE Computer, vol. 26, no. 10, pp. 71-72, 1993

[Williams2008] Williams, B. "What is the Real Business Case for the Internet of Things? ", Synthesis Journal, Information Technology Standards Committee, 2008

[WSDAPI2008] Microsoft, "WSDAPI Client Application and Device Host Development", http://msdn.microsoft.com/en-us/library/bb821828.aspx , 2008.

[WSDL2001] Web Services Description Language (WSDL) 1.1, W3C Note, 15 March 2001, http://www.w3.org/TR/wsdl

[WSDL2007] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation 26 June 2007, http://www.w3.org/TR/wsdl20/

[Yan et al 2008] Yan, L., Zhang, Y., Yang, L.T., Ning, H., "The Internet of things; from RFID to the next-generation pervasive networked systems", Auerbach Publications, 2008

[Zeeb et al 2007] E. Zeeb, et al., "Lessons learned from implementing the Devices Profile for Web Services", published in the proceedings of the IEEE-IES Digital EcoSystems and Technologies Conference, pp 229-232, 2007

[Zhang2004] Zhang, L. and Ardagna, D., "SLA based profit optimization in autonomic computing systems", In Proceedings of the 2nd international Conference on Service Oriented Computing (ICSOC'04), pp.173-182, New York, NY, USA, November 15 - 19, 2004.
DOI= http://doi.acm.org/10.1145/1035167.1035193

[Zschaler2004] Steffen Zschaler and Simone Röttger, "Types of Quality of Service Contracts for Component-Based Systems", in Proceedings of Software Engineering (SE'04), Innsbruck, Austria, February 17–19, 2004

Annexes

X.1 Annexe A
Bundles et composants iPOJO

Les Tableau 10 et Tableau 11 décrivent respectivement les bundles OSGi et les composants iPOJO contenus dans ces bundles.

Bundle	Contenu	Lignes de code
Disruptions logger	Spécifications du DisruptionLogService et du DisruptionsListener	62
Disruptions logger implementation	Composant iPOJO d'implémentation du DisruptionLogger	239
DSLAAdmin	Spécifications des services DSLAAdmin et de négociation, format de SLA, …	594
DSLAAdmin implementation	Composite iPOJO DSLAAdmin • DSLA Negotiator • Compliance Monitor (DSLM) • DSLAAdmin service implementation	620
Handler DSLA require	Handler, méta-données, DSLADependency	662
Handler DSLA provide	Handler, méta-données, DSLAProvider	382

Tableau 10. Contenu des bundles

Composant iPOJO	Services fournis	Services requis	Handlers externes
DisruptionLogger	DisruptionsLog Service		JMX
DSLAAdmin Service Implementation	DSLAAdminService	DSLANegotiator, DSLMFactory, EventAdmin, LogService	JMX
DSLANegotiator Implementation	DSLANegotiator		
Compliance Monitor	DSLMFactory	DisruptionsLog Service	
DSLADependency Handler	N/A	DSLAAdmin	EventAdmin
DSLAProvider Handler	N/A	DSLAAdmin, LogService	EventAdmin

Tableau 11. Détail des composants iPOJO

X.2 Annexe B
Méta-données iPOJO DSLA pour l'application de contrôle du froid dans AspireRFID

X.2.1 Méta-données du consommateur

```
<ipojo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:dsla="fr.liglab.adele.ipojo.handlers.dsla">

<!-- component -->
<component
     classname=fr.liglab.adele.dsla.examples.app.coldchain.TemperatureProducer"
     name="coldchain_app" immediate="true">

     <provides>
           <property name="service.pid" field="m_PID"/>
           <property name="wireadmin.producer.flavors" field="m_flavors" />
           <property name="data.type" type="java.lang.String"
                     value="temperature"/>
           <property name="sensor.location" type="java.lang.String"/>
     </provides>

     <callback transition="validate" method="start" />
     <callback transition="invalidate" method="stop" />

     <properties>
           <property name="flavors" method="setFlavors" />
     </properties>

     <!-- DSLA Configuration -->
     <dsla:requires pid="examples.coldchain.thermostat"
                  name="Cold chain application"
                  description=
       "Cold chain application example to illustrate DSLA binding policy">
           <require field="tempProducer"
                     filter="(service.pid=spot-temp-prod*)">
                  <slo name="maxDisruptionTime" value="1800000"
                    description="MaxTTR" unit="ms" relation.order="lessThan">
                  </slo>
                  <slo name="maxCumulDisruptionTime" value="2400000"
               description="Cumulated MaxTTR"
```

```
            unit="ms" relation.order="lessThan"/>
          </require>
          <require field="coolerService">
                <slo name="maxDisruptionTime" value="1800000"
                    description="MaxTTR" unit="ms" relation.order="lessThan">
                </slo>
                <slo name="maxCumulDisruptionTime" value="2400000"
                      description="Cumulated MaxTTR"
                      unit="ms" relation.order="lessThan"/>
          </require>
    </dsla:requires>

</component>

<instance component="coldchain_app" name="cold-chain-app-inst">
    <property name="flavors" type="array">
          <property value="org.osgi.util.measurement.Measurement"/>
    </property>
    <property name="service.pid"
              value="fr.liglab.adele.dsla.example.coldchain"/>
    <property name="sensor.location" value="fridge"/>
</instance>
</ipojo>
```

X.2.2 Méta-données de l'implémentation fictive du CoolerService

```
<ipojo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:dsla="fr.liglab.adele.ipojo.handlers.dsla">

<!-- component -->
<component classname=
          "fr.liglab.adele.dsla.examples.coldchain.cooler.CoolerServiceImpl"
          name="cooler" immediate="true">

    <provides>
          <property name="service.pid" type="java.lang.String"/>
    </provides>

    <!-- DSLA Configuration -->
    <dsla:provides pid="fr.liglab.adele.dsla.example.coldchain.cooler"
```

```
                  name="Fictive Cooler Service Provider"
                  description="An example of a DSLA service provider"
                   specification=
                 "fr.liglab.adele.dsla.examples.coldchain.cooler.CoolerService"
                  sla.duration="86400000">
            <slo name="maxDisruptionTime" value="1700000" description="MaxTTR"
                 unit="ms" relation.order="lessThan">
                  <extension name="service.level" value="PREMIUM"/>
                  <extension name="value.delta" value="1000"/>
            </slo>
            <slo name="maxCumulDisruptionTime" value="2200000"
                 description="Cumulated MaxTTR"
                 unit="ms" relation.order="lessThan">
                  <extension name="value.delta" value="1000"/>
            </slo>
    </dsla:provides>
</component>

<instance component="cooler" name="cooler-inst">
    <property name="service.pid"
              value="fr.liglab.adele.dsla.example.coldchain.cooler"/>
    </instance>
</ipojo>
```

X.3 Annexe C
Descripteur de WireApp pour AspireRFID

```
<?xml version="1.0" encoding="UTF-8"?>
<wireapp
    id="org.ow2.aspirerfid.sensor.wireadminbinder.sensors"
      description="A wired application that binds several sensor
                   measurements producers to AspireRFID">

    <!-- a one-to-many wireset without wire properties -->
    <wireset
          id="temperature-producers2ALE-consumer"
          description="a simple wire between ALE measurement consumer
                       and temperature producers"
          producers-filter="(&amp;(wireadmin.producer.flavors=
                            *org.osgi.util.measurement.Measurement)
                            (data.type=temperature))"
          consumers-filter="(service.pid=
                            org.ow2.aspirerfid.service.aleconsumer)"
          removepolicy="ifDisconnected"
    />
</wireapp>
```

Résumé

L'informatique s'est récemment développée autour de deux axes : l'informatique ambiante d'une part avec la multiplication des objets communicants, et l'internet des services d'autre part suite à l'essor parallèle des centres de traitement de données et d'Internet. Dans ces domaines, la disponibilité fluctuante des ressources, qui entraîne une intermittence des services fournis, représente désormais une préoccupation majeure dans la conception d'applications.

La programmation orientée composants appliquée aux architectures orientées service simplifie la gestion des liaisons de service via des politiques. Néanmoins les politiques de liaison existantes suivent soit une approche statique interdisant alors toute reconfiguration d'architecture en cours d'exécution, soit une approche dynamique ne garantissant pas une stabilité minimale de l'architecture dans le cas de services intermittents.

Cette thèse propose un compromis entre stabilité architecturale et dynamisme en plaçant l'interruption de service au centre des préoccupations du concepteur. La politique de liaison résultante offre ainsi une tolérance aux interruptions de service jusqu'à une certaine limite au-delà de laquelle l'architecture est reconfigurée dynamiquement. Afin de situer cette limite, notre proposition se base sur l'utilisation d'accords de niveau de service, une forme enrichie des contrats de service permettant, entre autres, l'expression de contraintes sur la disponibilité des services.

L'approche a été expérimentée sur la plate-forme à services OSGi en étendant les mécanismes de gestion des liaisons du modèle à composants iPOJO, puis validée dans le contexte de l'informatique ambiante ainsi que sur le serveur d'applications JOnAS.

Mots clé : architecture dynamique orientée service, disponibilité, service intermittent, composant orienté service, politique de liaison, accord de niveau de service

Abstract

Lately, the evolution of information technologies has been following two trends. On the one hand the proliferation of communicating devices contributes to the creation of an ambient intelligence. On the other hand, the booming of Internet associated with the rapid growth of data centres capabilities results in the emergence of an internet of services. In both domains, application design is challenged by the dynamic availability of computing resources and data.

The combination of component-based software engineering and service-oriented computing techniques allows service bindings to be driven by policies. However, for the time being, policies either follow a dynamic approach which does not suit the needs of architectural stability when dealing with intermittent services, or a static approach which does not allow dynamic reconfiguration.

The work presented in this thesis proposes a trade-off between the two approaches by considering service disruptions as a major concern. The proposed binding policy relies on service level agreements to be disruption-tolerant, since service-level agreements allow expressing and enforcing obligations regarding availability and quantified disruptions.

This approach has been implemented on the OSGi service platform and iPOJO, a service-oriented component model for OSGi. iPOJO service dependency management has been extended in order to support our policy. The latter was validated both in the context of ambient intelligence, and on open-source and OSGi-based JOnAS application server.

Keywords : dynamic service-oriented architecture, availability, intermittent service, service-oriented component, binding policy, service-level agreement