HAL
open science

Orchestration de services hétérogènes et sécurisés

Stéphanie Chollet

To cite this version:

Stéphanie Chollet. Orchestration de services hétérogènes et sécurisés. Génie logiciel [cs.SE]. Université Joseph-Fourier - Grenoble I, 2009. Français. NNT : . tel-00544420

HAL Id: tel-00544420

https://theses.hal.science/tel-00544420

Submitted on 8 Dec 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 JOSEPH FOURIER (GRENOBLE I)

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER

Discipline : **INFORMATIQUE**

soutenue et présentée publiquement par

Stéphanie CHOLLET

le 1er décembre 2009

ORCHESTRATION DE SERVICES HÉTÉROGÈNES ET SÉCURISÉS

Directeur de thèse :

Philippe LALANDA

JURY

Président	Ioannis PARISSIS	Professeur à Grenoble INP
Rapporteurs	Pierre-Alain MULLER	Professeur à l'Université de Haute Alsace, Mulhouse
	Fabio CASATI	Professeur à l'Université de Trento
Examinateurs	David HILL	Professeur à l'Université Blaise Pascal, Clermont-Ferrand
	Alexandre LEFEBVRE	Directeur R&D, France Telecom
Encadrant	Philippe LALANDA	Professeur à l'Université Joseph Fourier, Grenoble

Thèse préparée au sein du Laboratoire d'Informatique de Grenoble dans l'équipe ADELE

Remerciements

Au terme de ce travail, je tiens à remercier tous ceux sans qui il n'aurait pu devenir ce qu'il est. Je pense tout d'abord à ceux qui ont accepté d'être les membres de mon jury, qui ont pris la peine de me suivre dans ma démarche, de l'évaluer et de la reconnaître. J'ai pu profiter des observations judicieuses de M. Pierre-Alain Muller et de M. Fabio Casati, rapporteurs ; je tiens à leur dire combien je leur en sais gré comme des remarques des examinateurs, M. Alexandre Lefebvre et M. David Hill, dont les cours à l'Université Blaise Pascal de Clermont-Ferrand m'ont ouvert l'esprit à la recherche et à l'ingénierie dirigée par les modèles. Je veux dire à M. Ioannis Parissis, président du jury, que je me suis sentie encouragée pendant toute la soutenance par son attention bienveillante, que j'avais pu apprécier pour avoir suivi ses enseignements à l'Université Joseph Fourier de Grenoble.

Ce travail a été proposé et dirigé par M. Philippe Lalanda, dont j'ai pu apprécier la patience, l'écoute, la compétence, la courtoisie, l'humour et la rigueur tout au long de ces trois années consacrées à cette recherche. Qu'il trouve ici l'expression de ma profonde reconnaissance.

Que soient aussi remerciés M. Jacky Estublier, M. Philippe Lalanda et M. Pierre-Yves Cunin, responsables de l'équipe Adèle, pour m'avoir accueillie au sein de leur équipe, ce qui m'a permis de travailler dans les meilleures conditions. Je ne saurais oublier ceux avec qui j'ai partagé les bons moments ou les doutes et les inquiétudes propres à ce genre d'activité, que ce soit au quotidien comme avec Vincent, Cristina, Mikael, Etienne, Thomas, Eric, Yoann, Mehdi, Ada, Gabriel, German, Lionel... ou de façon plus espacée comme avec Pierre, Julien et Yoan dont la sympathie était d'autant plus sincère qu'ils étaient engagés dans la même voie. Un grand merci aussi pour mes deux stagiaires Romain et Paul pour m'avoir aidée et encouragée.

Ce serait faire preuve d'ingratitude ici que de ne pas citer mon père, ma mère, ma sœur et Olivier, qui a eu le privilège de suivre jour après jour l'avancement de mon travail et de la courbe de mon moral.

Résumé

Récemment, l'approche à services est apparue en ayant pour but de construire des applications à partir d'entités logicielles, nommées services. Un service fournit un ensemble de fonctionnalités définies par une description de services. A partir de cette description, un consommateur de service peut rechercher un service qui corresponde à ses besoins, le sélectionner et l'invoquer. La construction d'applications par composition de services demeure néanmoins une activité complexe puisqu'il faut traiter conjointement les aspects métier et techniques ; la composition doit satisfaire aux exigences fonctionnelles et non-fonctionnelles ainsi que respecter les contraintes des technologies à services liées, notamment, à l'hétérogénéité des plates-formes. Par ailleurs, les points forts de l'architecture à services, qui sont la distribution et le déploiement des services sur des plates-formes hétérogènes, ouvrent d'importantes failles de sécurité.

Nous proposons une approche dirigée par les modèles pour simplifier la réalisation d'applications basées sur une orchestration de services hétérogènes en prenant en considération les aspects de sécurité dès l'étape de conception. Pour cela, nous avons défini deux méta-modèles : l'un pour l'orchestration de services et l'autre pour la sécurité, ainsi que des liens entre ces méta-modèles dans le but d'étendre l'orchestration avec des propriétés de sécurité. Ainsi, il est possible de réaliser des modèles d'orchestration de services hétérogènes et sécurisés conformes aux méta-modèles. A partir de ces modèles, nous générons le code nécessaire à l'exécution de l'orchestration. L'exécution se fait en fonction des modèles définis dans la phase de conception et des services disponibles qui répondent aux spécifications. Notre approche a été validée avec la plate-forme Secure FOCAS, qui a été réalisée dans le cadre du projet Européen ITEA SODA.

Mots-clé : Approche orientée service, composition de services, sécurité, approche générative.

Abstract

Service-oriented Computing (SOC) has appeared recently as a new software engineering paradigm. The very purpose of this reuse-based approach is to build applications through the late composition of independent software elements, called services, which are made available at run-time by internal or external providers. SOC brings properties of major interest. First, it supports rapid application development. Using existing, already tested, services is likely to reduce the time needed to build up an application and the overall quality of this application. SOC also improves software flexibility through late binding. A service to be used by an application is chosen at the last moment, based on its actual availability and on its properties at that moment.

The service orientation has also to face thorny problems, as in any reuse-based approach. In this work, we focus on two major issues: the integration of heterogeneous service-oriented technologies and the management of security aspects when invoking a service. Security is actually a major concern to SOC practitioners. SOC technologies have allowed companies to expose applications, internally and externally, and, for that reason are heavily used. However, in some distributed environments, software services and process engines can be alarmingly vulnerable. Service-based processes can expose organizations to a considerable amount of security risk and dependability degradation.

We propose to use a model-driven approach for solving this problem. During system design, paradigms such as abstraction, separation of concerns and language definition are used to define a model of the service composition with security properties. This model is transformed into an execution model. We present a generative environment applying these principles for service composition. This environment has been built as part of the SODA European project and validated on several industrial use cases.

Keywords: Service-oriented approach, service composition, security, generative approach.

SOMMAIRE

1 Introduction **21**

1 Problématique 22
2 Objectifs 23
3 Organisation du document 24

I Etat de l'art **27**

2 L'approche orientée service et son utilisation **29**

1 Services : définitions et architecture 30
1.1 Services 30
1.2 SOC : *Service-Oriented Computing* 32
1.3 SOA : *Service-Oriented Architecture* 34
1.4 Besoins 35
1.5 SOC et le dynamisme 37
1.6 Caractérisation 38
2 Composition de services 39
2.1 Définitions 39
2.2 Composition par procédés 40
2.3 Composition structurelle 42

3 Services Web 43
3.1 Principes 43
3.2 WSDL : le langage de description des services Web 44
3.3 UDDI : l'annuaire de services Web 45
3.4 SOAP : les communications entre services Web 46
3.5 WS-BPEL 47
3.6 Synthèse 49
4 Composants orientés service 50
4.1 Principes 50
4.2 OSGi™ 54
4.3 iPOJO 56
4.4 Synthèse 58
5 Autres technologies à services 59
5.1 CORBA et Jini 59
5.2 UPnP et DPWS 63
5.3 SCA 67
6 Synthèse 69

3 La sécurité pour les services 71

1 Attaques et menaces dans les architectures à services 72
1.1 Attaque passive d'écoute du réseau et d'analyse du trafic 73
1.2 Tromperie 74
1.3 Détournement d'informations 75
1.4 Déni de service 76
1.5 Synthèse des attaques et des menaces 77
2 Sécurité : objectif, définition et concepts 78
2.1 Principe et définition 78
2.2 Concepts 79
2.3 Classification des solutions techniques 80
3 Techniques de base de sécurité 82
3.1 Chiffrement et déchiffrement 82
3.2 Nom d'utilisateur et mot de passe 83
3.3 Signature électronique 84

3.4 Certificat électronique 86
3.5 Synthèse 88
4 Technologies de sécurité 88
4.1 XML Digital Signature et XML Encryption 88
4.2 XKMS 96
4.3 XACML et XrML 97
4.4 Authentification unique et SAML 98
4.5 Synthèse 100
5 WS-Security : technologie pour les services Web 101
5.1 Principes 101
5.2 Authentification avec WS-Security 102
5.3 Signature avec WS-Security 103
5.4 Chiffrement avec WS-Security 104
5.5 WSS4J : *Web Services Security for Java* 105
5.6 Le *framework* WS-Security 108
6 Synthèse 110

II Contribution 111

4 Proposition 113

1 Problématique et objectifs 114
2 Introduction à l'Ingénierie Dirigée par les Modèles 115
2.1 Concept de modèle 115
2.2 Concept de méta-modèle 117
2.3 Composition de modèles 118
3 Notre approche 119
3.1 Présentation générale 119
3.2 Partie conception 121
3.3 Partie exécution 127

5 Méta-modèles et modèles 131

1 Méta-modèle de l'orchestration de services hétérogènes 132
2 Méta-modèle de la sécurité 136

3 Composition des méta-modèles 138
4 Modèles de l'orchestration et de la sécurité 140
5 Génération de code 142
5.1 Génération du code fonctionnel 142
5.2 Génération du code de sélection 142
5.3 Génération du code de sécurité 143
6 Synthèse 146

6 Réalisation et expérimentations 147

1 Plate-forme FOCAS 148
2 Extension de sécurité pour FOCAS : Secure FOCAS 151
2.1 Présentation générale 151
2.2 Extension de sécurité 152
2.3 Extension de sélection retardée 156
3 Cas d'utilisation 159
3.1 Présentation générale 159
3.2 En langage APEL 160
3.3 Cas d'utilisation sécurisé 162
3.4 Dans Secure FOCAS 163
4 Expérimentations 164
4.1 L'orchestration statique et dynamique 164
4.2 La génération du code métier et de sécurité 165
4.3 L'apprentissage des utilisateurs 166
5 Synthèse 171

7 Extensions et perspectives 173

1 Gestion des identités 174
1.1 Dans notre approche 174
1.2 Dans l'outil Secure FOCAS 176
1.3 Expérimentations 177
1.4 Synthèse 178
2 Ajout de nouvelles propriétés non-fonctionnelles 179
2.1 Dans notre approche 179

2.2 Mise en application avec deux propriétés non-fonctionnelles 182
2.3 Ajout dans la plate-forme Secure FOCAS 186
2.4 Synthèse 188

8 Conclusion **189**

Bibliographie **193**

Les figures

1 Introduction **21**

2 L'approche orientée service et son utilisation **29**

2.1 Acteurs et interactions dans l'architecture à services. 33
2.2 Mécanismes nécessaires pour un environnement d'intégration de services. 34
2.3 Fonctionnalités d'une architecture orientée service étendue. 36
2.4 Interactions de l'approche à services dynamique. 37
2.5 Composition de services. 39
2.6 Chorégraphie de services. 40
2.7 Orchestration de services. 41
2.8 Composition structurelle. 42
2.9 Architecture pour les services Web. 43
2.10 Le fichier WSDL. 45
2.11 Exemple de message SOAP pour interroger un service Web. 47
2.12 Exemple d'un procédé WS-BPEL. 48
2.13 Exemple d'assemblage de composants. 51
2.14 Structure d'un composant. 52
2.15 Cycle de vie d'un *bundle* OSGi™. 54

2.16 Architecture de OSGi™. 55
2.17 Architecture de iPOJO. 56
2.18 Architecture de CORBA. 60
2.19 Architecture de Jini. 61
2.20 Architecture de UPnP. 63
2.21 Architecture de DPWS. 65
2.22 Structure d'un composant SCA. 67
2.23 Exemple de composition de composants SCA. 68

3 La sécurité pour les services 71

3.1 Communications à travers différentes couches techniques. 72
3.2 Divulgation d'informations. 74
3.3 Tromperie sur l'identité du client. 74
3.4 Détournement d'informations. 75
3.5 Interruption de service. 76
3.6 Cycle de vie sécurisé d'un système. 78
3.7 Principe de base du chiffrement. 82
3.8 Les chiffrements symétrique et asymétrique. 83
3.9 Principe de l'authentification par mot de passe. 84
3.10 Signature d'un document. 85
3.11 Réception du document avec sa signature. 85
3.12 Exemple de certificat X.509. 86
3.13 Fonctionnement d'une IGC. 87
3.14 Exemple d'écriture d'un noeud en langage XML. 89
3.15 Les différents types de signature. 90
3.16 Structure d'une signature avec XML Digital Signature. 91
3.17 Exemple de signature respectant le standard XML Digital Signature. 92
3.18 Structure d'un chiffrement avec XML Encryption. 94
3.19 Exemple de chiffrement respectant le standard XML Encryption. 94
3.20 Principe de fonctionnement de XKMS. 96
3.21 Structure de l'authentification par nom d'utilisateur. 102
3.22 Structure d'une signature dans un message SOAP. 103
3.23 Structure d'un message SOAP chiffré. 104

3.24 Echange de messages avec Apache Axis. 105
3.25 Requête SOAP d'interrogation du service Web. 106
3.26 Réponse SOAP du service Web. 106
3.27 Principe de sécurisation avec la bibliothèque WSS4J. 107
3.28 Pile de standards. 108

4 Proposition 113

4.1 Relation de représentation. 116
4.2 Relation de conformité. 118
4.3 Principes de notre approche. 119
4.4 Montée en abstraction pour les services. 121
4.5 Modèle d'orchestration de services abstraits. 122
4.6 Séparation des modèles. 124
4.7 Composition des méta-modèles. 125
4.8 Synthèse de la partie conception. 126
4.9 Modèle d'exécution composé de deux modèles. 128
4.10 Génération de code en deux phases. 130

5 Méta-modèles et modèles 131

5.1 Mise en application de notre approche. 132
5.2 Méta-modèle de l'orchestration de services abstraits hétérogènes. 133
5.3 Exemple de description de service au format WSDL et UPnP. 134
5.4 Expression d'un procédé en langage APEL. 135
5.5 Méta-modèle de la sécurité. 136
5.6 Composition des deux méta-modèles. 139
5.7 Un modèle d'orchestration selon la syntaxe graphique de APEL. 140
5.8 Un modèle d'orchestration de services hétérogènes. 140
5.9 Un modèle de sécurité. 141
5.10 Composition des modèles. 141
5.11 Règles d'implication pour les activités composites. 141
5.12 Modèle d'exécution. 143
5.13 L'authentification dans les messages. 144

5.14 L'intégrité dans les messages. 144
5.15 L'audit des messages. 145

6 Réalisation et expérimentations 147

6.1 Environnement de conception FOCAS. 149
6.2 Interface pour l'exécution. 149
6.3 Capture d'écran du logiciel Secure FOCAS. 152
6.4 Capture d'écran de l'onglet *Non-Functional Properties*. 153
6.5 Capture d'écran de l'onglet *Security Details*. 154
6.6 Extrait du code du fichier de déploiement pour le client. 155
6.7 Capture d'écran de l'onglet de sélection. 157
6.8 Architecture pour la sélection dynamique de services. 157
6.9 Spécification de la chaîne d'acquisition. 159
6.10 Chaîne d'acquisition exprimée avec APEL. 160
6.11 Détail de l'activité *Acquisition*. 161
6.12 Détail de l'activité *Processing*. 161
6.13 Spécification de la chaîne d'acquisition sécurisée. 162
6.14 Onglet *Non-Functional Properties* pour sécuriser l'activité *Analysis*. 163
6.15 Onglet *Security Details* pour sécuriser l'activité *Analysis*. 163
6.16 Scénario de test. 165
6.17 Pourcentage de lignes de code générées et écrites. 166
6.18 Représentation des différents temps obtenus pour les trois scénarios. . . . 168
6.19 Temps obtenus pour le scénario 1 pour des testeurs débutants et experts. . 169
6.20 Temps obtenus pour le scénario 2 pour des testeurs débutants et experts. . 170
6.21 Temps obtenus pour le scénario 3 pour des testeurs débutants et experts. . 170

7 Extensions et perspectives 173

7.1 Architecture pour la gestion des identités des utilisateurs. 175
7.2 Capture d'écran de l'onglet de sécurité détaillée avec la gestion des identités. 176
7.3 Extrait du code du fichier *template* pour la recherche d'identité. 177
7.4 Séparations des modèles. 180
7.5 Composition des méta-modèles. 181

7.6 Modèle d'exécution. 181
7.7 Méta-modèle de la journalisation. 182
7.8 Méta-modèle pour la qualité de service. 183
7.9 Composition des méta-modèles de l'orchestration et de la journalisation d'événements. 184
7.10 Composition des méta-modèles de l'orchestration et de la qualité de service. 184
7.11 Modèle d'exécution avec la qualité de service. 185
7.12 Impact de la journalisation d'événements sur la génération de code. 186
7.13 Onglet général pour les propriétés non-fonctionnelles. 187
7.14 Onglet des détails techniques pour la journalisation d'événements. 187

8 Conclusion **189**

8.1 Notre approche globale. 190

Les tables

1 Introduction **21**

2 L'approche orientée service et son utilisation **29**

2.1 Récapitulatif de la technologie des services Web. 49
2.2 Comparatif des technologies OSGi™ et iPOJO. 58
2.3 Comparatif des technologies CORBA et Jini. 62
2.4 Comparatif des technologies UPnP et DPWS. 66

3 La sécurité pour les services **71**

3.1 Récapitulatif des attaques et des menaces dans les architectures à services. 77
3.2 Mise en relation des concepts et des solutions techniques. 80
3.3 Synthèse des techniques de sécurité pour les différents concepts. 88
3.4 Récapitulatif des différentes technologies de sécurité. 100
3.5 Propriétés et techniques supportées par la recommandation WS-Security. . 101

4 Proposition **113**

5 Méta-modèles et modèles **131**

6 Réalisation et expérimentations **147**

6.1 Nombre de lignes de code générées et écrites pour les deux types de sélection. 164
6.2 Nombre de lignes de code générées et écrites pour les propriétés de sécurité. 165
6.3 Les scénarios de tests. 167
6.4 Temps obtenus par les testeurs pour chacun des scénarios. 167
6.5 Temps obtenus par les testeurs experts pour chacun des scénarios. 169

7 Extensions et perspectives **173**

7.1 Lignes de code générées et écrites avec la gestion des identités. 177

8 Conclusion **189**

1

Introduction

1 Problématique

L'évolution des méthodes et des technologies en génie logiciel fait apparaître régulièrement de nouvelles approches pour le développement d'applications. Ainsi, on a vu l'apparition des approches à objets [Tay98] dans les années 1980, puis à composants [Szy02] au cours des années 2000. Le but de ces nouvelles approches est de combler les lacunes et les failles des approches précédentes. Une nouvelle approche s'appuie souvent sur un nouveau paradigme. En général, les nouveaux paradigmes ne remplacent pas complètement les anciens ; au contraire, ils essaient de s'appuyer sur les anciens tout en ajoutant ou modifiant certains éléments. Ils reposent généralement sur les principes fondamentaux que sont l'abstraction, la modularité et la séparation des préoccupations.

Récemment, nous avons vu apparaître l'approche à services (*Service-Oriented Computing*) [Pap03], qui cherche à combler certaines faiblesses de l'approche à composants. L'élément-clé de cette nouvelle approche est le service qui peut être vu comme une entité logicielle mise à disposition de tiers. Un service fournit un ensemble de fonctionnalités définies dans une description de service. Cette description comporte des informations sur la partie fonctionnelle mais aussi sur les aspects non-fonctionnels qu'il fournit et/ou requiert. A partir de cette spécification, un consommateur potentiel peut rechercher un service qui corresponde à ses besoins, le sélectionner et l'invoquer.

Le but de cette approche est de traiter la construction d'applications comme une composition de services. Au delà de la réutilisation, un avantage certain de cette approche est d'aboutir à des architectures d'applications faiblement couplées. De fait, seule la description du service est partagée par les consommateurs et les fournisseurs de services. De plus, l'hétérogénéité des plates-formes et des implantations est masquée au consommateur du service, tout comme sa localisation.

Les principes de l'approche à services ont déjà été mis en œuvre par différentes technologies et utilisés dans des domaines variés. Les services Web [W3C04b], par exemple, sont une technologie particulière qui met en œuvre une grande partie des principes de l'approche à services, notamment pour l'intégration d'applications. Les technologies DPWS [Mic06], UPnP [UPn08] et OSGi™ [OSG07] ont été, quant à elles, développées pour les applications du domaine de l'informatique *pervasive* et sont aujourd'hui fortement répandues.

La construction d'applications par composition de services demeure néanmoins **une activité complexe**. Il s'agit, en effet, de traiter conjointement des aspects métier et techniques ; la composition doit satisfaire les exigences fonctionnelles et non-fonctionnelles ainsi que respecter les contraintes des technologies à services liées, notamment, à l'hétérogénéité des plates-formes.

Par ailleurs, les points forts de l'architecture à services, qui sont la distribution et le déploiement des services sur des plates-formes hétérogènes, ouvrent **des failles de sécurité** importantes. En effet, il existe des risques d'écoute, de vol et de divulgation des informations qui circulent entre les différents acteurs de l'approche à services. Ces failles

de sécurité nécessitent de mettre en place des mécanismes de sécurité matériels mais aussi logiciels.

Au niveau des services, le principe est de sécuriser les messages qui sont échangés. Il existe plusieurs techniques de sécurité pour assurer l'intégrité et/ou la confidentialité d'un message mais aussi pour authentifier l'émetteur d'un message. Ces techniques de sécurité ont été adaptées aux technologies qui se sont développées pour le Web et, en particulier, pour les technologies à services. Ainsi, la technologie des services Web propose d'ores et déjà une recommandation pour sécuriser les messages échangés. Nous constatons alors que **la complexité des techniques et des technologies de sécurité** s'ajoute à la complexité de construction d'applications à services précédemment mentionnée.

De plus, l'ajout de propriétés de sécurité est souvent fait de façon ad hoc, par l'insertion de code par les développeurs. Cette tâche est délicate puisque le développeur doit avoir une connaissance de la partie métier (partie fonctionnelle) mais aussi une connaissance de la sécurité pour l'insérer à l'endroit approprié dans le code fonctionnel. A cette complexité s'ajoute le fait que cette tâche est souvent répétitive et par conséquent source d'erreurs.

2 Objectifs

Nous pouvons donc constater qu'actuellement l'utilisation de l'approche à services pour réaliser des applications nécessite l'intervention de nombreux experts de façon à aborder tous les aspects fonctionnels et techniques. Les outils permettant la composition de services sont actuellement dédiés à une unique technologie à services. L'intégration de services hétérogènes est alors complexe, les outils sont difficilement adaptables à différentes technologies à services. De plus, les aspects de sécurité ne sont pas pris à en considération dans la majorité des environnements de conception. Le code de sécurité est implanté directement dans le code fonctionnel et celui-ci n'est, par conséquent, pas réutilisable.

L'objectif principal de cette thèse est de simplifier la réalisation d'application à base de services hétérogènes en prenant en considération les aspects de sécurité dès l'étape de conception. Nous considérons qu'une composition de services doit être simple, rapide et automatisée. Il doit en être de même pour l'ajout de la sécurité. Nous souhaitons donc cacher la complexité technique des différentes technologies à services mais aussi des technologies de sécurité pour que les utilisateurs ne se concentrent que sur la partie métier de leur application.

Dans cette thèse, nous proposons une approche générative et outillée qui permette de réaliser des compositions de services hétérogènes avec des caractéristiques de sécurité. L'idée est d'utiliser les principes de l'Ingénierie Dirigée par les Modèles pour spécifier la composition de services avec la sécurité de façon abstraite afin de cacher les détails techniques d'implantation. Le code exécutable est alors généré à partir de cette spécification

abstraite. Notre approche a été validée avec la plate-forme Secure FOCAS, qui met en œuvre les différents éléments de notre approche. Cet outil a été réalisé dans le cadre du projet Européen ITEA SODA (`http://www.soda-itea.org/`).

3 Organisation du document

Après cette introduction, ce document est divisé deux grandes parties, que sont l'état de l'art et la contribution. L'état de l'art est présenté en deux chapitres :

- le **Chapitre 2** présente l'approche orientée services, en commençant par une définition de la notion de service. Le deuxième point abordé dans ce chapitre est la composition des services, qui est actuellement un défi très important lancé par l'approche à services. Nous allons ensuite faire un point sur les services Web, qui sont la technologie à services la plus utilisée actuellement et qui propose un langage de composition. Par la suite, plusieurs technologies à services seront détaillées en fonction de leur domaine d'application et de leur type de composition.
- le **Chapitre 3** propose une vision particulière du domaine de la sécurité appliqué à l'approche à services. La première partie montre quelles sont les attaques et les menaces qui existent dans les architectures à services. Ensuite, nous définissons quels sont les objectifs et les solutions techniques du domaine. La troisième partie est un rappel des techniques de base existantes et reconnues pour assurer la sécurité d'un système. Par la suite, nous détaillons les technologies de sécurité qui ont été proposées suite au développement du Web. Nous terminerons par la recommandation WS-Security qui est une proposition pour sécuriser des services Web.

La deuxième grande partie concerne notre contribution. Elle est répartie en quatre chapitres :

- le **Chapitre 4** expose la problématique, les objectifs de cette thèse ainsi que notre approche pour résoudre le problème de la composition de services sécurisés. Nous proposons de suivre une approche dirigée par les modèles. Nous faisons donc, dans un premier temps, une brève introduction aux concepts du domaine de l'ingénierie dirigée par les modèles avant de détailler, dans un deuxième temps, les principes de notre approche.
- le **Chapitre 5** est la mise en application de l'approche présentée dans le chapitre précédent. Dans ce chapitre, nous présentons les méta-modèles que nous avons développés pour l'orchestration de services abstraits hétérogènes et pour la sécurité. Nous définissons l'impact de la composition de ces méta-modèles sur la sélection des services et sur la génération du code fonctionnel et du code de sécurité.
- le **Chapitre 6** présente notre environnement Secure FOCAS. Nous mettons en application les principes de notre approche dans cet environnement. La deuxième

partie de ce chapitre se concentre sur un cas d'application issu du projet Européen ITEA SODA et sur les résultats que nous avons obtenus.

– dans le **Chapitre 7**, nous détaillons deux extensions qui sont, d'une part, la gestion des identités et, d'autre part, l'extension de notre approche à plusieurs propriétés non-fonctionnelles. Nous présentons pour chacune de ces extensions les principes pour les intégrer dans notre approche et les premiers résultats que nous avons obtenus.

Enfin, le **Chapitre 8** synthétise les principales idées de notre proposition. Nous en profitons pour faire le point sur nos principales contributions et aussi sur les perspectives possibles de nos travaux.

Première partie

Etat de l'art

2

L'APPROCHE ORIENTÉE SERVICE ET SON UTILISATION

Dans une première partie, nous allons présenter l'approche à services et les besoins qui en découlent. Après avoir mis en avant les éléments caractéristiques de cette approche, nous étudierons, dans une deuxième partie, les deux types de compositions de services : par procédés et structurelle. Dans la troisième partie, nous nous focaliserons sur une technologie particulière que sont les services Web. Ces derniers permettent de mettre en œuvre une grande partie des principes de l'approche à services. Dans une quatrième partie, nous présenterons les principes des composants orientés service, qui regroupent les avantages des composants et ceux des services et nous exposerons aussi deux technologies OSGi™ et iPOJO, qui utilisent ses principes. Avant de conclure, nous ferons le point sur les différentes technologies à services qui existent, en fonction de leur domaine d'utilisation.

1 Services : définitions et architecture

L'approche à services est une approche relativement récente qui présente un certain nombre d'avantages pour la réalisation d'applications. Nous allons, dans un premier temps, définir l'élément-clé de l'approche à services, c'est-à-dire le service. Ensuite, nous présenterons les différents acteurs et leurs interactions pour cette approche. Dans une troisième partie, nous détaillerons l'architecture à services. Nous expliquerons aussi les besoins qui en découlent. Finalement, nous mettrons en évidence les éléments spécifiques qui nous permettront par la suite de caractériser les différentes technologies qui mettent en œuvre cette approche.

1.1 Services

La notion de service reste floue. M.P. Papazoglou a proposé la définition suivante :

> « *Services are self-describing, platform agnostic computational elements.* » [Pap03]

Un service est vu comme une entité logicielle qui peut être utilisée grâce à sa description. Le consommateur du service l'utilise sans avoir connaissance de la technologie sous-jacente pour son implantation ainsi que de sa plate-forme d'exécution. De plus, le service ne connaît pas le contexte dans lequel il va être utilisé par le client. Cette indépendance à double sens est une propriété forte des services qui facilite le faible couplage. Cette définition contient une faiblesse : elle sous-entend qu'un service est exécuté sur une plate-forme distante et qu'il ne peut pas être importé sur une plate-forme locale.

Le consortium OASIS [1], qui est en charge de la normalisation et de la standardisation des applications Internet, a aussi donné une définition du terme service :

> « *A service is a mechanism to enable access to one or more capabilities, where the access is provided by using a prescribed interface and is exercised consistent with constraints and policies as specified by the service composition. A service is accessed by means of a service interface where the interface comprises the specifics of how to access the underlying capabilities. There are no constraints on what constitutes the underlying capability or how access is implemented by the service provider. A service is opaque in that its implementation is typically hidden from the service consumer except for (1) the information and behaviour models exposed through the service interface and (2) the information required by service consumers to determine whether a given service is appropriate for their needs.* » [OAS06a]

1. *Organization for the Advancement of Structured Information Standards*, `http://www.oasis-open.org/home/index.php`.

Pour le consortium OASIS, un service fournit un ensemble de fonctionnalités décrites dans une spécification, appelée interface, ainsi qu'un ensemble de contraintes et de politiques d'accès aux fonctionnalités offertes. L'implantation du service n'est pas visible pour l'utilisateur. Seules les informations qui peuvent permettre de savoir si le service correspond aux besoins de l'utilisateur sont disponibles. Nous pouvons noter qu'il est cependant difficile de discerner une information utile d'une information inutile pour le choix d'un service.

A. Arsanjani apporte une autre définition de service. A la différence des autres, il met en avant les principales interactions qui permettent l'utilisation des fonctionnalités du service :

> « *A service is a software resource (discoverable) with an externalized service description. This service description is available for searching, binding, and invocation by a service consumer. Services should be ideally be governed by declarative policies and thus support a dynamically reconfigurable architectural style.* » [Ars04]

Cette définition s'intéresse à l'utilité de la description du service. Dans un premier temps, le client recherche le service correspondant à ses besoins en fonction de la description de service fournie. Ensuite, il fait la liaison avec lui et il l'invoque. Ces actions peuvent être réalisées avant ou pendant l'exécution de l'application à services.

Pour rendre plus explicites ces définitions, nous pouvons dire qu'un service est :

> « *une entité logicielle qui fournit un ensemble de fonctionnalités définies dans une description de service. Cette description comporte des informations sur la partie fonctionnelle du service mais aussi sur ses aspects non-fonctionnels. A partir de cette spécification, un consommateur de service peut rechercher un service qui correspond à ses besoins, le sélectionner et l'invoquer en respectant le contrat qui a été accepté par les deux parties.* »

Nous avons introduit dans cette définition la notion de contrat. Un contrat entre deux parties permet de s'assurer que chacune respectera ce à quoi elle s'est engagée. Un contrat est le résultat d'une négociation entre le fournisseur et le consommateur. Ce contrat représente un accord de niveau de service, en anglais *Service Level Agreement* (SLA) [AFM05], qui définit les engagements que prend le fournisseur sur la qualité de son service, et les pénalités encourues en cas de manquement. Cette qualité doit être mesurable et mesurée selon des critères objectifs acceptés par les deux parties. Un exemple d'accord de service peut être le temps de rétablissement d'un service en cas d'incident, le fournisseur et le consommateur définissent un délai pour le rétablissement du service. Si le délai est dépassé, le fournisseur doit indemniser le consommateur selon les termes du contrat. La définition de l'accord de niveau de service peut se faire selon plusieurs niveaux, comme défini dans [BJPW99] :

- le **niveau syntaxique** : les différents acteurs se mettent d'accord sur la signature des méthodes proposées par le service, ce qui correspond au nom des méthodes, aux types de paramètres d'entrées et de sorties ainsi qu'aux types d'exceptions qui peuvent être levées. Ces éléments font partie, en général, de l'interface programmatique du service.
- le **niveau comportemental** : c'est une extension du niveau précédent, qui prend également en considération les pré-conditions, les post-conditions et les invariants.
- le **niveau synchronisation** : ce niveau gère le comportement global pour l'enchaînement des appels de méthodes sous forme de synchronisation. Le contrat décrit les dépendances entre les services. Les appels peuvent se faire de manière séquentielle, parallèle ou sans contrainte.
- le **niveau qualité de service** : ce niveau s'appuie sur tous les précédents. Il ajoute des contraintes de qualité aux services et à leurs interactions : des facteurs de qualité qui respectent des critères mesurables.

Ces quatre niveaux d'accord de service permettent donc aux consommateurs et aux fournisseurs de s'entendre sur la qualité attendue du service, tout comme sur ses fonctionnalités présentées dans une interface.

1.2 SOC : *Service-Oriented Computing*

Les services sont l'élément-clé de l'approche orientée service, en anglais *Service-Oriented Computing* (SOC). Son but est de permettre la construction d'applications à partir d'entités logicielles particulières, que sont les services, tout en assurant un faible couplage. Cette approche n'est pas une technologie ; elle peut être vue comme un style architectural [SG96].

Ce style architectural repose sur un patron qui définit un ensemble d'interactions entre différents acteurs. Ce patron est présenté dans la Figure 2.1. Le modèle d'interactions et les acteurs découlent de la définition des services. Les acteurs sont au nombre de trois :

- le **fournisseur de service** qui propose un service décrit dans une spécification ;
- le **consommateur de service** qui utilise des services des fournisseurs ;
- le **registre de services ou annuaire** qui stocke l'ensemble des descriptions de services déclarées par le fournisseur de service. Il permet aussi aux consommateurs de service de rechercher et de sélectionner le service qui leur sera utile.

Aux trois acteurs de l'architecture orientée service, il a été ajouté trois primitives de communication :

1. la **publication de services** : les fournisseurs de service enregistrent leur service dans l'annuaire ;
2. la **découverte d'un service** : les consommateurs de service interrogent l'annuaire pour trouver un service qui correspond à leurs besoins ;

3. la **liaison et l'invocation d'un service** : un fois le service choisi, le consommateur de service peut se lier au fournisseur et utiliser le service.

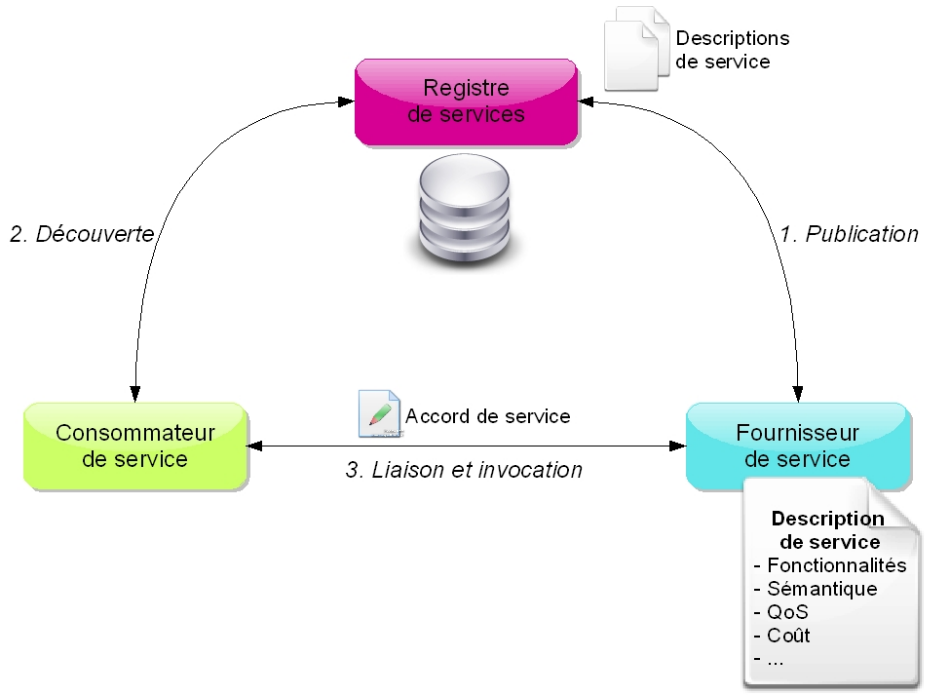


FIGURE 2.1 – Acteurs et interactions dans l'architecture à services.

L'avantage certain de cette architecture est que seule la description de service est partagée entre les différents acteurs ; ceci permet d'obtenir un très faible couplage. La description de service peut prendre différentes formes et fournir différents degrés de précision selon les approches, mais son but principal est de spécifier les fonctionnalités offertes par le service. Comme cette architecture procure un faible couplage, il apparaît un autre avantage : l'hétérogénéité des implantations et des plates-formes est masquée au consommateur de service, tout comme la localisation du service.

En conséquence du faible couplage obtenu grâce à cette architecture, nous obtenons une nouvelle propriété : la substituabilité. En effet, il est possible de remplacer un service par un autre de façon transparente grâce à l'interface du service dès lors qu'il respecte le contrat que le fournisseur et le client ont passé.

Finalement, cette architecture favorise la communication entre un client et un fournisseur de services appartenant à des domaines différents d'administration. Ceci est un des éléments importants de l'approche à services, même si, dans la plupart des cas, les services sont utilisés au sein d'une même entreprise.

L'intérêt grandissant pour les services Web a conduit à une confusion entre les termes de services et architecture à services avec la notion de services Web. Les services Web ne sont qu'une implantation particulière des principes de l'approche à services. Il existe de

nombreuses autres implantations telles que Jini [Sund], UPnP [2] [UPn08] utilisé dans le contexte des services répartis, OSGi™ [3] [OSG07] pour des services localisés sur une même machine virtuelle. Toutes ces technologies seront détaillées dans les sections 4 et 5 de ce chapitre.

1.3 SOA : *Service-Oriented Architecture*

Pour réaliser le style architectural présenté précédemment, il faut mettre en place un environnement d'intégration et d'exécution des services. Cet environnement doit être capable de gérer les interactions entre les différents acteurs. Nous pouvons diviser en deux catégories les éléments d'un tel environnement, illustrés par la Figure 2.2 :

- les **mécanismes de base** qui assurent la publication, la découverte, la composition, la négociation, la contractualisation et l'invocation des services ;
- les **mécanismes additionnels** qui prennent en charge les besoins non-fonctionnels tels que la sécurité, les transactions ou encore la qualité de service.

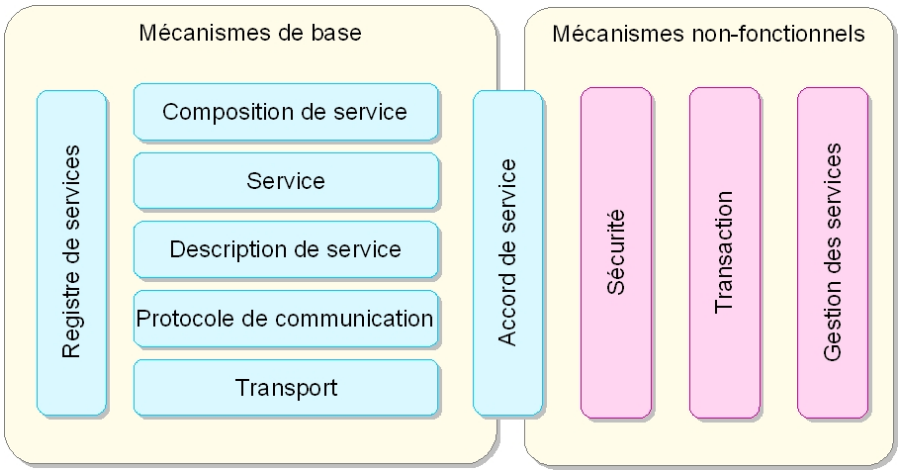


FIGURE 2.2 – Mécanismes nécessaires pour un environnement d'intégration de services.

Les mécanismes de *Transport* et le *Protocole de communication* sont la base d'un environnement d'intégration de services. Ils permettent d'assurer les communications, c'est-à-dire les requêtes et les réponses, entre les différents acteurs. Pour réaliser les interactions de base, il faut aussi pouvoir décrire le service dans un langage de description spécifique à l'environnement d'intégration. Cette description doit comprendre les fonctionnalités du service, ses éléments non-fonctionnels ainsi que la manière dont il doit être invoqué. De manière transverse, il est nécessaire d'ajouter un *Registre de services* qui permet pour le

2. *Universal Plug and Play*
3. *Open Services Gateway Initiative*

fournisseur d'enregistrer son service, pour le client de rechercher un service qui répond à ses besoins.

L'*accord de service* est un élément qui fait partie des mécanismes de base mais aussi de la partie non-fonctionnelle. Il représente le contrat qui existe entre le fournisseur de service et le client. Dans ce contrat sont définies les fonctionnalités que le service doit rendre. Mais, il y est aussi spécifié les propriétés non-fonctionnelles, comme le temps de réponse ou la fiabilité que le service s'engage à respecter.

Les autres éléments non-fonctionnels nécessaires à un environnement d'intégration de services sont :

- la *Sécurité* qui permet par exemple au fournisseur de service de gérer l'accès à son service.
- la *Transaction* qui est utile dans le cas où divers services sont utilisés en collaboration. Dans ce cas, les transactions permettent de s'assurer de la cohérence des données.
- la *Gestion de services* qui assure l'administration des ressources et de leur utilisation au sein de la plate-forme pour un bon fonctionnement des applications.

1.4 Besoins

Nous avons présenté jusqu'ici l'approche à services comme un nouveau paradigme qui a pour but de faciliter la réalisation d'applications à partir des services. Cependant, dans la section précédente, nous n'avons pas introduit la notion d'application à services ; nous avons seulement défini les mécanismes nécessaires pour construire un environnement d'intégration de services. Ces mécanismes sont nécessaires à la réalisation d'application à services mais ils ne sont pas suffisants pour répondre à tous les besoins d'une application. M.P. Papazoglou [Pap03] a donc proposé une architecture orientée service étendue pour exprimer la composition de services et l'administration d'application, tout en s'appuyant sur les mécanismes de base de l'approche à services. La Figure 2.3 est la pyramide définie par M.P. Papazoglou sur les fonctionnalités attendues de l'architecture à services.

La base de la pyramide contient les fonctionnalités que requiert un intergiciel proposant une approche à services. Ces fonctionnalités sont celles présentées précédemment : la publication, la découverte, la liaison, la sélection et la spécification.

Le deuxième étage de la pyramide introduit les mécanismes de composition de services. Ces mécanismes permettent de coordonner les services, de gérer les compositions et de s'assurer de la conformité de la composition. Le service composite créé doit respecter les principes de base de l'approche à services. La composition de services sera présentée dans la section 2 de ce chapitre.

Le sommet de la pyramide concerne l'administration des services et leurs compositions, c'est-à-dire les applications à services. Ceci correspond à la gestion et à la surveillance d'applications.

De manière transversale à ces niveaux de la pyramide, une architecture orientée service étendue doit prendre en considération la gestion des propriétés non-fonctionnelles, comme la sécurité ou encore la qualité de service.

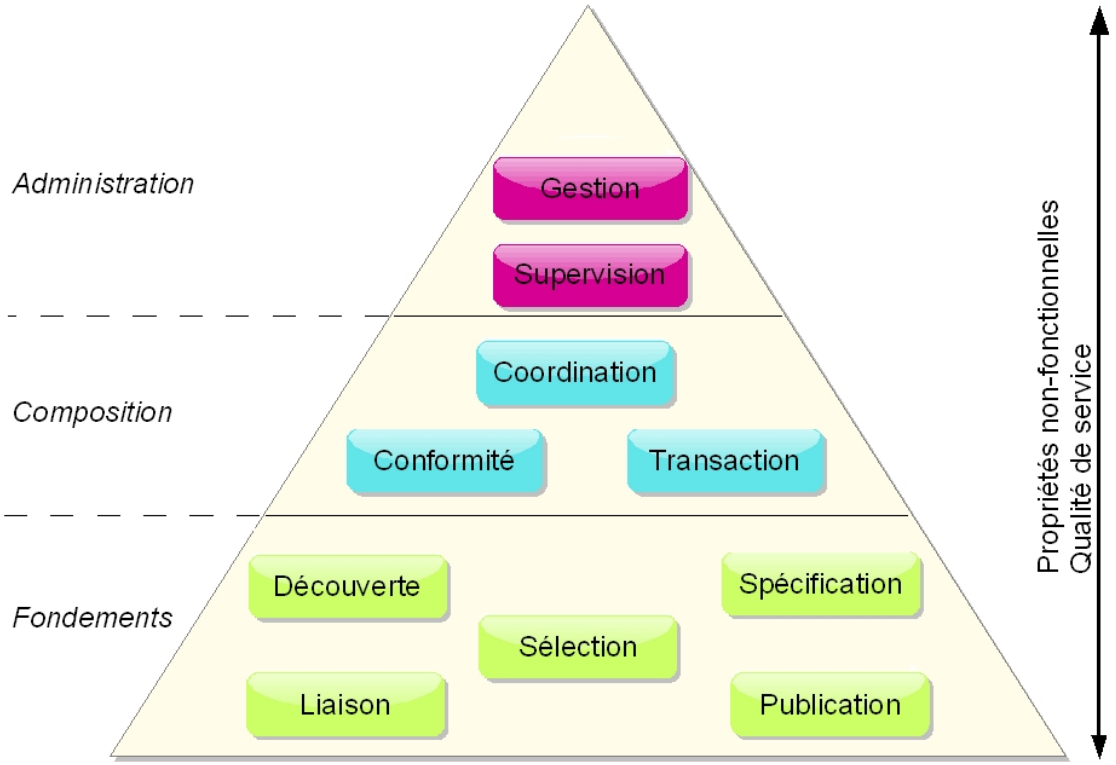


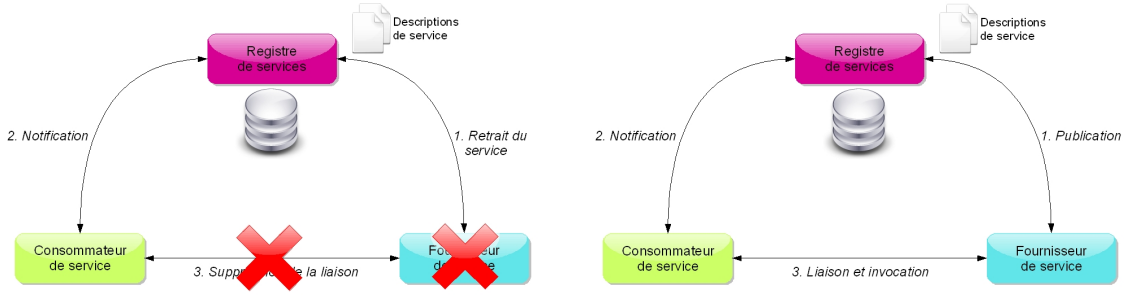
FIGURE 2.3 – Fonctionnalités d'une architecture orientée service étendue.

Cependant, cette pyramide de l'architecture orientée service est difficile à mettre en œuvre. La plupart des intergiciels ne proposent pas l'ensemble des fonctionnalités présentées. Ils ne fournissent, en général, que les principes de base. Mais, il est très important de mettre en place la composition et l'administration ; car sans ces niveaux, l'utilisateur n'a pas de vision complète de l'application.

1.5 SOC et le dynamisme

Nous avons présenté la vision communément admise de l'approche à services dans les sections précédentes en présentant les acteurs et les interactions. Cependant, certaines définitions [Esc08] introduisent des interactions supplémentaires pour ce que l'on appelle l'approche à services dynamique. Cette approche s'intéresse aux modifications de l'environnement d'exécution des services. Il a donc été ajouté deux primitives à l'approche à services qui sont les suivantes :

- le **retrait de service** qui signale qu'un fournisseur de service n'est plus en mesure de proposer son service ;
- la **notification** qui informe les consommateurs de l'arrivée ou du départ d'un fournisseur qui propose un service répondant à leurs besoins.



(a) Retrait d'un fournisseur de service.

(b) Ajout d'un fournisseur de service.

FIGURE 2.4 – Interactions de l'approche à services dynamique.

Ces deux nouvelles primitives, si elles sont prises en compte par l'architecture à services, permettent au consommateur de choisir lors de l'exécution son fournisseur de service, voire d'en changer. La prise en compte de l'évolution dynamique de l'environnement est un nouvel avantage que l'on ajoute à ceux de l'approche à services. Cette caractéristique est appelée liaison à retardement, en anglais *late-binding*.

La pyramide de l'architecture orientée service étendue peut, elle-même, être étendue au dynamisme. Il faut ajouter à la base les deux nouvelles primitives : retrait et notification. Ensuite, l'ajout de fonctionnalités dans les niveaux supérieurs s'impose pour prendre en compte le dynamisme de la couche basse. La composition de services dans l'architecture dynamique doit permettre de s'assurer à tout moment de la conformité de cette composition. Alors, la partie administration doit être capable de gérer la supervision et la gestion de l'application en fonction des arrivées et des départs des services.

1.6 Caractérisation

Dans les sections précédentes, nous avons défini ce que sont les services. Nous avons montré quelle architecture est proposée pour mettre en application les concepts de l'approche à services et quels sont les besoins qui découlent de cette approche. Par la suite, nous allons étudier plusieurs technologies, dites à services, que nous allons caractériser selon les critères suivants :

- **la spécification :** comment sont décrites les fonctionnalités et les propriétés non-fonctionnelles des services ?
- **l'implantation :** dans quel langage de programmation sont réalisés les services ?
- **la découverte :** quel est le registre de services ? quel type de découverte est possible ?
- **la composition :** est-ce que la composition est possible ? Si oui, comment ?
- **la communication :** quel est le protocole de communication entre les différents acteurs ?
- **la liaison :** quelle est la politique de liaison mise en place pour faire face au dynamisme ?

Dans les parties suivantes, nous allons répondre à ces questions pour chacune des technologies à services étudiées, comme par exemple pour les services Web un standard très répandu, OSGi™ et iPOJO des technologies à composants orientés service, UPnP et DPWS des technologies à services pour la domotique.

2 Composition de services

2.1 Définitions

Comme nous l'avons vu précédemment, la mise en œuvre de l'approche à services ouvre des perspectives pour la composition de services dans le but de construire des applications. La composition de services peut être vue comme un mécanisme qui permet l'intégration des services pour réaliser une application. Le résultat d'une composition peut être un nouveau service, appelé service composite. Ce type de composition est dite récursive ou hiérarchique. La Figure 2.5 présente le principe de la composition de services ; à partir d'un ensemble de services disponibles dans un registre, nous pouvons construire une application à services.

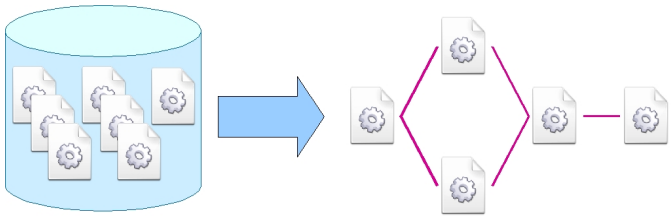


FIGURE 2.5 – Composition de services.

Cependant, pour passer d'un ensemble de services à une composition de services correctement structurée, il faut suivre un certain nombre d'étapes de la spécification à la composition concrète exécutable :

1. **la définition de l'architecture fonctionnelle :** cette phase est faite pour identifier les fonctionnalités attendues pour l'application résultant de la composition de services. Des travaux d'aide au niveau de cette phase commencent à exister [Ser].
2. **l'identification des services :** selon les fonctionnalités attendues, on détermine les services nécessaires à la composition ;
3. **la sélection des services et leur implantation :** à partir des services identifiés à l'étape précédente, il faut sélectionner les services qui répondent correctement aux besoins ainsi que les implantations adaptées ;
4. **la médiation entre services :** même si à l'étape précédente, les services les plus adaptés ont été sélectionnés, en général, il n'est pas possible de les assembler tels quels. Il faut souvent ajouter de la médiation, par exemple sémantique, pour que les interactions entre services fonctionnent comme prévu.
5. **le déploiement et l'invocation des services :** une fois la composition correctement réalisée, il faut déployer les services sur les plates-formes d'exécution. Il est ainsi possible d'invoquer les services pour obtenir la composition concrète.

Malgré cette décomposition en tâches pour réaliser une application à base de composants, ce processus reste une activité difficile qui peut même être longue. Chaque tâche peut être divisée en sous-tâches que les développeurs doivent parfois exécuter manuellement. La situation idéale qui consiste à avoir uniquement les « bons » services, c'est-à-dire des services avec les fonctionnalités attendues et totalement compatible entre eux, ne se présente que rarement aux développeurs. Ces derniers doivent gérer les problèmes d'incompatibilités par une couche de médiation qui permet d'adapter les services les uns aux autres. Les problèmes d'incompatibilités sont, par exemple, des problèmes de types de données pour les entrées et les sorties des services.

La complexité de la composition est double : d'une part, la complexité de la logique métier inhérente aux applications qui nécessite une expertise métier ; d'autre part, la complexité de la mise en œuvre de l'approche à services.

La composition de services est spécifiée selon une logique de coordination des services, c'est-à-dire selon le contrôle de la composition qui peut être extrinsèque ou intrinsèque aux services. Ces deux possibilités de gestion du contrôle définissent deux styles de composition : la composition par procédés, aussi appelée composition comportementale, et la composition structurelle ; elles seront présentées dans les deux sections suivantes.

2.2 Composition par procédés

Dans ce type de composition, la logique de coordination des services est spécifiée par un procédé. Un procédé est représenté par un graphe orienté d'activités et un flot de contrôle qui donne l'ordre d'exécution des activités. Chaque activité représente une fonctionnalité et cette dernière est réalisée concrètement par un service.

En pratique, la composition est décrite dans un langage spécifique qui est interprété par un moteur d'exécution. Toutes les communications avec les services sont gérées par le moteur, tout comme les erreurs.

Nous distinguons deux sous-types de composition en fonction du type de contrôle :

– **la chorégraphie de services** qui décrit la collaboration entre une collection de services dont le but est d'atteindre un objectif donné. L'accomplissement de ce but commun se fait alors par des échanges ordonnés de messages [ABPRT04]. Dans ce cas, le contrôle est distribué, comme le montre la Figure 2.6.

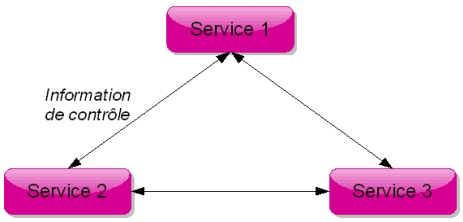


FIGURE 2.6 – Chorégraphie de services.

– **l'orchestration de services** qui décrit, du point de vue d'un service, les interactions de celui-ci ainsi que les étapes internes (ex. transformations de données, invocations à des modules internes) entre ses interactions [Pel03]. C'est une vision centralisée du contrôle, comme illustrée dans la Figure 2.7.

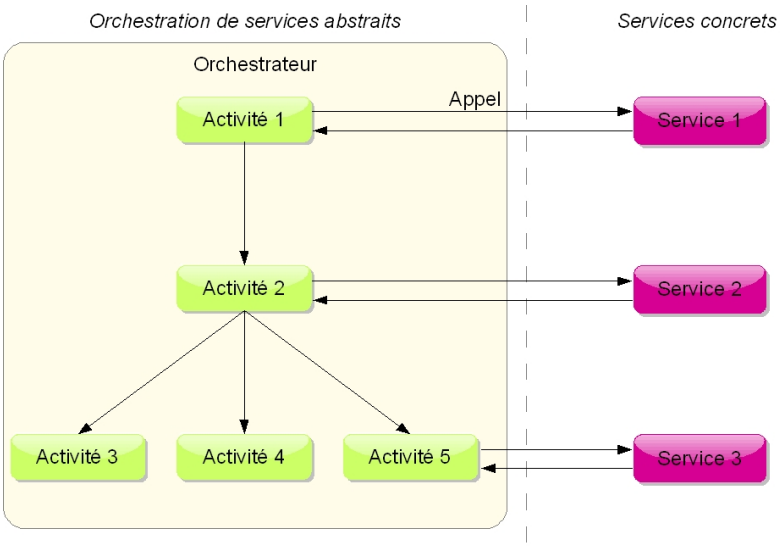


FIGURE 2.7 – Orchestration de services.

Ces deux points de vue de la composition par procédés ne sont utilisés actuellement que pour la technologie des services Web. WS-BPEL[4] [OAS07a], qui sera présenté par la suite dans la section 3.5 page 47, est un exemple de langage d'orchestration de services Web ; WS-CDL[5] [W3C04c] est, quant à lui, un langage de description de chorégraphie de services Web.

La composition par procédés permet de séparer distinctement le contrôle d'une application des fonctionnalités. Elle est utilisée par les développeurs pour construire des applications à base de briques logicielles dont le fonctionnement interne n'est pas connu. Ces briques logicielles peuvent être vues comme des « boîtes noires » . Les fonctionnalités ainsi identifiées peuvent être plus facilement réutilisées pour d'autres compositions et l'expression de la logique de contrôle est exprimée simplement. Cependant, il existe peu d'interactions entre les activités qui sont assemblées, puisque les langages de composition ne permettent pas de réaliser des algorithmes complexes. De plus, il n'est pas possible de détailler, par exemple, les types d'interactions qui existent entre les activités. La composition par procédés propose un mode de contrôle très avantageux mais qui reste limité à certains domaines.

4. *Web Service Business Process Execution Language*
5. *Web Service Choregraphy Description Language*

2.3 Composition structurelle

Par opposition à la composition par procédés, le contrôle dans une composition structurelle est exprimé à l'intérieur des services. Le contrôle n'est alors connu que du développeur et les seules informations qu'il possède sont celles concernant les fonctionnalités que le service fournit et celles que le service requiert.

Dans le cas de la composition structurelle, les services sont donc clairement identifiés avec leurs interactions. Il faut à l'assemblage résoudre les dépendances syntaxiques et sémantiques entre les composants pour s'assurer de la validité de la composition. C'est pourquoi pour définir le fonctionnement de la composition, le développeur livre aussi la logique de coordination, qui peut être, par exemple, sous forme de classe Java. Aujourd'hui, la spécification SCA [6] [OSO07], présentée dans la section 5.3 page 67, est l'une des rares propositions pour la spécification de compositions structurelles avec les travaux de C. Marin [Mar08].

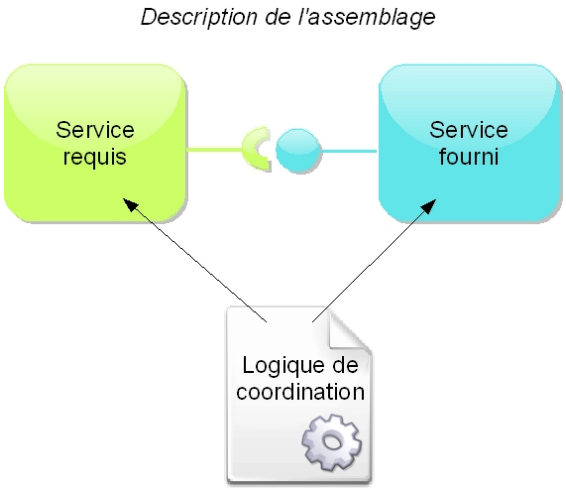


FIGURE 2.8 – Composition structurelle.

A l'inverse de la composition par procédés, la composition structurelle ne permet pas facilement la réutilisation des composants puisque le contrôle est interne aux services. Par contre, elle est plus efficace puisque la communication entre services est directe, elle ne passe pas par un intermédiaire comme dans la composition par procédés. De plus, le contenu du service est réalisé par le développeur, les algorithmes et les interactions entre services peuvent être plus complexes que ceux des compositions par procédés, qui ont un langage plus restreint.

6. *Service Component Architecture*

3 Services Web

Les services Web sont certainement la technologie la plus connue et la plus populaire dans le monde industriel et académique pour la mise en place d'architectures à services. Pour commencer, nous présenterons les principes des services Web [7]. Puis, nous détaillerons les trois standards utilisés par cette technologie, que sont WSDL, UDDI et SOAP. Enfin, nous traiterons de la composition de services Web avec le standard WS-BPEL.

3.1 Principes

Les services Web ont été créés pour rendre disponibles, sous un format standard, des applications sur l'Internet ou dans un Intranet. Ces services respectent les principes de l'approche orientée service précédemment présentés ; ils sont donc décrits, publiés et découverts. Un fournisseur de service Web enregistre son service, en décrivant ses fonctionnalités et certains de ses aspects non-fonctionnels dans un fichier WSDL, auprès d'un annuaire UDDI. Un client interroge un annuaire UDDI pour trouver un service qui répond à ses besoins. Pour le consommateur, le service Web est une boîte noire qui ne donne pas de détails techniques sur son implantation, seulement des informations sur ses fonctionnalités et quelques propriétés, sa localisation et les moyens pour l'interroger. Les communications se font par le protocole SOAP. L'architecture des services Web est illustrée dans la Figure 2.9.

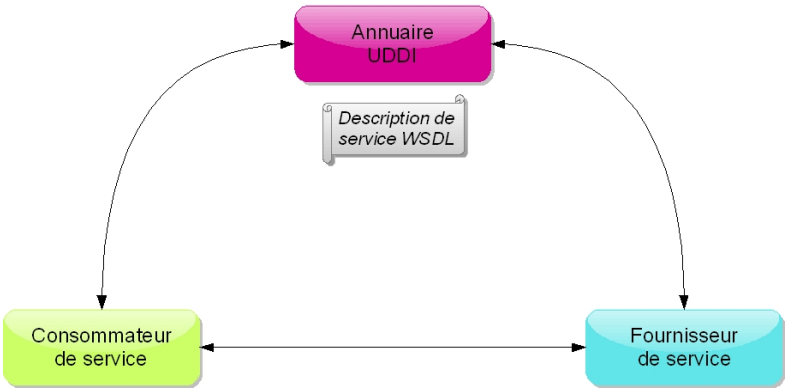


FIGURE 2.9 – Architecture pour les services Web.

Cette architecture cache à l'utilisateur toute la complexité de l'implantation du service Web. En effet, un service Web peut faire appel à d'autres services, Web ou non, pour réaliser les fonctionnalités qu'il propose sans que le client en soit conscient. Cela nécessite une coordination entre les différents appels de services. Le détail de chacune

7. Nous étudions ici les services Web standard et non les services Web REST [FT00].

des technologies présentées dans cette architecture est donné dans les trois sections suivantes.

3.2 WSDL : le langage de description des services Web

Le succès des services Web repose en partie sur le faible couplage qui existe entre les consommateurs de services et le fournisseur de service. La description dans un langage standard des différentes fonctionnalités du service par le fournisseur de service permet aux consommateurs de s'abstraire des langages de programmation utilisés pour réaliser les services Web. Seules les fonctionnalités du service sont présentées dans le fichier de description ; ainsi les détails techniques propres au fournisseur de service ne sont pas dévoilés aux consommateurs.

La description d'un service Web est faite dans le langage WSDL[8] [W3C01b]. Un fichier WSDL comprend une description des fonctionnalités d'un service, mais il ne se préoccupe pas de l'implantation de celles-ci. Il contient aussi des informations concernant la localisation du service, ainsi que les données et les protocoles à utiliser pour l'appeler. En pratique, le fichier WSDL est un fichier XML qui se divise en deux parties :

- la **définition abstraite** de l'interface du service avec les opérations supportées par le service Web, ainsi que leurs paramètres et les types des données ;
- la **définition concrète** de l'accès au service avec la localisation, par une adresse réseau du fournisseur de service, et les protocoles spécifiques d'accès.

La Figure 2.10 correspond à la structure d'un fichier WSDL. La partie abstraite est définie par les balises `types`, `messages` et `port types`. Ainsi sont décrits les types de données utilisés, les messages échangés et les opérations possibles pour un service Web. La partie concrète, qui est spécifique à l'implantation, comprend les balises `binding` et `service`. L'élément `binding` spécifie le protocole avec lequel les clients peuvent invoquer le service. L'élément `service` permet d'associer au service Web une adresse réseau.

Le fichier WSDL détaille les fonctionnalités d'un service Web. Mais pour l'instant, il ne tient pas compte des propriétés non-fonctionnelles, telles que la sécurité, les transactions par exemple, qui peuvent lui être associées. De nouvelles extensions pour les fichiers WSDL sont proposées pour traiter ces aspects non-fonctionnels comme les spécifications WS-Security [OAS04b] pour la sécurité, WS-Transaction [IBM04] pour les transactions... Cependant, ces travaux de spécifications avancent lentement et évoluent régulièrement.

La description détaillée de l'interface d'un service Web avec un fichier WSDL n'est pas suffisante pour son utilisation par un client. Avant d'utiliser un service Web, le client doit d'abord le rechercher et s'assurer qu'il corresponde à ses besoins. L'architecture des services Web propose l'utilisation d'un annuaire de services UDDI[9] présenté dans la section suivante.

8. *Web Services Description Language*
9. *Universal Description Discovery and Integration*

(a) Structure. (b) Exemple.

FIGURE 2.10 – Le fichier WSDL.

3.3 UDDI : l'annuaire de services Web

Un des principaux avantages de l'adoption des services Web par les entreprises est le partage de services sur Internet ou dans un Intranet. Le partage de services Web permet le développement plus rapide d'applications, soit un gain de temps pour l'entreprise ainsi qu'un gain d'argent. L'élément-clé pour ce partage des services est l'annuaire UDDI [OAS04a], qui permet de référencer et de classifier leurs fonctionnalités.

UDDI est une spécification d'annuaire qui propose d'enregistrer et de rechercher des fichiers de description de services Web correspondant aux attentes d'un client. UDDI a été initialement conçu par et pour des industriels, en ayant pour but d'avoir un standard indépendant des plates-formes d'implantation, afin de

- **connaître** les entreprises qui fournissent des services Web ;
- **découvrir** les services Web disponibles qui répondent aux attentes du client.

Pour simplifier les recherches de services, le standard UDDI propose trois types d'an-

nuaires qui ont des critères particuliers :

- les **pages blanches** permettent de connaître les informations concernant les entreprises ;
- les **pages jaunes** présentent les services selon leurs fonctionnalités en utilisant une taxonomie industrielle standard ;
- les **pages vertes**, quant à elles, informent sur les services fournis par les entreprises. Elles référencent la localisation des fichiers de descriptions WSDL des services Web.

Les annuaires UDDI ne sont pratiquement pas utilisés aujourd'hui. Il apparaît que les entreprises mettent en place des architectures à base de services Web en développant leur propre annuaire, par exemple la poste Allemande, *Deutsche Post*. Cette solution, coûteuse mais compréhensible, s'effacera quand des technologies d'annuaire plus complètes et plus robustes s'affirmeront.

WSDL et UDDI sont deux standards pour les services Web. Le premier permet d'afficher les fonctionnalités et les moyens d'utilisation du service d'un fournisseur. Ce dernier peut aussi enregistrer son service dans un annuaire UDDI pour qu'il soit référencé et utilisé par des consommateurs de service. Il existe un autre point important dans l'architecture des services Web ; ce sont les communications entre les différentes parties qui se font avec le protocole SOAP, présenté dans la section suivante.

3.4 SOAP : les communications entre services Web

SOAP [W3C00], à l'origine acronyme de *Simple Object Access Protocol*, est un protocole dont la syntaxe est basée sur XML. Son but est de permettre des échanges standardisés de données dans des environnements distribués. Il permet d'accéder à des services distants indépendamment de la plate-forme d'exécution. Initialement proposé par Microsoft et IBM, la spécification SOAP est aujourd'hui une recommandation W3C. Il faut noter que depuis la version 1.2 SOAP n'est plus un acronyme, la notion d'objet étant devenue obsolète.

Un message SOAP contient une enveloppe obligatoire, un en-tête facultatif et un corps obligatoire qui se présente sous la forme d'un document XML :

- l'élément **en-tête** contient des éléments-fils, appelés *entrées*, permettant d'ajouter des extensions à un message. Ces extensions permettent, par exemple, de prendre en charge les transactions et la sécurité. SOAP se voulant un protocole léger et simple a volontairement ignoré la sécurité.
- l'élément **corps** du message contient la méthode à invoquer ainsi qu'éventuellement ses paramètres d'appel.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <symbol xmlns="http://stock.samples">XXX</symbol>
  </soapenv:Body>
</soapenv:Envelope>
```

FIGURE 2.11 – Exemple de message SOAP pour interroger un service Web.

Le protocole SOAP s'appuie sur des standards de communication comme le protocole HTTP [10], mais il peut aussi utiliser des protocoles autres comme SMTP [11]. L'avantage d'utiliser SOAP avec le protocole HTTP est que la communication est facilitée, en particulier les *proxys* et les pare-feu peuvent être franchis sans problème. Il est ainsi facilement adaptable à toutes les technologies antérieures, tout en restant simple et extensible. Le protocole SOAP a pour avantage d'être indépendant de la plate-forme d'exécution et du langage de programmation.

SOAP est un protocole limité par sa simplicité et ses faibles performances. Des alternatives apparaissent aujourd'hui, notamment pour apporter plus d'efficacité.

3.5 WS-BPEL

L'assemblage de services Web repose sur l'orchestration, il n'existe pas de composition structurelle de services Web. WS-BPEL [OAS07a], acronyme de *Web Services Business Process Execution Language*, est une spécification du consortium OASIS. Elle en est à sa version 2.0 depuis mars 2007. Cette spécification est l'une des plus connues pour l'orchestration de services Web. Elle remplace les précédentes spécifications XLANG [12] de Microsoft, et WSFL [13] d'IBM.

WS-BPEL est un langage de procédés basé sur la technologie XML, tout comme les autres standards des services Web. WS-BPEL permet de construire des procédés interprétables et exécutables par un moteur d'orchestration. Les procédés peuvent être modélisés de deux manières :

- **abstraite** : seuls les échanges de messages entre les différents participants sont spécifiés. Mais le comportement interne de ces participants n'est pas explicité.
- **exécutable** : les activités du procédé sont ordonnées ; les partenaires impliqués sont identifiés ainsi que les messages qui sont échangés. A ceci s'ajoute le traitement des fautes et des exceptions pour les cas d'erreurs.

10. *HyperText Transfer Protocol*
11. *Simple Mail Transfer Protocol*
12. `http://www.ebpml.org/xlang.htm`
13. *Web Services Flow Language*, `http://xml.coverpages.org/wsfl.html`

Un procédé est composé d'activités qui s'enchaînent grâce à des échanges de données. Les activités peuvent être de deux types : basiques ou complexes. Les activités basiques sont des types de base comme `invoke` pour appeler un service Web, `receive` pour attendre une invocation, `reply` pour une réponse... A partir de ces types de base, il est possible de créer des activités composites avec des structures de construction du contrôle du flot de données : `flow` pour une ou plusieurs activités concurrentes, `sequence` pour une séquence d'activités, `switch` pour des conditions, `while` pour une boucle... Il faut quand même noter que l'échange de données n'existe pas en tant que tel, puisqu'il faut passer par des affectations de variables entre les activités. En général, ces outils sont munis d'interfaces graphiques qui simplifient la réalisation des compositions de services. La Figure 2.12 est une capture d'écran d'un procédé WS-BPEL réalisé avec BPEL Project, qui est un *plug-in* Eclipse [14].

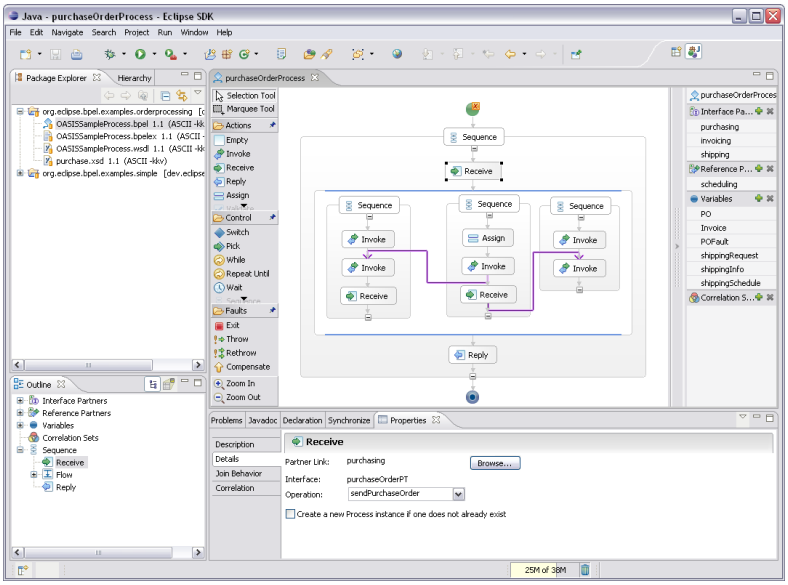


FIGURE 2.12 – Exemple d'un procédé WS-BPEL.

L'exécution des procédés se fait avec un moteur d'exécution spécifique. Il existe actuellement sur le marché de nombreux moteurs d'exécution de procédés comme Orchestra [15] de Bull, ActiveBPEL [16] qui sont *open source* et en Java, Oracle BPEL Process Manager [17] ou encore Netbeans BPEL Service Engine [18]. Le moteur d'exécution est centralisé et permet d'exécuter la composition de services définie sous forme d'orchestration. Il permet de réaliser les communications avec les services concrets et l'invocation des fonctionnalités de ces services.

14. `http://www.eclipse.org/bpel/`
15. `http://orchestra.objectweb.org/xwiki/bin/view/Main/WebHome`
16. `http://www.activevos.com/community-open-source.php`
17. `http://www.oracle.com/technology/products/ias/bpel/index.html`
18. `https://open-esb.dev.java.net/kb/preview3/ep-bpel-se.html`

WS-BPEL est aujourd'hui très limité et de nombreux travaux sont en cours pour l'étendre. Deux cas privilégiés sont l'utilisation de spécification abstraite de service, comme dans SCA, et l'ajout de propriétés non-fonctionnelles, comme la sécurité.

3.6 Synthèse

La technologie des services Web est l'une des plus connues et répandues pour appliquer les principes de l'approche orientée service. Elle est basée sur le standard XML et permet de rendre disponibles dans un annuaire UDDI des services dont les fonctionnalités sont décrites dans des fichiers WSDL. Le protocole SOAP permet aux consommateurs d'interroger l'annuaire et d'invoquer le service facilement à travers l'Internet ou un Intranet. Cependant, ces technologies ne tiennent pas compte d'un certain nombre de caractéristiques importantes, comme les propriétés non-fonctionnelles, même si de nombreuses spécifications ont été proposées. Elles sont d'ailleurs tellement nombreuses qu'il est difficile de toutes les maîtriser. On peut penser qu'une stabilisation interviendra, mais ce n'est pas encore le cas aujourd'hui.

Le Tableau 2.1 récapitule les informations concernant la technologie des services Web en fonction des caractéristiques importantes de l'approche à services.

	Services Web
Spécification	– Interface WSDL – Des propriétés non-fonctionnelles
Implantation	– Nombreux langages – Génération des talons clients
Découverte	– UDDI – Découverte active
Composition	– Oui, orchestration WS-BPEL
Communication	– SOAP sur HTTP – Synchrone
Liaison	– Type : direct – Statique

TABLE 2.1 – Récapitulatif de la technologie des services Web.

Avec la spécification WS-BPEL, les services Web font une avancée en ce qui concerne la composition des services. Cette composition est une orchestration de services qui a les avantages des compositions par procédés mais aussi les inconvénients comme le problème du moteur d'exécution centralisé. Cependant, les services Web ne supportent pas les mécanismes du dynamisme de l'architecture à services.

4 Composants orientés service

4.1 Principes

L'approche à composants est une approche antérieure à l'approche à services. Elle préconise l'assemblage de composants pour réaliser des applications [HC01]. Un composant peut être vu comme une brique logicielle qui est faite pour être assemblée avec d'autres composants [WD01]. Les composants tentent de répondre à deux objectifs :

- le *megaprogramming*, c'est-à-dire la programmation à partir de modules, logiciels gros grains pour améliorer la productivité et faire face à la taille toujours croissante des logiciels. Il s'agit également de favoriser la réutilisation.
- l'amélioration de la modularité des applications qui n'est pas fournie de façon suffisante avec des approches à objets par exemple.

Les composants ont été conçus principalement pour être réutilisés dans différentes applications, sans que l'implantation du composant soit connue. Il suffit d'avoir une connaissance des fonctionnalités du composant et des moyens de l'assembler. Dans la suite, nous détaillons les caractéristiques d'un composant ainsi que les bénéfices qui ont été tirés de l'approche à composants et de l'approches à services pour ce que l'on appelle l'approche à composants orientés service.

4.1.1 Les composants

Il n'existe pas de définition consensuelle de la notion de composant. Cependant, l'une des plus citées et connues est celle de C. Szyperski :

> « *A software component is a unit of composition which contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.* » [Szy02]

Cette définition met en évidence les concepts importants qui caractérisent un composant :

- **la spécification d'interfaces :** un composant expose les fonctionnalités qu'il fournit ;
- **les dépendances explicites :** un composant présente les besoins qu'il requiert pour réaliser ses fonctionnalités. Il existe en fait deux types de dépendances :
  - la dépendance « logique » sur des fonctionnalités d'autres composants ;
  - la dépendance « physique » sur du code, par exemple des bibliothèques.
- **l'instanciation :** un composant fournit un type et peut être instancié plusieurs fois ;

– **l'indépendance de déploiement :** un composant est une unité de déploiement à part entière. Toutefois cette caractéristique est controversée.

Les composants ayant été créés pour être assemblés, il faut ajouter à la notion de composant un mécanisme de composition illustré dans la Figure 2.13. En fait, chaque approche à composants définit un langage de description d'architecture d'application à partir de composants, appelé ADL [19]. Les ADL, qui n'ont pas été faits à l'origine pour les composants, sont basés sur deux éléments :

– **les instances :** les instances de composants forment la logique métier de l'application, ce sont les éléments requis à la réalisation des fonctionnalités de l'application ;
– **les connecteurs :** ce sont les liaisons qui sont réalisées entre les différentes instances de composants de l'application. Ces liaisons sont déterminées en fonction des propriétés fournies et requises par les diffèrents types de composants utilisés.

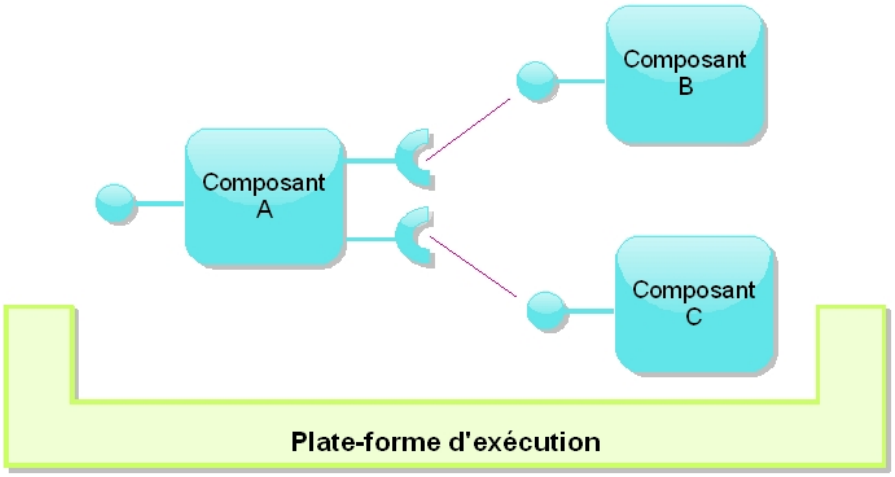


FIGURE 2.13 – Exemple d'assemblage de composants.

A cette approche, qui définit les caractéristiques de composants et leur composition, est associé un environnement d'exécution qui prend en charge la réalisation de l'application résultante. L'environnement d'exécution est parfois comparé à un mini-système d'exploitation [BBB+00] car il est chargé de gérer des aspects divers tels que le cycle de vie des instances ou bien les propriétés non-fonctionnelles.

D'après les principes de l'approche à composants qui ont été cités précédemment, nous pouvons noter que son point fort est la gestion complète du cycle de vie des composants ; en particulier, le déploiement et l'instanciation ne sont que peu traités par l'approche à services. Par contre, dans cette approche, la méthode d'implantation du composant n'est pas du tout abordée.

19. *Architecture Definition Language*

Toutefois, il est intéressant de noter que, au sein de la communauté scientifique travaillant sur des approches à composants [20], une approche particulière a été proposée en se basant sur le principe de la séparation des préoccupations. La structure des composants suit ce principe, c'est-à-dire que le code fonctionnel du composant est clairement identifié et séparé du code non-fonctionnel, comme par exemple les transactions, la persistance ou la sécurité. Une des approches les plus utilisées pour appliquer la séparation des préoccupations est l'utilisation des conteneurs. Un conteneur est illustré dans la Figure 2.14.

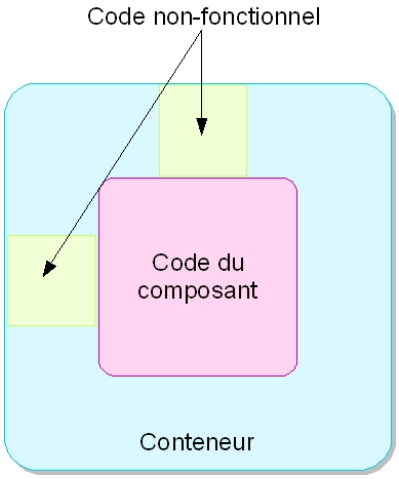


FIGURE 2.14 – Structure d'un composant.

Le code métier du composant est englobé dans un conteneur qui prend en charge certains aspects non-fonctionnels en suivant les spécifications données par l'utilisateur ainsi que la gestion du cycle de vie du composant. Il existe de nombreuses implantations de modèles à composants détaillées et comparées dans [Cer04]. Ces implantations se focalisent en général sur un unique aspect non-fonctionnel, par exemple, la communication par événements des JavaBeans [Sunc], ou encore les transactions et la persistance des données gérée par les EJB [21] [Suna]. Comme un certain nombre d'aspects non-fonctionnels sont récurrents dans les approches à composants, il existe des bibliothèques standard de code non-fonctionnel. Cependant, certaines approches laissent l'utilisateur libre de définir ses propres aspects non-fonctionnels ; ces approches sont dites approches à composants extensibles, Fractal [Obj04] en est un exemple.

20. *Component-Based Software Engineering* (CBSE).
21. *Enterprise JavaBeans*

4.1.2 Approche à composants orientés service

L'approche à composants orientés service, proposée en 2004 dans [CH04], combine les avantages de l'approche à composants et de l'approche à services. Les concepts de l'approche à services enrichissent le modèle à composants. A ceux-ci s'ajoutent des mécanismes permettant aux applications de s'adapter pour supporter la disponibilité dynamique des composants à l'exécution.

Cette approche est basée sur les principes suivants :

- **un service fournit une fonctionnalité.** Un service propose un ensemble d'opérations réutilisables.
- **un service est décrit par une spécification de service** qui contient des informations syntaxiques et, éventuellement, comportementales et sémantiques. Cette spécification est utile pour la composition de services, les interactions et la découverte. Dans la partie syntaxique de cette spécification, les dépendances éventuelles avec d'autres services sont annoncées.
- **un composant implante une spécification de service.** Le composant doit respecter les contraintes qui ont été définies dans la spécification. Si l'implantation impose d'autres contraintes, celles-ci doivent être énoncées.
- **le modèle d'interaction de l'approche à services dynamique est utilisé pour résoudre l'ensemble des dépendances du système.** Les services sont sous forme d'instances de composants qui sont enregistrées dans un annuaire. Cet annuaire est utilisé lors de l'exécution pour découvrir les instances qui résolvent les dépendances des services.
- **les compositions sont décrites sous forme de spécifications de services.** La composition est une spécification de services qui permet de choisir les composants à instancier. La composition devient alors concrète à partir du moment où les composants sont découverts et instanciés. Les liaisons ne sont pas définies de manière explicites, elles sont inférées à partir des dépendances des services.
- **les spécifications de services sont les fondements de la substituabilité.** Dans une composition, un composant peut être remplacé par n'importe quel autre composant pourvu qu'il respecte la même spécification de service.

Il est clair qu'avec cette approche à composants orienté service, on peut bénéficier des avantages de l'approche à composants : un modèle de développement simple et une description de la composition ; ainsi que des avantages de l'approche à services : un faible couplage et le dynamisme. Dans la suite, nous présentons deux technologies OSGi™ et iPOJO qui mettent en œuvre cette approche.

4.2 OSGi™

OSGi™ [OSG07] est une spécification de plate-forme à services qui a été proposée par le consortium OSGi™ Alliance, qui réunit de nombreux acteurs tels que IBM, Motorola, Nokia,... La spécification OSGi™ avait pour but à l'origine de proposer une plate-forme pour les passerelles réseaux et résidentielles. Ces passerelles sont limitées en taille mémoire et en capacité d'exécution. Elles ont en plus des contraintes fortes en ce qui concerne le cycle de vie des composants logiciels ; puisque, dans ces domaines, de nouveaux composants logiciels doivent pouvoir être déployés et assemblés dynamiquement sur la passerelle, sans interrompre l'exécution des autres composants logiciels installés. Aujourd'hui avec l'émergence de nouveaux domaines d'application pour l'informatique embarquée, tels que la téléphonie ou encore l'automobile, les plates-formes OSGi™ deviennent de fait des standards pour le développement d'applications dynamiquement reconfigurables. OSGi™ devient aussi un standard puisqu'il est utilisé pour des serveurs d'applications, tels que OW2 JOnAS 5 [Obj07] ou Sun Glassfish 3 [Sunb] et pour des applications à *plug-ins* telles que l'IDE Eclipse [GHM+05].

Un des points forts de la spécification OSGi™ est de prendre en compte le cycle de vie complet du logiciel, c'est-à-dire de son conditionnement jusqu'à sa désinstallation. Pour cela, OSGi™ propose un système de gestion de déploiement d'applications spécifique. Ce système permet de déployer indépendamment différentes parties d'une application, l'unité de déploiement utilisée est le *bundle*. Ces unités de déploiement sont déployées et administrées à l'exécution. Il est donc possible d'installer, de démarrer, d'arrêter, de mettre à jour ou de supprimer des *bundles* à l'exécution sans interrompre la plate-forme. Le cycle de vie d'un *bundle* est représenté par la Figure 2.15. Une des caractéristiques des *bundles* OSGi™ est qu'ils spécifient les dépendances qu'ils ont avec les autres unités de déploiement au niveau des *paquetages* fournis et requis. Ceci permet de contrôler au démarrage d'une application que toutes les dépendances de code sont résolues.

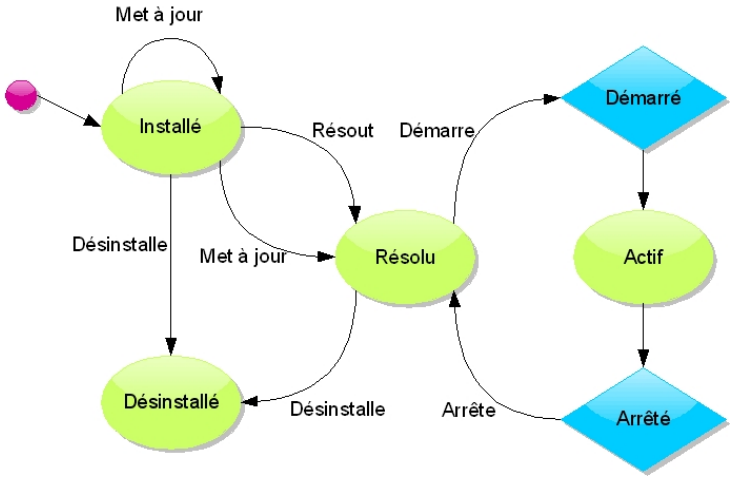


FIGURE 2.15 – Cycle de vie d'un *bundle* OSGi™.

La spécification OSGi™ spécifie une architecture orientée service dynamique. Elle s'appuie sur la couche de déploiement précédemment présentée mais aussi sur le langage Java. OSGi™ est une plate-forme centralisée avec un registre de services. Les services sont décrits à l'aide d'interface Java et d'un ensemble de propriétés dont la sémantique est laissée libre aux utilisateurs. L'invocation d'un service se fait par l'intermédiaire d'un appel de méthode, identique aux appels de méthodes de l'approche objet de Java. La Figure 2.16 illustre l'architecture orientée service de la spécification OSGi™.

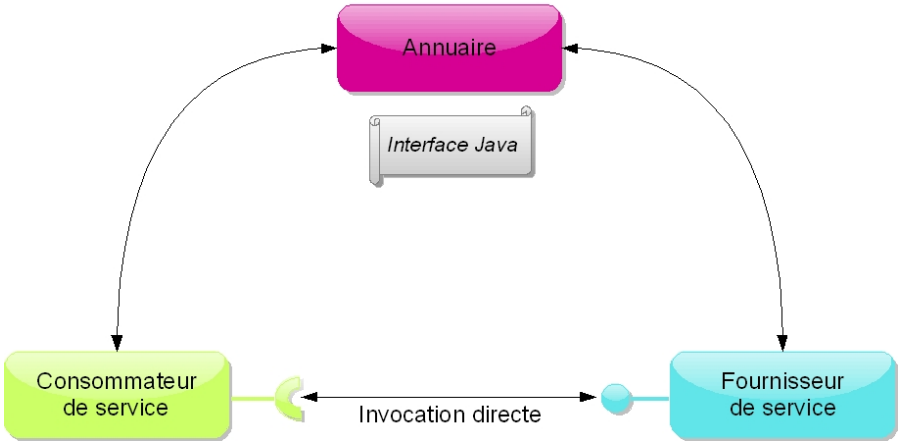


FIGURE 2.16 – Architecture de OSGi™.

Le principe de fonctionnement de l'architecture à services proposée par la spécification est le suivant :

- un fournisseur de service publie son service au niveau du registre de services fournit par OSGi™. Le service est décrit par une interface Java qui définit sa partie fonctionnelle ainsi que la syntaxe pour l'appeler. Cette interface est complétée par un ensemble de propriétés qui sont à l'appréciation du fournisseur.
- un consommateur de service interroge l'annuaire de services pour rechercher le service qui convient à ses besoins, qui peuvent être sur la partie fonctionnelle du service ou bien sur ses propriétés complémentaires. La requête est traitée par l'annuaire qui liste quels sont les fournisseurs disponibles répondant aux besoins.
- le consommateur invoque directement un des fournisseurs de services proposés par l'annuaire.

La spécification OSGi™ procure une architecture orientée service dynamique centralisée. Le dynamisme de cette architecture est supporté en grande partie par l'annuaire de service qui permet de retrouver à tout moment l'ensemble des fournisseurs de services disponibles. Il maintient une correspondance entre les descriptions de services et les fournisseurs de services disponibles. A cela, la plate-forme OSGi™ ajoute un ensemble de primitives et de mécanismes pour créer des applications capables d'être informées dynamiquement de la disponibilité des services. Un fournisseur de service peut alors

être retiré et, par conséquent, les services qu'il fournit deviennent indisponibles. Aucun consommateur ne peut alors l'utiliser.

Il existe plusieurs implantations de la plate-forme OSGi™, autant commerciales que *open source*, telles que Apache Felix [22] de la communauté Apache, Equinox [23] de IBM ou encore Knopflerfish [24] maintenu par Makewave.

La spécification permet les interactions prévues par l'architecture à services mais celles-ci doivent toutes être gérées manuellement par le développeur. Cette tâche est délicate et demande une grande connaissance des mécanismes OSGi™ pour bien traiter tous les cas possibles afin d'éviter les erreurs. Pour remédier à ce problème de gestion manuelle des interactions, la technologie iPOJO, présentée dans la section suivante, propose de les gérer automatiquement.

4.3 iPOJO

La technologie Apache iPOJO [EHL07, EH07], acronyme de *injected Plain Old Java Object*, est une implantation des principes de l'approche à composants orientés service. Elle a été développée au sein de l'équipe Adèle du Laboratoire d'Informatique de Grenoble (LIG). Elle est un sous-projet du projet Apache Felix [25], qui est lui-même une implantation *open source* de la spécification OSGi™ R4.

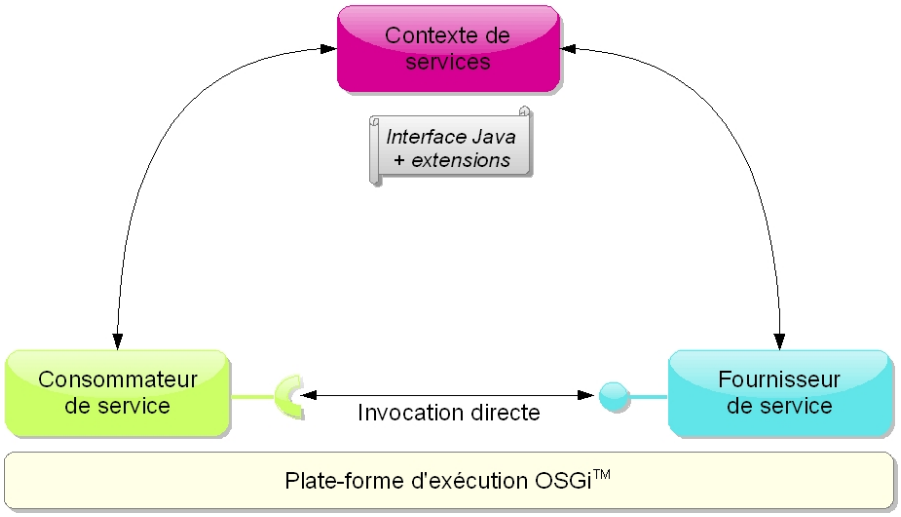


FIGURE 2.17 – Architecture de iPOJO.

22. http://felix.apache.org/site/index.html
23. http://www.eclipse.org/equinox/
24. http://www.knopflerfish.org/
25. http://felix.apache.org/site/index.html

L'objectif principal d'iPOJO est de fournir une plate-forme d'exécution facilitant le développement d'applications basées sur le principe de l'architecture à services dynamique. Cette plate-forme repose sur la spécification OSGi™, et par conséquent, elle hérite d'une partie de ses caractéristiques : elle est en Java, centralisée et supporte le dynamisme. Dans l'optique de simplifier le développement des applications à services dynamiques, iPOJO propose d'appliquer le principe de l'approche à composants orientés service, c'est-à-dire que les services sont implantés sous forme de composants.

La technologie iPOJO propose :

- **un modèle et une machine d'exécution fortement liés :** tous les éléments du modèle existent au moment de l'exécution, ceci afin de rendre plus compréhensible la structure de l'application.
- **une architecture à services dynamique avec isolation :** les interactions de l'architecture à services dynamique sont issues de l'architecture de OSGi™. Cette architecture propose aussi un principe d'isolation des services pour que certains services d'une application puissent rester privés.
- **un modèle de développement simple :** il permet de cacher toute la complexité de l'architecture SOA en utilisant les idées du modèle de développement des *Plain Old Java Object* (POJO) [Fow00] et surtout il cache toute la complexité du dynamisme de OSGi™. Pour cela, la machine d'exécution d'iPOJO fournit une machine d'injection et d'introspection permettant une gestion transparente de ces préoccupations pour le développeur.
- **un langage de composition structurelle :** iPOJO permet de construire des applications à partir d'une composition structurelle de services. La composition est modélisée avec des spécifications de services indépendamment de l'implémentation. Ce découplage est un point fort de iPOJO puisqu'ainsi à l'exécution l'infrastructure choisit une implémentation disponible. Les applications conçues ainsi sont gérées de manière dynamique.
- **des fonctionnalités d'introspection et de reconfiguration dynamique :** ces mécanismes permettent d'avoir un retour sur la structure du système qui change grâce au dynamisme.
- **des mécanismes d'extension :** ils permettent d'ajouter facilement des propriétés non-fonctionnelles comme la persistance, la sécurité ou la qualité de service.

Grâce à ces mécanismes, iPOJO permet de construire plus facilement qu'auparavant des applications à services dynamiques tout en respectant les principes de l'architecture à services dynamique. La simplification de la gestion du dynamisme est un apport très important qui aide considérablement le développeur. De plus, la possibilité de gérer uniquement les propriétés non-fonctionnelles que souhaite l'utilisateur est un atout.

4.4 Synthèse

Les deux technologies OSGi™ et iPOJO sont deux technologies qui sont des implantations des principes de l'approche à composants orientée service. Ces deux technologies respectent les mécanismes de l'approche à composants mais aussi ceux de l'approche à services. Le Tableau 2.2 récapitule les principales fonctionnalités de ces deux technologies.

	OSGi™	**iPOJO**
Spécification	– Java – Aucune propriété non-fonctionnelle	– Java – Propriétés non-fonctionnelles – Extensions possibles
Implantation	– Java – Conditionnée dans des *bundles* – Deux niveaux de dépendances	– Java, composite – Conditionnée dans des *bundles* – Trois niveaux de dépendances
Découverte	– Annuaire de services OSGi™ – Découverte active et passive	– Contexte, pas d'annuaire global – Découverte active et passive
Composition	– Structurelle – Sans ADL	– Structurelle – ADL implicite
Communication	– Java – Synchrone, centralisée	– Java – Synchrone, centralisée
Liaison	– Type : indirect – Autonomique	– Type : indirect – Dynamique

TABLE 2.2 – Comparatif des technologies OSGi™ et iPOJO.

Nous pouvons constater que iPOJO hérite de tous les avantages de OSGi™, comme le dynamisme, mais étend une partie des fonctionnalités, par exemple, il est possible de gérer les propriétés non-fonctionnelles. Le principal avantage de iPOJO est l'automatisation de la gestion du dynamisme qui est une tâche très difficile pour un développeur s'il souhaite réaliser correctement une application. OSGi™ et iPOJO sont cependant des technologies centralisées qui supportent uniquement le langage Java.

OSGi™ et les composants orientés service tels qu'iPOJO deviennent très répandus. OSGi™ est par exemple devenu incontournable dans le domaine des serveurs d'applications (OW2 JOnAS [Obj07], Websphere [IBM05], Glassfish [Sunb],...) et iPOJO est aujourd'hui intégré dans JOnAS et bientôt dans Glassfish.

5 Autres technologies à services

Dans les sections précédentes, nous nous sommes focalisés sur les technologies à services qui sont en plein essor grâce à leurs avancées et leur réactivité face aux différentes demandes du marché. Cependant, les services Web avec WS-BPEL pour leurs compositions, OSGi™ et iPOJO ne sont pas les seuls standards existants pour mettre en œuvre l'architecture à services.

Dans cette partie, nous allons détailler CORBA et Jini qui font partie des premières spécifications de cette architecture. Puis, nous nous intéresserons aux technologies UPnP et DPWS qui voient le jour dans le domaine de la domotique. C'est un environnement qui s'appuie sur une grande partie des caractéristiques du SOA, telles que le faible couplage, l'hétérogénéité des équipements, le dynamisme... Finalement, nous présenterons la spécification la plus connue pour les composants qui est le modèle SCA.

5.1 CORBA et Jini

5.1.1 CORBA

CORBA® [Obj08], acronyme de *Common Object Request Broker Architecture*, est une norme qui a été rédigée par un groupe de travail nommé *Object Management Group* (OMG™). Dans ce groupe, on retrouve les principaux acteurs informatiques tels que Sun Microsystems, Oracle ou encore IBM. L'OMG™ travaille à la définition de « l'architecture distribuée idéale », c'est-à-dire une architecture distribuée dans laquelle des clients émettent des requêtes à destination d'objets qui s'exécutent dans des processus serveurs. Ces objets répondent aux clients par la transmission d'informations bien que ces deux éléments (client et serveur) soient localisés dans des espaces mémoire différents, généralement situés sur des machines distantes.

La norme CORBA®a été créé en 1992 et supporte l'hétérogénéité et l'interopérabilité entre les différents langages de programmation et les environnements informatiques. Ceci est dû en grande partie au principe du langage de description d'interface [26] (IDL). En 1996, la deuxième version de CORBA sort avec comme innovation le protocole de communication IIOP [27] et des améliorations pour la conversion des langages de programmation en IDL et inversement. La dernière version de CORBA, qui date de 2002, s'enrichit avec l'ajout de seize types de services, tels que celui de nommage, l'annuaire des objets, le cycle de vie, la notification d'événements, la sécurité... Parmi ces services, le plus important est le service d'annuaire sur lequel repose la mise en place d'applications dynamiques avec l'infrastructure CORBA. Avec le service d'annuaire, CORBA suit les principes de l'architecture orientée service comme le montre la Figure 2.18.

26. *Interface Description Language*
27. *Internet Inter-Orb Protocol*

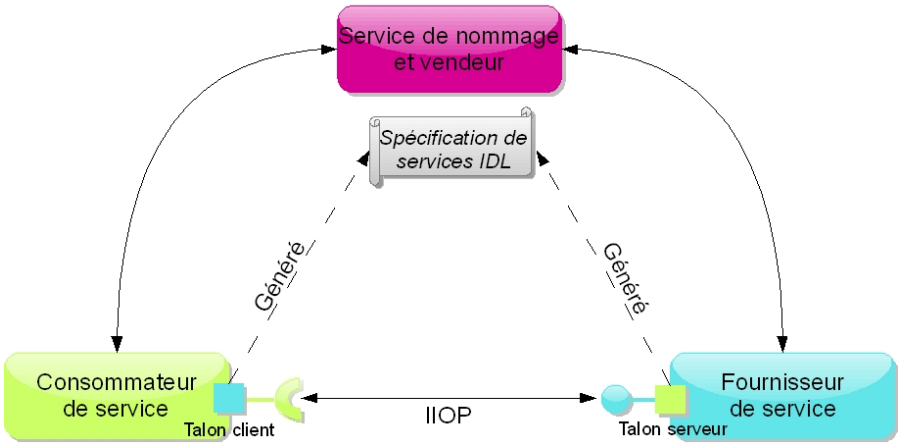


FIGURE 2.18 – Architecture de CORBA.

Un service est décrit avec le langage de description d'interfaces, cette description contient une référence vers l'interface du service ainsi que les propriétés qui le caractérisent. Les informations contenues dans les descriptions sont uniquement syntaxiques et restreintes en nombre. Le fournisseur enregistre la description de service ; ensuite celle-ci est publiée. Le service est alors disponible pour les clients.

Un consommateur de service peut rechercher un service dans un annuaire. La requête d'interrogation de l'annuaire peut être plus ou moins complexe en fonction des besoins du client. Parmi les services disponibles retournés par l'annuaire, le client en choisit un et peut alors l'invoquer en utilisant les méthodes de son interface. En général, le protocole IIOP est utilisé pour interroger un service.

Dans la spécification CORBA, un client peut interroger un annuaire ou une fédération d'annuaires. En effet, pour augmenter les chances d'obtenir un service qui corresponde parfaitement aux attentes du client, un annuaire peut interagir avec d'autres annuaires tout en restant transparent lors de son utilisation.

CORBA suit les principes de l'architecture orientée service mais ne supporte pas la notification de l'arrivée ou du départ d'un service comme dans l'architecture à service dynamique. Un consommateur doit toujours s'assurer de la disponibilité du service avant de l'utiliser et le libérer sitôt qu'il n'en a plus l'utilité. Cependant, la composition de services CORBA n'est pas fournie dans la spécification, certains travaux proposent des compositions comportementales [MPN98] ou structurelles [MTK97].

5.1.2 Jini

Jini [Sund] est une spécification proposée à l'origine par Sun Microsystems. Actuellement, cette spécification fait partie du projet River[28], géré par Apache. Elle permet de construire un réseau de dispositifs communicants capables de fonctionner avec une machine virtuelle Java (JVM). L'utilisateur a ainsi la possibilité d'accéder facilement à distance à ce réseau et d'utiliser tous les dispositifs qui y sont branchés. Jini est indépendant de tout système d'exploitation.

Les dispositifs communicants supportés par Jini peuvent être autant matériels que logiciels, ils sont considérés comme des objets indépendants qui peuvent communiquer. Ainsi, Jini peut les réunir en *fédération* d'objets qui s'installent automatiquement et qui fonctionnent dès qu'ils sont branchés dans un réseau local dynamique.

Jini définit un service comme une entité logicielle qui peut être utilisée par l'ensemble des acteurs du réseau, c'est-à-dire une personne, un programme ou un autre service. Un service est une fonctionnalité utilisable à distance, comme par exemple, un service d'impression offert par une imprimante. Le service ainsi que ses propriétés sont décrits avec une interface Java. Cette définition n'est que syntaxique.

Jini suit les principes de l'architecture à services ; nous retrouvons donc dans son architecture le fournisseur de service, le service de recherche et le consommateur de service en tant qu'acteurs, comme représentés dans la Figure 2.19.

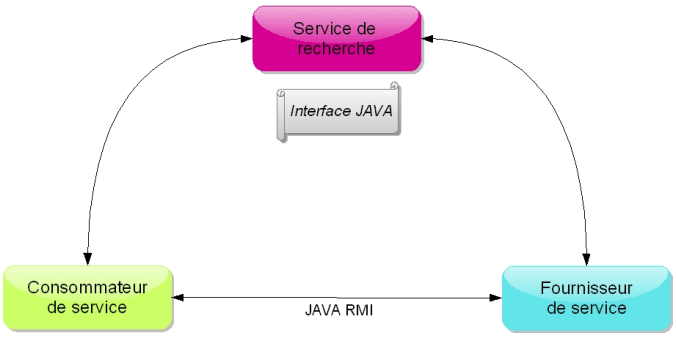


FIGURE 2.19 – Architecture de Jini.

Un service est enregistré dans un service de recherche par son fournisseur. Le service de recherche peut être contacté à une adresse fixe ou bien découvert. Une fois que le service est enregistré et publié par le service de recherche, le client doit découvrir le service de recherche. Ensuite, il peut rechercher et utiliser le service. A ces mécanismes de l'architecture à services, Jini en ajoute un pour gérer de façon explicite les politiques de libération des instances des services avec un système de bail. Lors de la publication du service par le fournisseur dans le registre, il est possible d'indiquer la durée du bail.

28. `http://incubator.apache.org/projects/river.html`

Lorsque le bail se termine, le fournisseur doit renouveler son bail, sinon le service est supprimé du registre. Ce mécanisme de bail est aussi utilisé pour la découverte passive de services par le consommateur.

Toutefois avec ces briques de base de l'architecture à service, Jini ne propose pas de modèle de composition de services. Le développeur est libre de gérer la composition de services ainsi que le dynamisme dû aux arrivées et aux départs des services. Cependant, un langage de composition a été proposé par Y. Huang [Hua03] dans le but d'exécuter des compositions comportementales au-dessus de Jini. Ceci consiste simplement à traduire en Java la composition écrite dans ce langage.

5.1.3 Synthèse

CORBA et Jini font partie des plus anciennes technologies qui implantent les principes de l'architecture orientée service. Le Tableau 2.3 récapitule les informations de ces technologies en fonction des éléments intéressants de l'architecture à services.

	CORBA	**Jini**
Spécification	– IDL – Aucune propriété non-fonctionnelle	– Interface Java – Aucune propriété non-fonctionnelle
Implantation	– Nombreux langages – Génération des talons – Déploiement manuel des talons	– Java
Découverte	– Service de nommage et vendeur – Découverte active	– Service de recherche – Découverte active et passive
Composition	– Non (Client/Serveur)	– Non
Communication	– Principalement IIOP – Synchrone, asynchrone, publication/souscription	– Principalement RMI – Synchrone, asynchrone, décentralisée
Liaison	– Type : indirect – Dynamique	– Type : indirect – Dynamique

TABLE 2.3 – Comparatif des technologies CORBA et Jini.

Ces deux technologies supportent par défaut les principes de base de l'architecture : les services sont spécifiés et implantés, ils peuvent être découverts. Jini restreint l'implantation des services à un unique langage alors que CORBA permet d'intégrer plusieurs langages de programmation. Cependant, la composition de services n'est pas prise en considération par ces technologies ; c'est une limite à leur utilisation aujourd'hui.

5.2 UPnP et DPWS

La popularité des objets communicants comme, par exemple, les téléphones portables, les assistants personnels, etc., ont poussé le développement des technologies de communication comme le Bluetooth [29], UPnP et DPWS. Ce sont des protocoles de haut niveau, qui cachent la complexité des technologies utilisées tout en permettant non seulement des communications sans fils entre différents équipements hétérogènes de réseaux domestiques ou d'entreprise mais aussi, pour UPnP et DPWS, leur découverte. Dans cette partie, nous allons détailler UPnP et DPWS qui sont deux spécifications basées sur le principe de l'architecture orientée service dynamique et qui sont dédiées aux équipements des réseaux domestiques ou d'entreprise.

5.2.1 UPnP

UPnP [UPn08], acronyme de *Universal Plug and Play*, est une spécification issue d'une initiative industrielle et gérée par l'UPnP Forum. L'objectif de cette spécification est de pouvoir simplifier au maximum la connexion de dispositifs communicants hétérogènes et la mise en place de réseaux domotiques. UPnP est inspiré et dérivé de la technologie *Plug and Play*, qui permet d'attacher de manière dynamique de nouveaux périphériques à un ordinateur.

La spécification UPnP propose un ensemble de protocoles, basés sur des protocoles standard de l'Internet, pour mettre en place une architecture orientée service dynamique spécialisée dans la domotique. La Figure 2.20 illustre cette architecture de UPnP :

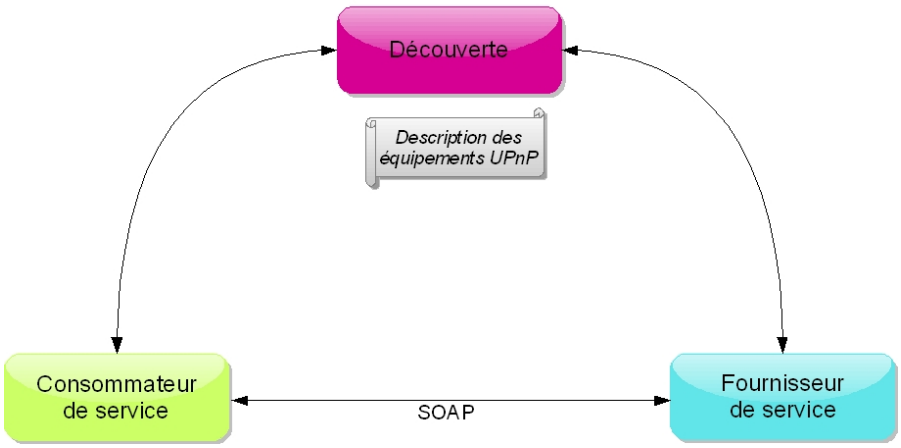


FIGURE 2.20 – Architecture de UPnP.

29. http://www.bluetooth.com/

Le but de UPnP est de pouvoir découvrir de nouveaux équipements dans un réseau ; c'est pourquoi les équipements ainsi que leurs services sont décrits dans un format propre à UPnP et basé sur XML. Lorsqu'un équipement rejoint un réseau, il s'annonce auprès de tous les autres intervenants présents dans ce réseau. Pour qu'un équipement ait une connaissance des autres équipements du réseau, il peut émettre une requête pour récupérer les informations des différents équipements. Comme nous sommes dans une architecture à services dynamique, il est aussi possible qu'un équipement soit arrêté. Dans ce cas, cet équipement est capable d'envoyer un message pour notifier son départ aux autres équipements.

Finalement, UPnP se propose de fournir une spécification pour la communication et la découverte d'équipements et de leurs services dans des réseaux domestiques ou d'entreprise. L'évolution des technologies dans le domaine de la domotique nécessite de pouvoir mettre en place des applications en fonction des équipements découverts dans les maisons ; ainsi UPnP commence à devenir un standard de plus en plus populaire.

Toutefois, il est clair que UPnP est centré sur les équipements plutôt que sur les applications que l'on peut réaliser avec ces équipements, ce qui est un obstacle à l'intégration de ces équipements dans les applications. Mais, il manque aussi des fonctionnalités importantes à UPnP. En particulier, la sécurité n'est pas du tout abordée, alors qu'il est essentiel dans des réseaux domotiques de pouvoir restreindre certains équipements et/ou services avec des authentifications par exemple.

5.2.2 DPWS

DPWS [JMS05, ZBBG07] est l'acronyme de *Device Profile for Web Services*. C'est une spécification [Mic06] proposée et supportée par Microsoft. Windows Vista, qui est la dernière génération de système d'exploitation de Microsoft, intègre nativement cette spécification. Cette dernière permet de simplifier la mise en place d'appareils dans un réseau domestique ou dans une entreprise. La politique affichée par Microsoft avec cette stratégie monopolistique est de remplacer UPnP.

La spécification DPWS se base sur la technologie des services Web. Elle part du constat que les services Web sont aujourd'hui une technologie très riche en standards et en fonctionnalités qui répondent à de nombreux besoins. Cependant, pour être adaptable à tous les services Web, Microsoft propose un noyau de fonctionnalités restreintes basées sur des spécifications des services Web :

- **l'envoi et la réception** de messages sécurisés pour un service Web avec la spécification WS-Security [OAS04b],
- **la découverte dynamique** de service Web avec la spécification WS-Discovery [Mic05],
- **la description** d'un service Web avec la spécification WS-MetadataExchange [W3C08a],
- **la souscription et la réception** d'événements pour un service Web avec la spécification WS-Eventing [W3C05a].

L'architecture de DPWS suit le principe de l'architecture orientée service, en particulier celle des services Web ; elle est illustrée dans la Figure 2.21 ci-après :

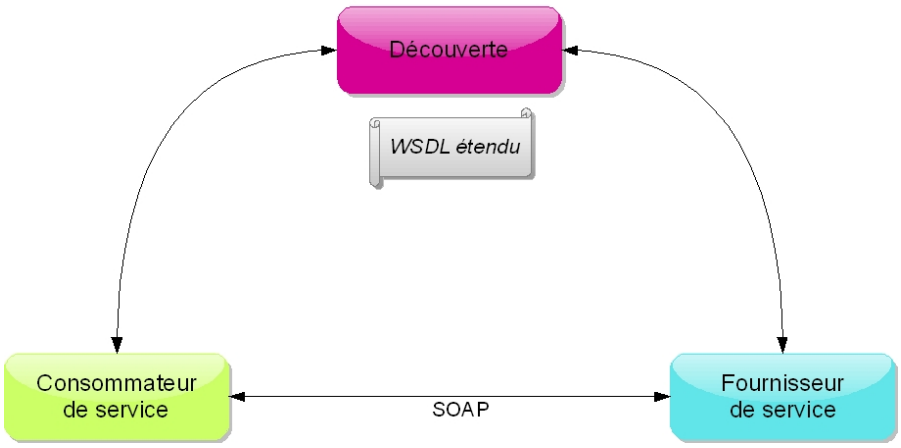


FIGURE 2.21 – Architecture de DPWS.

Après avoir obtenu une adresse IP [30] sur le réseau, l'équipement doit découvrir tous les équipements du réseau et signaler son existence ainsi que les services qu'il propose. En général, ces communications sont réalisées en mode *multicast*. Les fonctionnalités d'un équipement sont données dans un fichier de description WSDL étendu. Ensuite, il est possible de demander un service à un équipement. Cette communication se fait avec le protocole SOAP et peut être asynchrone.

Pour conclure, la spécification de DPWS est fortement inspirée de celle de UPnP ; d'ailleurs elle se veut être la nouvelle version de UPnP. Les fonctionnalités offertes sont semblables à celles d'UPnP, à la différence qu'elles sont fournies de façon plus claire et plus précise, grâce, entre autres, aux spécifications des services Web. Un autre avantage de cette spécification est l'ajout de la sécurité. Cependant, malgré les améliorations apportées et les efforts de Microsoft, DPWS n'est implanté que sur très peu d'équipements et il est limité à des réseaux locaux ; par conséquent, l'adoption de ce standard risque d'être un peu longue.

5.2.3 Synthèse

Les deux technologies UPnP et DPWS sont en train de devenir des standards pour la domotique. Elles s'appuient sur l'architecture orientée service dynamique. Ainsi les équipements sont décrits et disponibles pour être utilisés par des clients. Nous présentons dans le Tableau 2.4 un récapitulatif des points importants de l'architecture à services pour UPnP et DPWS.

30. WS-Addressing [W3C04a]

	UPnP	DPWS
Spécification	– Description d'équipement UPnP – Aucune propriété non-fonctionnelle	– WSDL étendu – Propriété non-fonctionnelle : sécurité
Implantation	– Nombreux langages – Serveur HTTP côté équipement – Proxy généré côté client	– Nombreux langages – Serveur HTTP côté équipement – Proxy généré côté client
Découverte	– Déclaration *multicast* – Découverte active et passive	– Déclaration *multicast* – Découverte active et passive
Composition	– Client/Serveur	– Client/Serveur
Communication	– SOAP – Synchrone, à événements	– SOAP – Synchrone, à événements
Liaison	– Type : direct – Dynamique	– Type : direct – Dynamique

TABLE 2.4 – Comparatif des technologies UPnP et DPWS.

Le Tableau 2.4 montre que les deux technologies UPnP et DPWS sont très proches. La spécification de DPWS a montré un effort de standardisation au niveau du fichier de description. La mise en place d'un standard commun à tous les équipements facilite leur utilisation. Un autre atout très important dans DPWS est la gestion de messages sécurisés, ceci permet de restreindre l'accès à certains équipements et ainsi de réaliser des applications plus sûres.

Le point le plus important pour ces technologies est le dynamisme issu de l'architecture orientée service dynamique. Le dynamisme permet en effet d'envisager des applications réactives à l'ajout ou au retrait d'équipements dans un réseau domestique ou d'entreprise. Par exemple, dans une maison, on peut créer une application qui utilise une télévision et un assistant personnel, il est clair que la télévision restera dans la maison alors que l'assistant personnel risque de suivre les déplacements de son propriétaire à l'extérieur de la maison.

Enfin, on peut regretter le manque de maturité logicielle de ces deux technologies. L'expérience montre que la mise en place d'applications avec UPnP et/ou DPWS requiert un développement logiciel et, surtout, une phase de tests et de robustesse très coûteuse. On peut également douter de la capacité de ces technologies à passer à l'échelle. UPnP, par exemple, est très verbeux et émet de nombreux messages sur le réseau.

5.3 SCA

Le consortium OSOA [31] a été à l'initiative de la spécification SCA, acronyme de *Service Component Architecture*, pour définir une architecture orientée service. Aujourd'hui SCA est un ensemble de spécifications gérées par le consortium OASIS, qui a pour but de le standardiser. L'objectif de SCA est de simplifier l'écriture d'applications à base de composants sans se soucier du langage d'implantation du composant. Il existe plusieurs implantations de la spécification SCA telles que Websphere proposé par IBM [IBM05], Aqualogic de BEA Systems [BEA08] ou encore Tuscany d'Apache Software Foundation [Apa08].

Une application est définie comme un ensemble de composants logiciels qui interagissent. Un composant SCA implante la logique métier dans un langage de programmation quelconque. Il est composé de trois parties :

- les **services** qui sont les fonctionnalités de la logique métier rendues accessibles aux autres composants. La description des services est faite selon la technologie utilisée pour réaliser la logique métier. Par exemple, on a une interface Java pour un composant implanté en Java ; pour des procédés BPEL, un fichier WSDL donne la description du service.
- les **références** qui sont les fonctionnalités requises pour le bon fonctionnement du composant. Ce sont des dépendances à d'autres services fournis par d'autres composants.
- les **propriétés** qui sont des informations de configuration nécessaires lors de l'instanciation du composant.

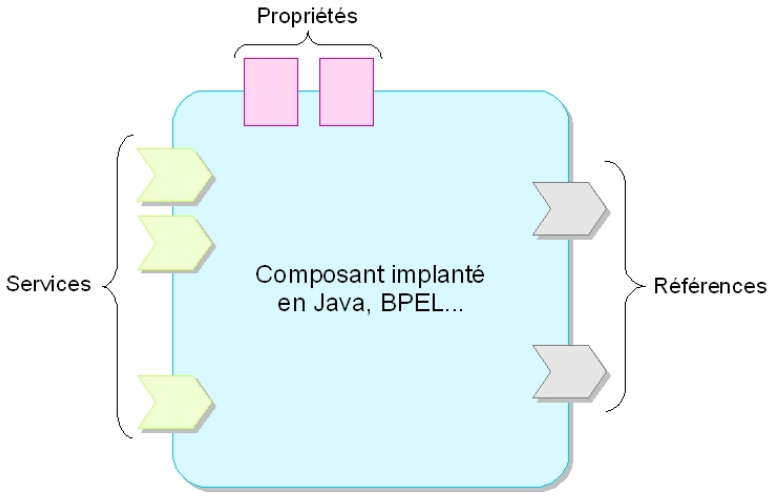


FIGURE 2.22 – Structure d'un composant SCA.

31. *Open Service Oriented Architecture*

La spécification SCA a été proposée pour créer des applications ; ces dernières sont alors le résultat de l'assemblage de composants SCA. Cet assemblage suit le modèle de la composition structurelle qui permet d'obtenir une fonctionnalité plus complexe à partir des services fournis par différents composants. Un tel assemblage est un composite qui contient des composants et des liaisons qui représentent les liens entre composants. La nature des composants dans un composite peut être ou non hétérogène. La composition hiérarchique est aussi possible : un composite peut être lui-même utilisé dans un autre composite.

L'architecture obtenue par assemblage de composants est donnée dans un langage de description appelé SCDL [32]. Ce langage permet de décrire les composites à différents niveaux de granularité et d'abstraction. Un composite est une unité d'encapsulation et de visibilité. En effet, les composants qui sont à l'intérieur d'un composite, ne sont pas visibles de l'extérieur du composite et ne peuvent pas être liés à des services provenant de l'extérieur. Tout comme pour un composant, un composite peut exposer ses services, ses références et ses propriétés. Ces informations sont issues des composants du composite. La Figure 2.23 est une illustration du modèle de composition de SCA.

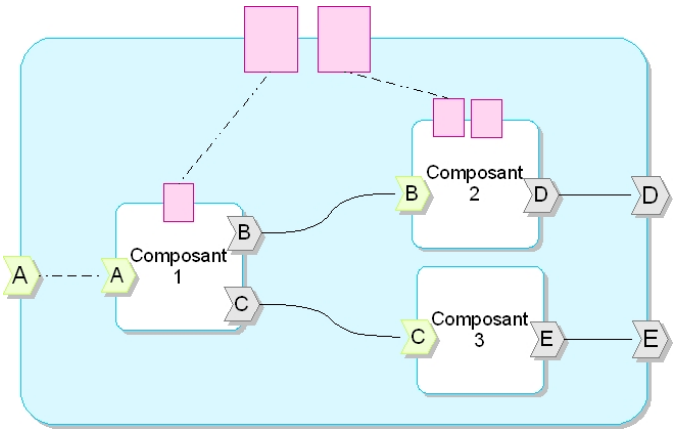


FIGURE 2.23 – Exemple de composition de composants SCA.

En conclusion, la spécification SCA se base sur un ensemble de concepts favorables à la réalisation d'applications à base de composants. La composition est définie clairement dans un langage de description d'architecture qui sépare les services des liaisons. Un autre point fort de SCA est qu'il est possible de créer des compositions structurelles hiérarchiques en imbriquant des composants composites implantés dans n'importe quel langage. Cependant, la composition n'est faite qu'au niveau conceptuel et non au niveau exécution, par conséquent le dynamisme et la reconfiguration à l'exécution ne sont pas traités.

32. *Service Component Definition Language*

6 Synthèse

Ce chapitre nous a permis d'avoir une vision générale de l'approche orientée service et de son utilisation. Le service est l'élément principal de cette approche. Il peut être vu comme une entité logicielle qui fournit un ensemble de fonctionnalités et d'aspects non-fonctionnels spécifiés dans une description de service. Cette description permet au fournisseur de service de publier son service et au consommateur de le rechercher dans un annuaire, de le sélectionner et de l'invoquer. La description de service est un moyen de cacher à l'utilisateur les détails techniques de l'implantation du service tels que la technologie d'implantation et la plate-forme d'exécution. Quant à lui, le fournisseur du service propose un service sans même connaître le contexte d'utilisation de celui-ci.

L'avantage certain de cette approche est que seule la description du service est partagée entre les différents acteurs ; ceci permet d'obtenir un très faible couplage. Cette propriété forte de l'approche à services facilite l'intégration de services hétérogènes pour réaliser une application à base de services. Une telle application peut être réalisée sous forme d'une composition structurelle ou bien d'une composition par procédés.

Dans ce chapitre, nous avons également vu qu'en introduisant deux nouvelles primitives à cette approche, nous obtenions du dynamisme au niveau de l'exécution. En effet, le consommateur du service peut s'adapter lors de l'exécution en choisissant ou en changeant de fournisseur de service. Cette caractéristique très intéressante est appelée liaison à retardement.

Les services Web sont une des technologies les plus répandues qui met en œuvre une grande partie des principes de l'approche à service. Cette technologie se base sur les standards Internet que sont HTTP, XML, SOAP, WSDL et UDDI. Il existe d'autres technologies à services comme DPWS et UPnP, qui ont pour domaine applicatif l'informatique *pervasive*, comme OSGi™et iPOJO qui sont des technologies regroupant les avantages des composants et des services.

Cependant, cette approche est complexe à mettre en place par les développeurs d'applications puisqu'ils doivent avoir, en plus de l'expertise métier, une connaissance approfondie de l'approche à services et de ses technologies. De plus, les applications ne sont pas seulement des briques fonctionnelles, elles nécessitent généralement des éléments non-fonctionnels tels que la sécurité, la gestion des transactions,... Et, pour l'instant, de nombreuses technologies à services ont vu le jour mais elles se sont concentrées essentiellement sur les parties fonctionnelles des services. Des standards pour les propriétés non-fonctionnelles commencent seulement à être mis en place, en particulier pour les services Web.

3

La sécurité pour les services

Dans une première partie, nous allons mettre en évidence les différents types d'attaques et de menaces qui existent dans l'architecture à services. La deuxième section présente les principes généraux de la sécurité en définissant ses objectifs et les concepts associés. Ensuite, dans une troisième partie, nous détaillerons les techniques de base utilisées pour sécuriser des systèmes ; ces techniques comprennent le chiffrement, la signature et les certificats. Nous étudierons, dans une quatrième partie, des technologies appropriées aux caractéristiques de l'architecture à services, en particulier, XML Digital Signature et XML Encryption, qui adaptent respectivement la signature électronique et le chiffrement aux messages XML. Ces deux recommandations sont une base pour les autres technologies étudiées, telles que XKMS, XACML, XrML ou encore SAML. Avant de conlure, nous focaliserons notre étude sur une technologie adaptée plus particulièrement aux services Web : WS-Security. Nous détaillerons les différentes recommandations qui le composent. Nous présenterons aussi une implantation de cette recommandation WS-Security.

1 Attaques et menaces dans les architectures à services

Les architectures à services ont été proposées pour permettre de réaliser facilement des applications à partir de services. Les services sont distribués, déployés sur des plates-formes hétérogènes et utilisés par de nombreux utilisateurs. Ces caractéristiques sont considérées comme des points forts de l'architecture à services.

Néanmoins, les architectures à services se basent sur un ensemble de requis techniques nécessaires à leur bon fonctionnement. La distribution des services implique l'existence d'un réseau qui permette les communications entre services. Les services sont exécutés sur des plates-formes hétérogènes qui doivent chacune être administrée correctement. Les plates-formes à services doivent aussi être capables de supporter un nombre important d'utilisateurs. Finalement, l'approche à services est un paradigme supplémentaire qui s'appuie sur une pile de technologies comprenant le réseau avec un protocole de communication ainsi que la plate-forme d'exécution, comme l'illustre la Figure 3.1. Les communications entre services doivent passer par chacun des éléments de la pile, autant à l'émission qu'à la réception d'un message. L'approche à services présente des avantages pour la construction d'application à base de services mais ne règle pas les problèmes des couches inférieures.

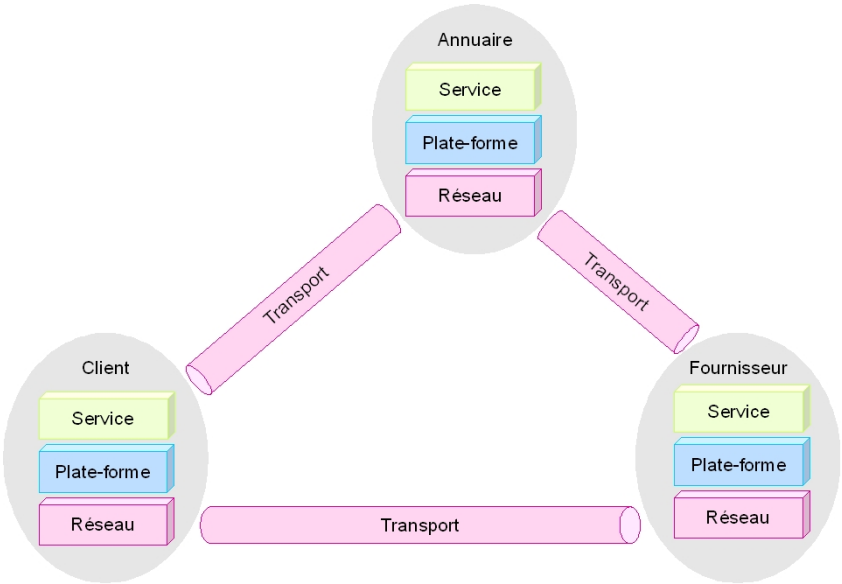


FIGURE 3.1 – Communications à travers différentes couches techniques.

Chacune des couches de cette pile comprend des failles de sécurité internes et héritées des niveaux inférieurs. Nous pouvons noter que ces failles de sécurité sont issues de la structure client/serveur qui est mise en place entre les différents acteurs de l'approche à services. Les problèmes de sécurité qui existent dans l'architecture client/serveur sont des problèmes :

- d'écoute du réseau qui permet l'espionnage des données échangées,
- de tromperie sur l'identité d'une des parties,
- de détournement d'informations qui circulent sur le réseau,
- de saturation du serveur.

Ces failles de sécurité existent entre le client et le fournisseur de service mais aussi entre le fournisseur et l'annuaire ainsi qu'entre le client et l'annuaire. A chaque fois, les communications sont faites suivant une architecture client/serveur. A ces problèmes, nous pouvons aussi ajouter les problèmes intrinsèques au service, à l'annuaire et au client. Le service, tout comme l'annuaire, peut avoir des failles internes de sécurité : par exemple, s'il nécessite un accès à une base de données, cette dernière doit aussi être sécurisée.

A ceci s'ajoutent les failles de sécurité qui existent au niveau physique du réseau. Un réseau physique est composé de nombreuses machines qui sont, par exemple, des serveurs pour les services, pour les bases de données... Chaque machine a des failles de sécurité en fonction des ports de communication qui sont ouverts, de son système d'exploitation... Pour prévenir d'une partie des risques, il existe des moyens techniques à mettre en place comme créer une zone démilitarisée et/ou installer des pare-feu.

Il est intéressant de noter que les services Web, qui sont la technologie la plus utilisée pour implanter l'architecture à services, utilisent le protocole SOAP pour les communications entre les différentes entités. Le protocole SOAP est basé sur le protocole HTTP, qui n'est pas sécurisé à la différence du protocole HTTPS [1]. De plus, pour supporter l'hétérogénéité des plates-formes, le protocole SOAP a été créé pour passer les pare-feu, grâce aux caractéristiques du protocole HTTP. L'utilisation de ce protocole rend donc très vulnérable les systèmes basés sur les services Web.

Dans cette section, nous allons décliner les quatre types d'attaques possibles pour les architectures à services, en particulier entre le client et le fournisseur : l'écoute du réseau et l'analyse du trafic, la tromperie, le détournement d'information et le déni de service. Pour chacun de ces types d'attaques, nous montrerons le fonctionnement de l'attaque avec un scénario simple. Puis, nous détaillerons les conséquences sur le système et finalement, nous énoncerons quelques solutions pour éviter de telles attaques.

1.1 Attaque passive d'écoute du réseau et d'analyse du trafic

Le consommateur et le fournisseur de service communiquent via un réseau Internet ou Intranet. Ces communications peuvent être à tout moment interceptées par une personne ou une machine malveillante à des fins délictueuses, comme illustrées dans la Figure 3.2.

1. HTTPS, pour *HyperText Transfer Protocol Secured*, est la version sécurisée du protocole HTTP.

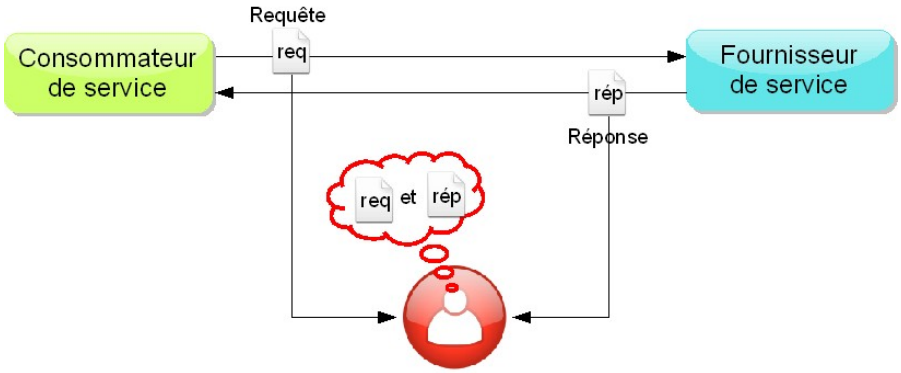


FIGURE 3.2 – Divulgation d'informations.

Le but de cette attaque est de récupérer des informations provenant du consommateur et du fournisseur de service. Pour ce faire, une machine écoute les communications et vole les informations transmises entre les deux parties. Ces informations peuvent être ensuite divulguées à n'importe qui. Pour lutter contre ce type d'attaque, il faut que les informations qui circulent sur le réseau soient et restent confidentielles, c'est-à-dire compréhensibles uniquement par le consommateur et le fournisseur de service. Une solution pour rendre les informations confidentielles est de les chiffrer.

1.2 Tromperie

L'attaque par tromperie est une attaque plus sophistiquée que l'attaque par écoute du réseau : une personne ou une machine malveillante se fait passer pour un « vrai » client auprès du fournisseur de service. La machine malveillante interroge le fournisseur avec les informations d'un « vrai » client et récupère des informations, qui peuvent être confidentielles. La Figure 3.3 synthétise le fonctionnement de cette attaque.

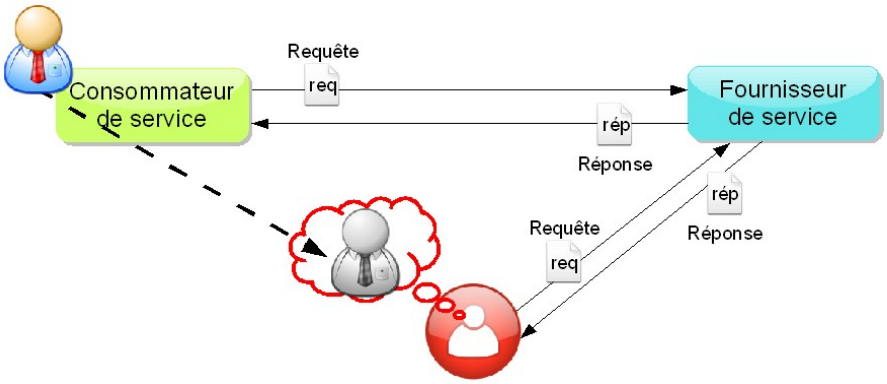


FIGURE 3.3 – Tromperie sur l'identité du client.

Pour cette attaque, la machine malveillante émet des messages en volant l'identité d'un « vrai » client. Les messages sont donc échangés entre le fournisseur et la machine malveillante sans que le fournisseur se rende compte de la supercherie. Ce type d'attaque peut se faire après avoir écouté longtemps le réseau pour utiliser les informations récupérées ou en piratant les informations du client sur sa machine.

Pour que le fournisseur de service soit sûr de l'identité de son client, il faut mettre en place un système d'authentification qui permette de reconnaître les « vrais » clients des pirates. En plus de ce système d'authentification, il faut que les informations d'authentification ainsi que les autres informations échangées restent confidentielles entre le client et le fournisseur pour qu'il n'y ait pas de vol ni d'écoute par une machine malveillante.

1.3 Détournement d'informations

Le détournement d'informations est une attaque qui a des effets différents par rapport aux deux attaques présentées précédemment. En effet, les informations du consommateur sont détournées par une personne ou une machine malintentionnée et elles sont modifiées à des fins malveillantes.

Le principe de cette attaque est d'altérer les données du consommateur afin d'endommager le service du fournisseur. La personne ou la machine malintentionnée intercepte un message d'un « vrai » client s'adressant au fournisseur. Elle modifie le message initial en y insérant du code malveillant. Le message est ensuite envoyé au fournisseur de service qui l'interprète comme un message provenant du client qu'il connaît. Le code ajouté peut, par exemple, être un virus qui endommage le fournisseur de service. La Figure 3.4 illustre le scénario de cette attaque.

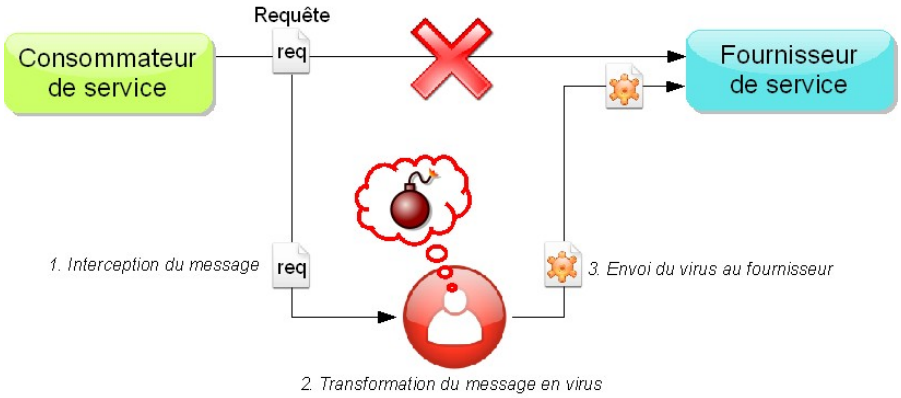


FIGURE 3.4 – Détournement d'informations.

Cette attaque a pour conséquence immédiate de faire perdre les informations contenues dans le message du client. Dans un deuxième temps, elle endommage le serveur qui est alors dans l'incapacité de continuer à répondre correctement aux autres requêtes.

Pour lutter contre ce type d'attaque, il faut que les informations qui circulent restent intègres, c'est-à-dire qu'il faut pouvoir s'assurer que les informations émises n'ont pas été modifiées ou détruites jusqu'à leur réception. Une pratique courante est de chiffrer les informations et d'ajouter une somme de contrôle [2].

1.4 Déni de service

L'attaque par déni de service [3] est une attaque qui a pour but de rendre indisponible le service pendant un temps indéterminé. Cette menace concerne en général le fournisseur de service. Son but n'est pas de s'approprier ou d'altérer les données du service, mais uniquement de nuire à la performance et à la réputation du service. Le fournisseur de service peut ainsi perdre la confiance de ses clients.

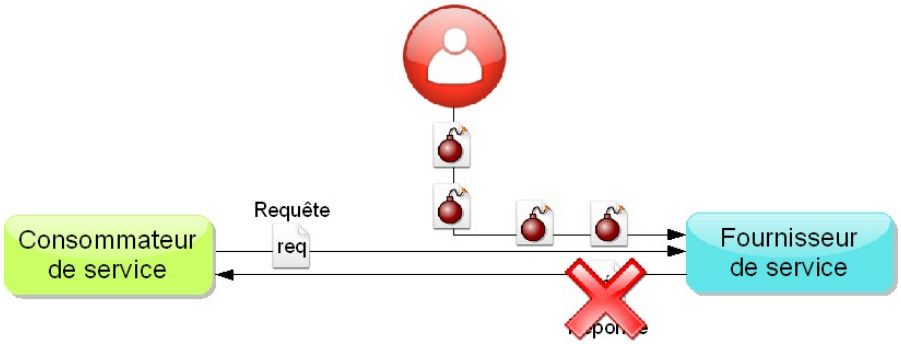


FIGURE 3.5 – Interruption de service.

Le principe de l'attaque par déni de service consiste à saturer le fournisseur de service, en lui envoyant, notamment trop de messages à traiter ou des messages empoisonnés. Pour la technologie des services Web, les messages XML permettent d'attaquer les services. En effet, il suffit d'envoyer un message XML empoisonné. Ce message peut nécessiter, par exemple, une récursion infinie pour l'étape de *parsing* du message à la réception de celui-ci par le fournisseur. Une autre solution est de saturer le service en envoyant un trop grand nombre de messages, le serveur prendra beaucoup de temps pour les traiter tous et sera donc indisponible pour les « vrais » clients. Il existe d'autres moyens de surcharger un service comme expliqués dans les travaux de M. Yunus et R. Mallal [YM05].

Les méthodes d'attaque pour saturer un service sont finalement assez simples mais assez difficiles à détecter, puisque les messages malveillants se font passer pour des messages normaux. Ces messages malveillants sont traités par le service et ils peuvent nécessiter plus de ressources que d'autres à cause, en particulier, de récursion infinie ou de

2. En anglais : *checksum*.
3. En anglais : *Denial of Service*, abrégé en DoS.

données trop grandes qui posent des problèmes de tampon. Le seul moyen pour s'assurer du bon fonctionnement des services est de mettre en place une surveillance des services qui permettent de suivre le temps de traitement des messages et des opérations.

1.5 Synthèse des attaques et des menaces

Le Tableau 3.1 synthétise les différentes attaques que nous avons présentées pour les architectures à services. La liste de ces attaques n'est pas exhaustive, mais elle regroupe la majorité de celles-ci qui peuvent être déclinées sous plusieurs formes. Ensuite, nous présentons les menaces encourues pour chacune des attaques étudiées, puis les conséquences sur le système. La dernière colonne propose des solutions à mettre en œuvre pour lutter contre les attaques possibles. Ces solutions seront étudiées dans la section 2.2. Elles sont à mettre en place dès la phase de conception de l'application.

Types d'attaques	**Menaces**	**Conséquences**	**Contre-mesures**
Ecoute du réseau et analyse du trafic	Divulgation d'informations	Atteinte aux données privées	Confidentialité des informations
Tromperie	Usurpation d'identité	Atteinte aux données privées Représentation erronée de l'utilisateur	Authentification de l'utilisateur Confidentialité des informations
Détournement d'informations	Virus	Perte de données Machine « vérolée » du côté du fournisseur	Intégrité des informations
Déni de service	Surcharge du service	Interruption de service Gêne pour les clients Système endommagé	Surveillance de la disponibilité

TABLE 3.1 – Récapitulatif des attaques et des menaces dans les architectures à services.

Il est à noter que les attaques répertoriées dans le Tableau 3.1 sont des attaques qui ne sont pas liées exclusivement à l'architecture à services, mais en partie à ses caractéristiques comme la distribution, l'hétérogénéité... Les attaques sont en fait des attaques classiques pour les architectures client/serveur distribuées. La difficulté est de mettre en place les contre-mesures dans l'architecture à services pour les en préserver.

2 Sécurité : objectif, définition et concepts

2.1 Principe et définition

Nous allons commencer par définir la sécurité informatique en s'appuyant sur la définition suivante :

> « *La sécurité informatique est l'ensemble des techniques qui assurent que les ressources (matérielles ou logicielles) du système sont utilisées uniquement dans le cadre où il est prévu qu'elles le soient.* »

La sécurité informatique met en œuvre différents moyens techniques, organisationnels, juridiques et humains. Ces moyens sont mis en place pour garantir la sécurité du système. La sécurité comprend autant la prévention des attaques que la surveillance et la remise en état du système après une attaque. Par conséquent, la sécurisation d'un système se divise en plusieurs étapes, qui ont lieu tout au long du cycle de vie. La Figure 3.6 représente chacune des étapes de la sécurisation en fonction des phases du cycle de vie d'un logiciel.

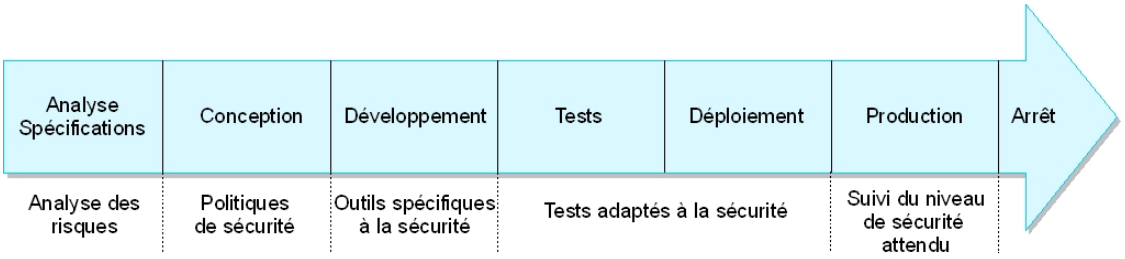


FIGURE 3.6 – Cycle de vie sécurisé d'un système.

La première étape est de faire une analyse des risques du système. Cette étape permet de déterminer tous les risques potentiels qui peuvent exister pour le système. Une mesure des risques doit être réalisée en fonction de la probabilité ou de la fréquence de leurs apparitions et de leurs effets possibles. Il faut aussi évaluer le coût de chaque risque ainsi que la sécurisation du système face à ceux-ci. La maîtrise des risques permet de protéger efficacement le système contre ces risques connus.

Suite aux résultats de l'analyse des risques, il faut mettre en place une politique de sécurité qui permette de définir le cadre d'utilisation des ressources du système et d'identifier les techniques de sécurisation à mettre en œuvre. La politique de sécurité est une réponse apportée aux risques encourus et choisis.

La phase de développement a pour objectif de créer des outils spécifiques à la sécurité et adaptés au système, ou d'intégrer des techniques de base de la sécurité, qui seront présentées dans la partie suivante. Ces outils doivent répondre à la politique de sécurité

qui a été spécifiée dans l'étape précédente. Avant de le mettre complètement en production, il faut réaliser des tests du système ainsi que des tests spécifiques à la sécurité pour évaluer les mécanismes de sécurité qui ont été mis en place.

Pendant la phase de production, le système doit fonctionner conformément aux spécifications et en respectant les politiques de sécurité. Pour s'assurer de son bon fonctionnement, généralement, des systèmes de surveillance sont mis en place. Ils permettent de vérifier qu'il n'y pas d'intrusion ou d'essai d'intrusion dans le système.

Pour conclure, notons que la sécurité se glisse dans toutes les étapes du cycle de vie du système. La réalisation d'un système sécurisé est donc très complexe, puisqu'elle nécessite une maîtrise de la partie métier du système mais aussi des diverses formes de sécurité qui entrent en jeu dans le cycle de vie.

La sécurité est un domaine vaste et complexe, pour lequel il existe une grande diversité de termes, de techniques et de technologies. Tous ses éléments ne sont pas toujours clairement différenciés. Dans ce travail, nous nous sommes attachés à bien séparer trois niveaux à savoir les concepts, les techniques et les technologies.

2.2 Concepts

Les principaux concepts de la sécurité sont les suivants :

- l'**authentification** qui garantit l'identité d'une personne et de l'origine de l'information. L'authentification est une technique qui est souvent couplée avec un mécanisme d'autorisation qui permet de contrôler l'accès à un système.
- l'**intégrité** qui est la qualité d'une ressource informatique de ne pouvoir être altérée, détruite par accident ou malveillance ;
- la **confidentialité** qui est la qualité d'une ressource informatique de n'être connue que par les personnes autorisées ;
- la **traçabilité** qui est la journalisation et le suivi des événements ;
- la **disponibilité** qui est la qualité d'une ressource informatique à être utilisable à la demande.

A ces concepts de sécurité, il est parfois ajouté la **non-répudiation** qui est une impossibilité de nier un échange pour l'émetteur ou le récepteur du message.

Il est intéressant de noter que l'on retrouve les concepts de sécurité citées sous forme de contre-mesures proposées dans le Tableau 3.1 page 77 pour lutter contre les menaces dans les architectures à services.

L'authentification, l'intégrité et la confidentialité sont trois propriétés essentielles, que Microsoft regroupe sous le nom de « sécurité de transfert » [4]. La sécurité de transfert fournit les techniques permettant de limiter les menaces auxquelles est exposée une application distribuée, comme c'est le cas pour les applications à base de services. Les

4. http://msdn.microsoft.com/fr-fr/library/ms735093.aspx

messages échangés entre les acteurs de l'architecture à services sont un des éléments les plus vulnérables. En effet, si nous prenons l'exemple d'un service de vente en ligne, le numéro de carte bancaire qui est envoyé par le client dans un message sur le réseau doit être absolument confidentiel et ne doit pas être altéré pendant sa transmission. Par la suite, nous concentrerons notre étude sur ces trois propriétés à savoir l'authentification, l'intégrité et la confidentialité ainsi que les mécanismes qui peuvent être mis en place pour les assurer.

2.3 Classification des solutions techniques

Dans la partie précédente, nous nous sommes attaché à définir les concepts de base de la sécurité. Nous présentons, dans cette partie, les techniques qui sont associées à ces concepts. En effet, il existe des solutions techniques pour assurer ses propriétés. Le Tableau 3.3 récapitule, pour chacun des concepts, les buts qui sont atteints avec les solutions techniques standard.

Buts	**Concepts**	**Solutions techniques**
Preuve de l'identité	Authentification	Nom d'utilisateur/Mot de passe Certificat
Contrôle d'accès	Autorisation	Définition des droits d'accès Contrôle du respect des droits d'accès
Protection de l'information	Intégrité	Signature
Protection de l'information	Confidentialité	Chiffrement
Suivi des opérations	Traçabilité	Traces d'exécution

TABLE 3.2 – Mise en relation des concepts et des solutions techniques.

Le but de l'authentification est de prouver au récepteur l'identité de l'émetteur. Pour ce faire, il existe deux méthodes, qui sont l'utilisation d'un mot de passe avec un nom d'utilisateur ou l'utilisation d'un certificat. Ces deux mécanismes consistent à transmettre la preuve de l'identité dans le message. A la réception d'un tel message, le récepteur doit contrôler l'accès à ses ressources, c'est-à-dire gérer les autorisations. La solution est de décrire des politiques de sécurité qui comprennent la définition des droits d'accès et le contrôle du respect de ceux-ci.

La protection de l'information implique deux critères de sécurité : intégrité et confidentialité. Pour s'assurer que l'information n'a pas été altérée ou détruite, il faut signer l'information. Pour garder confidentielle une information, la méthode la plus connue est le chiffrement de l'information.

Pour suivre ce qui se passe dans un système, le moyen le plus simple est de suivre avec une trace les événements qui se produisent. Cependant, selon le système, il faut repérer et tracer les éléments pertinents qui permettent, notamment de savoir s'il y a eu une intrusion dans le système.

La disponibilité et la non-répudiation sont des caractéristiques pour lesquelles les solutions techniques sont plus complexes et dépendent fortement du système à sécuriser.

La section 3 détaille le fonctionnement des techniques de base que sont le chiffrement, le nom d'utilisateur avec mot de passe, la signature électronique, les certificats et la gestion des clés. La section 4 présente l'adaptation de ces techniques de base aux besoins du Web ; ce sont les technologies de sécurité qui existent dans ce domaine. La section 5 est une présentation de la spécification du consortium OASIS faite pour mettre en œuvre ces techniques pour les services Web. Nous présenterons aussi la technologie WSS4J qui est une implantation de cette spécification.

3 Techniques de base de sécurité

Dans cette partie, nous présenterons quatre techniques de base de la sécurité. La première est le chiffrement et le déchiffrement, qui sont une technique du domaine de la cryptographie. La deuxième technique est l'authentification par nom d'utilisateur et mot de passe. Puis, nous détaillerons le fonctionnement de la signature électronique. Finalement, nous terminerons par une présentation du fonctionnement du certificat électronique.

3.1 Chiffrement et déchiffrement

Les techniques de chiffrement et de déchiffrement ont été inventées pour dissimuler le contenu des messages. Déjà à son époque, Jules César employait une méthode de substitution simple [5] avec l'alphabet normal dans les communications avec le gouvernement. Les méthodes de cryptographie ont évolué avec les époques et en fonction de la résistance du chiffrement aux attaques. Cependant, le principe reste toujours le même, comme représenté dans la Figure 3.7.

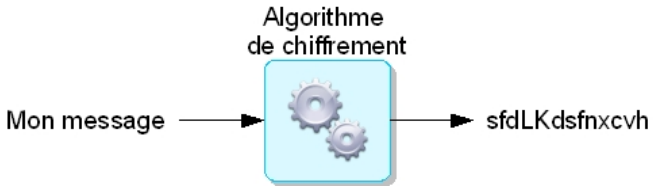


FIGURE 3.7 – Principe de base du chiffrement.

Un message en clair est donné en entrée d'un algorithme de chiffrement plus ou moins complexe. La sortie de l'algorithme est le message chiffré. Ce message chiffré ne peut être déchiffré qu'avec une clé de déchiffrement.

Il existe deux type de chiffrements, comme présentée dans la Figure 3.8 :

– **symétrique** : c'est la même clé qui sert pour le chiffrement et le déchiffrement. Les méthodes les plus connues sont le DES [6], le Triple DES [7] et l'AES [8]

5. Le chiffre de César consiste simplement à décaler les lettres de l'alphabet de quelques crans vers la droite ou la gauche. Suétone, *Vie de César* 56,8 : « il faut substituer à chaque lettre la troisième qui la suit dans l'alphabet : $A \rightarrow D$ »

6. Acronyme de *Data Encryption Standard*, algorithme de chiffrement par bloc utilisant des clés de 56 bits.

7. Algorithme de chiffrement symétrique enchaînant trois applications successives de l'algorithme DES sur le même bloc de données de 64 bits, avec deux ou trois clés DES différentes.

8. Standard de Chiffrement Avancé, en anglais *Advanced Encryption Standard*. Algorithme prenant en entrée un bloc de 128 bits (16 octets), la clé fait 128, 192 ou 256 bits.

– **asymétrique** : ce sont des clés différentes : une clé publique servant à chiffrer et une clé privée servant à déchiffrer. Le point fort de ce type de chiffrement est qu'il est impossible de déduire la clé privée à partir de la clé publique de façon calculatoire. La méthode RSA [9] est la plus répandues pour le chiffrement asymétrique.

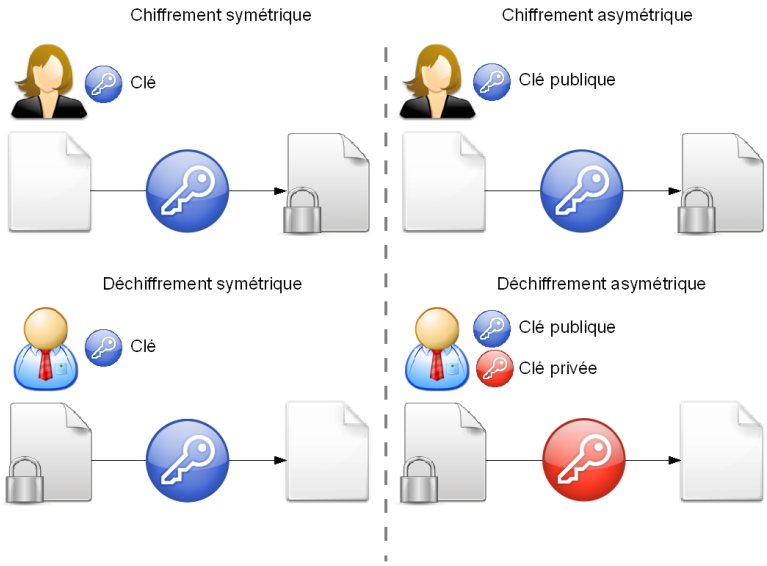


FIGURE 3.8 – Les chiffrements symétrique et asymétrique.

L'utilisation d'un système symétrique ou asymétrique dépend des tâches à accomplir. La cryptographie asymétrique présente deux intérêts majeurs : elle supprime le problème de transmission sécurisée de la clé, et elle permet la signature électronique comme cela est présenté dans la partie 3.3 page 84. Elle ne remplace cependant pas les systèmes symétriques car ses temps de calcul sont nettement plus longs.

3.2 Nom d'utilisateur et mot de passe

L'authentification par mot de passe permet à un utilisateur d'apporter la preuve de l'identité. Elle sert à contrôler l'accès à des données ou à autoriser des actions. Le principe de l'authentification par mot de passe est présenté dans la Figure 3.9.

9. Acronyme de *Rivest Shamir Adleman*.

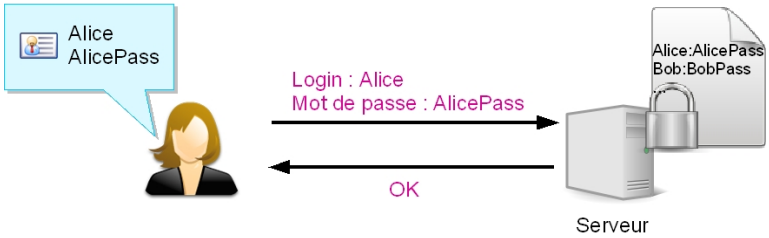


FIGURE 3.9 – Principe de l'authentification par mot de passe.

Le serveur stocke les informations concernant les utilisateurs qu'il autorise à se connecter et les droits associés. Ces informations doivent être suffisamment protégées pour qu'en cas d'attaque du serveur, ces données ne soient pas récupérées et/ou diffusées par une personne malveillante à des fins délictueuses.

Pour s'authentifier, l'utilisateur envoie un message au serveur contenant son nom d'utilisateur et son mot de passe. Le serveur valide ou non ces données et, en fonction de leur conformité, il retourne un message pour signifier que l'authentification s'est bien ou mal passée.

Cette technique permet à une personne malveillante de récupérer les informations nécessaires à l'authentification sur le serveur. Pour éviter que ces informations soient facilement disponibles, la technique du chiffrement du mot de passe est souvent utilisée. Pour encore plus de sécurité, il est possible d'ajouter un intervalle de temps de validité du nom d'utilisateur et du mot de passe, voire d'utiliser un nom d'utilisateur et un mot de passe à usage unique, c'est-à-dire que les éléments d'authentification ont une durée de vie qui ne permet pas à une personne malveillante de les récupérer et de les réutiliser.

3.3 Signature électronique

La signature électronique, aussi appelée signature numérique, a pour but de prouver l'identité de l'auteur d'un document électronique, en utilisant son empreinte. L'empreinte est calculée avec une fonction de hachage, comme MD5 [10] ou SHA-1 [11]. Elle est le résultat de la fonction de hachage sur le document initial, elle dépend donc de son contenu et par conséquent elle est unique pour le document. La signature électronique, en plus de l'empreinte, utilise le chiffrement asymétrique.

Pour signer un document, il faut, dans un premier temps, calculer son empreinte. Ensuite, l'empreinte est chiffrée avec la clé privée de l'auteur du document. La Figure 3.10 illustre le principe de la signature électronique d'un document.

10. MD5, pour *Message Digest 5*, est une fonction de hachage cryptographique.
11. SHA-1, pour *Secure Hash Algorithm* est une fonction de hachage cryptographique.

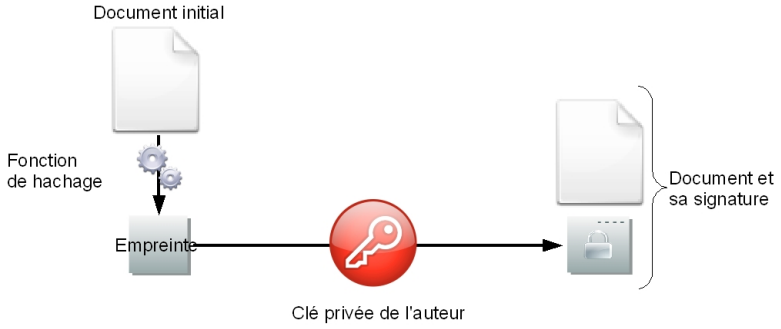


FIGURE 3.10 – Signature d'un document.

Le document doit être vérifié à sa réception par le lecteur. Premièrement avec la clé publique de l'auteur, il déchiffre l'empreinte ; ensuite, il calcule l'empreinte du document sans la signature. Le lecteur compare alors les deux empreintes et si elles sont identiques, la signature est valide. Le processus de vérification de la signature est illustré dans la Figure 3.11.

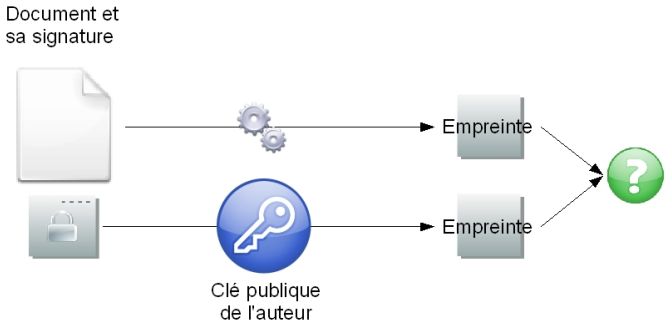


FIGURE 3.11 – Réception du document avec sa signature.

Deux cas peuvent se présenter lors de la vérification des empreintes. Soit les empreintes ne correspondent pas, il est possible que, d'une part, le document ait été modifié depuis sa signature ou, d'autre part, que la clé privée utilisée pour signer ne soit pas associée à la clé publique utilisée pour vérifier la signature. Si les empreintes sont les mêmes, on peut être assuré que le document n'a pas été modifié depuis sa signature et que le couple clé privée/clé publique est correct.

Avec cette technique de signature électronique, il est difficile de copier la signature d'un document pour l'associer à un autre document grâce à l'empreinte. En effet, l'empreinte est unique pour un document et il est facilement détectable qu'il y a eu un vol d'empreinte si elle est apposée à un autre document. La signature électronique garantit l'intégrité d'un document puisque toute modification du document après sa signature rend la signature invalide.

3.4 Certificat électronique

3.4.1 Présentation

Un certificat électronique permet d'identifier une entité physique ou non-physique auprès d'un système. Il peut être vu comme une carte d'identité numérique ; il est aussi parfois appelé certificat numérique. Le standard le plus utilisé pour les certificats numériques est le X.509 version 3, standard de l'Union Internationale des Télécommunications (UIT) [12]. Les champs les plus significatifs des certificats numériques sont :

- un numéro de série,
- un identifiant de l'algorithme de signature,
- le nom de l'autorité de certification émettrice du certificat,
- le nom distinctif du propriétaire du certificat,
- la durée de validité,
- la clé publique du propriétaire,
- la signature de l'autorité de certification.

La Figure 3.12 est un exemple de certificat électronique au format X.509.


```
MIIBkzCB/QIER+OKSDANBgkqhkiG9w0BAQQFADARMQ8wDQYDVQQDEwZjbGllbnQwHhcNMDgwMzIx
MTAxMzI4WhcNMDgwNjE5MTAxMzI4WjARMQ8wDQYDVQQDEwZjbGllbnQwgZ8wDQYJKoZIhvcNAQEB
BQADgY0AMIGJAoGBAITLQaisye3tSjzy3xjg4xxYflGD+YUuDgampOagv45Go9PG7lSHrE9itDdD
42+/Xjd+FkydcIOVroki8wgphEcqBA+/w4H3A2yEmzLnoMvnrjMV48hWPGNwbSOm61+NEWT7CMMg
BUvaan/4ykgdlRCLDe6uvuV3bwTrA5QjDILtAgMBAAEwDQYJKoZIhvcNAQEEBQADgYEALGCp9W3P
QBlggXBqveiL24Qi93T2ykdHgggiw/FxKwTzYHpqdFk8gjB/EufXYgOuSSlNqw0Z93VoMxh3QkO7
kHgNQT6e3I1B8Pf/zMgw06hivi8Z1utBavO05biO318pNpBw6nma6WLlR4w895bOD7Gafgyr/+QE
iEQOwBBqTk8=
```


FIGURE 3.12 – Exemple de certificat X.509.

Un certificat électronique prouve le lien qui existe entre l'identité de son propriétaire et la clé publique qu'il contient. Seule une autorité de certification peut générer un certificat. Mais, pour être sûr qu'un certificat est certifié par une autorité de certification, cette dernière doit le signer. Une autorité de certification possède donc une paire de clés publique et privée. La signature du certificat assure aussi l'intégrité du certificat ; il sera facilement détecté, s'il y a une modification de celui-ci.

Un certificat est donc un document public qui est distribué à tout un chacun, mais seul son propriétaire en a la clé privée. Pour que les utilisateurs des certificats puissent avoir confiance dans le certificat, il existe des infrastructures qui gèrent les certificats.

12. http://www.itu.int/net/home/index-fr.aspx

3.4.2 Infrastructure à Clé Publique

Une Infrastructure à Clé Publique (ICP) est aussi appelée Infrastructure de Gestion de Clés (IGC) et en anglais *Public Key Infrastructure* (PKI). C'est une infrastructure qui gère les clés et les certificats utilisés par des services sécurisés.

Les tâches de gestion qui incombent à l'IGC, sont les suivantes :

- la vérification de l'identité du titulaire lors de l'émission du certificat,
- la publication du certificat,
- le renouvellement du certificat,
- la révocation du certificat,
- la publication de la liste des certificats révoqués.

La Figure 3.13 présente le fonctionnement d'une IGC. L'autorité d'enregistrement vérifie la demande de certificat de l'utilisateur. Ensuite, elle transmet la demande de certificat signée au centre de certification. Le centre de certification crée le certificat, puis le publie dans l'annuaire de publication.

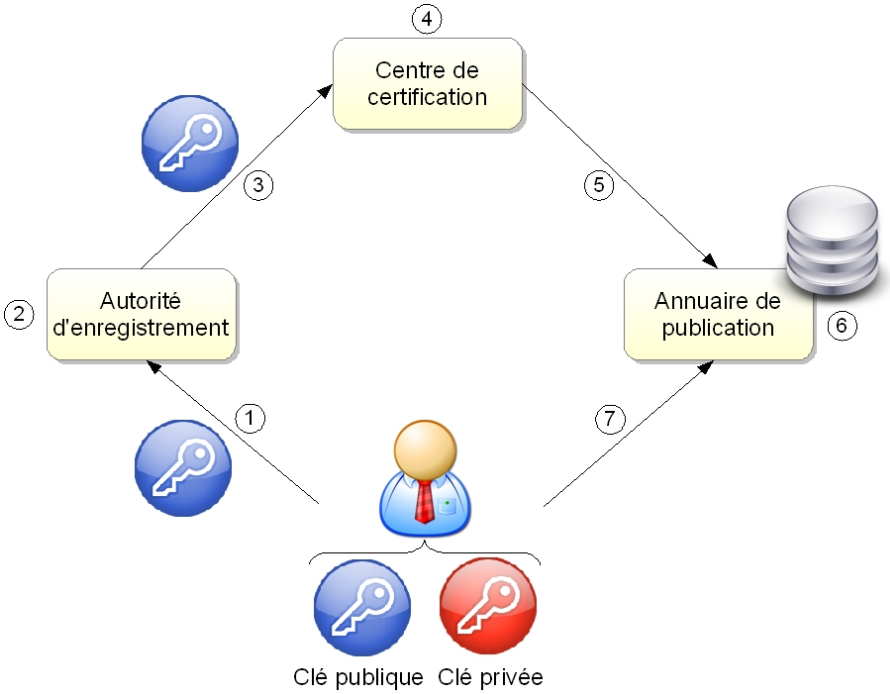


FIGURE 3.13 – Fonctionnement d'une IGC.

3.5 Synthèse

Le Tableau 3.3 récapitule pour chacune des techniques de base présentées précédemment les critères qui sont assurés. Le certificat électronique est une solution assez intéressante pour assurer une grande partie des critères de sécurité.

Techniques	**Concepts/Critères**
Chiffrement/Déchiffrement	– Confidentialité
Nom d'utilisateur/Mot de passe	– Authentification – Autorisation
Signature électronique	– Intégrité
Certificat électronique	– Authentification – Intégrité – Confidentialité – Non-répudiation

TABLE 3.3 – Synthèse des techniques de sécurité pour les différents concepts.

Dans la section suivante, nous allons présenter l'utilisation et l'adaptation de ces techniques aux évolutions technologiques dues à l'Internet et aux services Web.

4 Technologies de sécurité

Dans cette partie, nous détaillons les technologies qui mettent en œuvre les techniques de sécurité précédemment présentées en les adaptant aux nouvelles problématiques posées par la distribution des applications et l'hétérogénéité des technologies d'implantation. Nous présentons en premier XML Digital Signature et XML Encryption qui sont l'application de la signature et du chiffrement au format XML. Ces deux technologies servent de base à la compréhension de XKMS, XACML, XrML et SAML que nous étudierons dans cette partie.

4.1 XML Digital Signature et XML Encryption

XML Digital Signature et XML Encryption sont deux recommandations qui proposent de mettre en application les techniques de base de la sécurité pour des documents numériques en particulier au format XML[13]. Le résultat de la signature et du chiffrement de documents numériques est un document XML.

Le langage XML est un langage de balises qui permet de stocker des données sous forme d'arbre non ordonné. Cette structure arborescente non ordonnée permet d'avoir des équivalences d'écriture avec ce langage. Par exemple, la Figure 3.14 propose cinq possibilités d'écriture d'un noeud avec trois attributs. L'ordre n'est pas pris en considération ni même les espaces et les retours à la ligne multiples. De plus, par raccourci de langage, une balise `<noeud/>` remplace une balise ouvrante suivie d'une balise fermante.

```
<noeud a="1" b="2" c="3"/>

<noeud b="2" c="3" a="1"/>

<noeud c="3" a="1" b="2"></noeud>

<noeud    c="3" a="1"
             b="2"></noeud>

<noeud a="1" b="2" c="3"></noeud>
```

FIGURE 3.14 – Exemple d'écriture d'un noeud en langage XML.

Toutes ses équivalences d'écriture posent des problèmes, en particulier, pour les signatures. En effet, chacune des propositions de la Figure 3.14 a une signature différente des autres alors qu'elles ont le même contenu, seule la forme diffère. Pour remédier à

13. *Extensible Markup Language*, `http://www.w3.org/TR/xml11/`

ce problème, il est possible de normaliser ces écritures avec les recommandations Canonical XML [W3C01a] et/ou Exclusive XML Canonicalization [W3C02a] du W3C. Toutes les expressions de la Figure 3.14 ont pour forme canonique l'expression `<noeud a="1" b="2" c="3"></noeud>`.

4.1.1 XML Digital Signature

XML Digital Signature [W3C08b] est une recommandation proposée conjointement par le W3C et l'IETF [14] pour l'utilisation de signatures électroniques sur les documents numériques, entre autres, les documents XML. Le nom XML Signature est souvent utilisé comme abréviation pour la norme XML Digital Signature.

Le point fort de cette norme est qu'elle propose de signer un document numérique voire seulement une partie. En ne signant qu'une partie d'un document, il est ainsi possible de continuer de modifier le reste du document. Avec cette fonctionnalité, deux participants peuvent donc signer des parties différentes d'un même document. Ceci peut être très utile dans le cas de la composition de services ; un document peut circuler entre différents intervenants et chacun d'eux peut signer sa partie et en assurer la responsabilité. Le résultat d'une signature d'un document numérique est un document XML. Il existe trois types de signature :

détachée : la signature et les données signées font partie de noeuds différents ou sont dans des fichiers séparés ;

enveloppante : la signature contient le document signé ;

enveloppée : la signature fait partie des données signées.

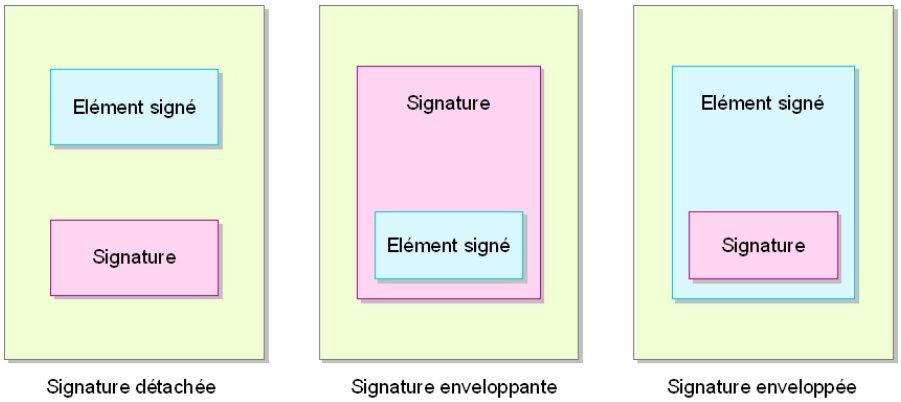


FIGURE 3.15 – Les différents types de signature.

14. *Internet Engineering Task Force*.

Une signature respectant la recommandation XML Digital Signature a la structure donnée dans la Figure 3.16. Le caractère ? signifie aucune ou une occurrence, le caractère + correspond à au moins une occurrence et le caractère * zéro ou plus d'occurrence.

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI? >
      (<Transforms>)?
       <DigestMethod>
       <DigestValue>
     </Reference>)+
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo>)?
  (<Object ID?>)*
</Signature>
```

FIGURE 3.16 – Structure d'une signature avec XML Digital Signature.

La signature numérique comprend les balises suivantes :

***SignedInfo* :** spécifie ce qui est signé et avec quels algorithmes. Les balises *SignatureMethod* et *CanonicalizationMethod* permettent de déterminer quel algorithme d'encodage et quel algorithme de « canonicalisation » sont utilisés pour l'élément *SignatureValue*. Il est possible d'ajouter une liste d'éléments *Reference* qui spécifie les prétraitements à la signature. Dans la balise *Reference*, les éléments *Transforms*, *DigestMethod* et *DigestValue* spécifient quelles sont les transformations, l'algorithme d'encodage et son résultat appliqué sur la ressource de l'*URI*.

***SignatureValue* :** est la valeur de l'encodage de la signature *SignedInfo* avec l'algorithme spécifié par *CanonicalizationMethod*.

***KeyInfo* :** est un élément optionnel qui permet d'autoriser les destinataires à obtenir la clé nécessaire à la validation de la signature.

***Object* :** est un élément optionnel pour contenir les données signées dans le cas d'une signature enveloppée.

La Figure 3.17 est un exemple de signature numérique sur un document XML. La signature est faite avec l'algorithme *SHA1* qui est appliqué aux données à signer.

```
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod
      Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <Reference URI="#23">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>Jy+2klnqh0iE2HeEiLhdPQbYG0U=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>JZR0GRLBvl2SPDi/eAdFcinrnoPd+WhsHt/JjX9ecR39To75qMb/hxcwr
S0oX1GvVzfBEgkpgV4aS+gqSz1nag==</SignatureValue>
  <KeyInfo>
    <KeyValue>
    <RSAKeyValue>
      <Modulus>lWtUkN60V8R3Gd0nB6gvCH2u5TmoziAb7jvLjJR5zY8ChHdo43+iAK0kXaEu
xQrcDynb9GU8SA1r9Zj1IGjyIw==</Modulus>
      <Exponent>AQAB</Exponent>
    </RSAKeyValue>
    </KeyValue>
  </KeyInfo>
/Signature>
```

FIGURE 3.17 – Exemple de signature respectant le standard XML Digital Signature.

La construction d'une signature, respectant la recommandation XML Digital Signature, nécessite un certain nombre d'étapes présentées avec l'Algorithme 1.

Algorithme 1 : Signature d'un document avec XML Digital Signature.

1 **début**
2 Appliquer les transformations nécessaires aux données;
3 Calculer la valeur de l'encodage *DigestValue*;
4 Créer l'élément *Reference*;
5 Créer l'élément *SignedInfo* en fonction des éléments *SignatureMethod*, *CanonicalizationMethod* et *Reference*;
6 Rendre canonique l'élément *SignedInfo*;
7 Calculer l'élément *SignatureValue* ;
8 Assembler les différents éléments de la signature;
9 **fin**

Toutes ces étapes sont réalisées du côté émetteur du message. Pour le récepteur, il faut vérifier que le message est bien valide, c'est-à-dire que le message n'a pas été altéré. Pour ce faire, il faut réaliser les étapes de l'Algorithme 2.

Algorithme 2 : Validation de la signature avec XML Digital Signature.

1 **début**
2 Obtenir les informations de vérification de la clé ;
3 Appliquer l'algorithme *CanonicalizationMethod* sur l'élément *SignedInfo* ;
4 Vérifier la valeur de *SignatureValue* en utilisant la forme canonique de la *SignatureMethod* ;
5 **pour chaque** *élément Reference* **faire**
6 Obtenir les données chiffrées ;
7 Déchiffrer les données en utilisant l'algorithme *DigestMethod* associé ;
8 Comparer la valeur calculée avec la valeur non chiffrée de l'élément *DigestValue* ;
9 **fin**

Une implantation Java de cette recommandation est proposée avec la *JSR 105 XML Digital Signature APIs* [15]. Elle permet de signer des documents XML ou non. Toutefois, elle nécessite une connaissance approfondie du processus de sécurisation d'un document.

XML Digital Signature est une recommandation pour signer des documents numériques, les propriétés de la technique de la signature sont donc respectées : l'intégrité du document, l'identité de l'auteur et la non-répudiation.

4.1.2 XML Encryption

XML Encryption [W3C02b] est une recommandation proposée par le W3C pour chiffrer le contenu ou une partie d'un document XML. Le résultat d'un chiffrement avec XML Encryption est un document XML.

Le résultat du chiffrement avec XML Encryption comprend les balises suivantes :

***EncryptionMethod* :** précise l'algorithme utilisé pour le chiffrement, par exemple DES, Triple DES, AES ou encore RSA.

***KeyInfo* :** est un élément optionnel qui permet d'autoriser les destinataires à obtenir la clé nécessaire au déchiffrement.

***CipherData* :** est un élément obligatoire qui fournit les données cryptées. Il doit soit contenir la séquence d'octets cryptée comme texte codé en base 64 dans l'élément *CipherValue*, soit fournir une référence vers un emplacement extérieur contenant la séquence d'octets cryptée, via l'élément *CipherReference*.

***EncryptionProperties* :** est un élément optionnel qui permet de stocker des informations supplémentaires concernant la génération des éléments *EncryptedData* ou *EncryptedKey*.

Un chiffrement respectant la recommandation XML Encryption a la structure donnée dans la Figure 3.18. Le caractère ? signifie aucune ou une occurrence, le caractère * correspond à zéro ou plus d'occurrence.

15. `http://www.jcp.org/en/jsr/detail?id=105`

```
<EncryptedData Id? Type? MimeType? Encoding?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
  </ds:KeyInfo>?
  <CipherData>
    <CipherValue>?
    <CipherReference URI?>?
  </CipherData>
  <EncryptionProperties>?
</EncryptedData>
```

FIGURE 3.18 – Structure d'un chiffrement avec XML Encryption.

La Figure 3.19 est un exemple de chiffrement avec l'algorithme AES des données XML `<noeud a="1" b="2" c="3"></noeud>`.

```
<xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  Type="http://www.w3.org/2001/04/xmlenc#Element">
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" />
  <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:CipherValue
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      W59qgnFpO3IP3iCqUVwDZnJH7liV9jyvptZODseTZsW71xuLFphjaP1h4F3tbK/rKEWSEJ
Q3YzFDyu54Dqbw/g==
    </xenc:CipherValue>
  </xenc:CipherData>
</xenc:EncryptedData>
```

FIGURE 3.19 – Exemple de chiffrement respectant le standard XML Encryption.

Le chiffrement d'un document ou d'une partie de ses parties en respectant la recommandation XML Encryption nécessite un certain nombre d'opérations présentées avec l'Algorithme 3.

Algorithme 3 : Chiffrement d'un document avec XML Encryption.

1 **début**
2 Sélectionner l'algorithme (et ses paramètres) à utiliser pour crypter les données ;
3 Obtenir la clé et créer l'élément *KeyInfo* si nécessaire ;
4 Chiffrer les données ;
5 Construire l'élément *CipherData* avec soit l'élément *CipherValue* soit l'élément *CipherReference* ;
6 Construire l'élément *EncryptedData* ou *EncryptedKey* ;
7 **fin**

Comme pour la signature, toutes ses opérations sont réalisées du côté de l'émetteur du message. Pour le récepteur, il faut déchiffrer le message pour obtenir le message initial. Le déchiffrement d'un message chiffré avec XML Encryption se fait en suivant les étapes présentées dans l'Algorithme 4.

Algorithme 4 : Validation d'un chiffrement avec XML Encryption.

1 **début**
2 Déterminer l'algorithme de chiffrement et ses paramètres ;
3 Obtenir les informations sur la clé de déchiffrement ;
4 Obtenir les données à déchiffrer en fonction de l'élément *CipherData* : soit l'élément *CipherValue* soit l'élément *CipherReference* ;
5 Déchiffrer l'élément *CipherValue* ou *CipherReference* de l'étape précédente en fonction de l'algorithme de chiffrement, des paramètres et de la clé ;
6 **fin**

XML Encryption est une proposition du W3C pour chiffrer et déchiffrer des documents numériques, en particulier des documents XML. La confidentialité des données est respectée grâce à cette norme. Une implantation Java de cette recommandation est proposée par la *JSR 106 : XML Digital Encryption APIs* [16].

Comme le résultat d'une signature et d'un chiffrement sont des données XML, les deux recommandations XML Digital Signature et XML Encryption peuvent être combinées. Dans le cas d'une signature puis d'un chiffrement, il est ainsi possible d'assurer la protection de la signature et de changer le chiffrement sans modifier la signature. Pour un chiffrement puis une signature, le document ne peut pas être partagé avec d'autres parties sans révéler la clé de déchiffrement et l'identité de l'émetteur est connu. La combinaison des deux recommandations permet donc de respecter les propriétés d'identité de l'auteur, de non-répudiation, d'intégrité et de confidentialité.

16. `http://jcp.org/en/jsr/detail?id=106`

4.2 XKMS

XKMS [W3C05b], acronyme de *XML Key Management Specification*, est une recommandation du W3C basée sur les standards XML Digital Signature et XML Encryption. L'idée générale est de déléguer le traitement de la signature à un serveur sécurisé sur le Web et ainsi remplacer les formats et les protocoles associés aux infrastructures de gestion de clés par des documents XML transportés avec le protocole SOAP.

La recommandation se divise en deux protocoles : X-KRSS pour *XML Key Registration Service Specification* et X-KISS pour *XML Key Information Service Specification*. X-KRSS permet d'enregistrer, de renouveler, de révoquer et de recouvrer une paire de clés. X-KISS est un protocole qui permet de résoudre les informations concernant les clés publiques contenues dans les signatures suivant la recommandation XML Digital Signature, concrètement l'élément `<ds:KeyInfo>`. La Figure 3.20 présente la coordination des différents protocoles pour l'envoi de données signées.

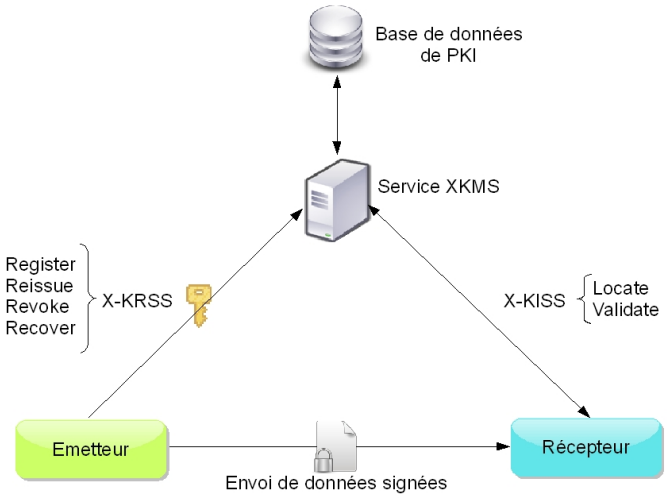


FIGURE 3.20 – Principe de fonctionnement de XKMS.

L'émetteur enregistre sa paire de clés auprès du serveur XKMS et construit un message sécurisé au format XML Digital Signature. A la réception du message, le parseur du récepteur extrait les informations de l'élément `<ds:KeyInfo>` et les fait parvenir au serveur XKMS. Le serveur valide ou non les informations. Le récepteur calcule si la signature est valide. Si la signature et les informations contenues dans l'élément `<ds:KeyInfo>` sont toutes les deux valides, alors le message reçu est considéré comme valide.

Cette recommandation permet de simplifier l'usage des infrastructures de gestion de clés dans les applications. Une implantation Java de cette recommandation est proposée par la *JSR 104 : XML Trust Service APIs* [17].

17. `http://jcp.org/en/jsr/detail?id=104`

4.3 XACML et XrML

Dans cette partie, nous présentons deux technologies XACML et XrML qui permettent respectivement de gérer des politiques d'accès à des ressources et de définir des droits sur des ressources.

4.3.1 XACML

XACML [OAS05b] est l'acronyme anglais de *eXtensible Access Control Markup Language*. XACML est une spécification qui a été définie par le consortium OASIS. Le but de cette spécification est de proposer un langage pour gérer les politiques de contrôle d'accès au format XML.

XACML permet d'exprimer des politiques selon une approche ABAC [18] [WWJ04]. A la différence du modèle RBAC [19] [FK92, SCFY96] qui permet de définir des politiques en fonction des rôles, l'approche ABAC permet de définir des permissions en ne se basant que sur des attributs. Un attribut est une caractéristique, pertinente en terme de sécurité, associée à un sujet, à une action, à une ressource ou encore à un environnement.

XACML comprend également un modèle d'une architecture et une description des modalités de récupération des attributs lors du processus de prise de décision. Ces modalités s'appuient sur un patron d'interactions de type requête/réponse synchrone [LD08].

Pour conclure, XACML est une spécification qui laisse la possibilité d'utiliser les énoncés et les messages SAML [20] [OAS05a]. Cette spécification est de plus en plus utilisée pour assurer les fonctions liées à l'autorisation dans les architectures à services.

4.3.2 XrML

XrML [Con] est l'acronyme de *eXtensible rights Markup Language*. XrML est un langage proposé par ContentGuard, il est basé sur la grammaire XML. Il permet de spécifier des droits et de contrôler l'accès à des ressources comme des services.

XrML permet d'identifier les parties qui peuvent utiliser des ressources, les droits qui sont associés, ainsi que les termes et les droits définis pour l'utilisation. Au cœur de XrML, nous retrouvons d'ailleurs les quatre concepts : *Principal* pour identifier une partie, *Resource* pour la ressource à protéger, *Right* pour les droits et *Condition* pour les conditions d'applications des droits.

XrML est une spécification qui permet de définir clairement les droits, cependant elle doit être utilisée en corrélation avec d'autres technologies de sécurité. Plus spécialement, il existe une extension dans la spécification WS-Security [OAS04b] pour l'utiliser.

18. *Attribute Based Access Control*
19. En anglais, *Role-Based Access Control* pour contrôle d'accès à base de rôles.
20. *Security Assertion Markup Language*, technologie présentée dans la section 4.4.2 page 99.

4.4 Authentification unique et SAML

Dans cette partie, nous définissions la notion d'authentification unique et comment elle est utilisée actuellement. Ensuite, nous présentons le protocole SAML qui propose une solution pour les problèmes liés à l'authentification unique.

4.4.1 Authentification unique

L'authentification unique, aussi appelée en anglais *Single Sign On*, est une méthode pour que l'utilisateur ne s'authentifie qu'une unique fois pour accèder à plusieurs applications informatiques. Cette méthode permet de simplifier pour l'utilisateur la gestion de ses mots de passe. En effet, plus un utilisateur doit gérer de mots de passe, plus il aura tendance à utiliser des mots de passe similaires ou simples à mémoriser, abaissant par la même occasion le niveau de sécurité offert par ces mots de passe. Du côté des applications, cette méthode permet de simplifier la coordination des différentes parties avec une sorte d'annuaire global et de faciliter la définition et la mise en œuvre de politique de sécurité.

Le « passeport » Microsoft, appelé *.NET passport*, est un service d'authentification unique qui permet aux utilisateurs d'accèder à des sites Web avec une unique authentification. Initialement, le passeport a été créé pour le site de courrier électronique Internet Hotmail [21] ; aujourd'hui, il est appelé identifiant *Windows Live ID*. Comme il est maintenant utilisé pour des sites de commerce électronique, il contient de nombreuses informations sensibles comme le numéro de carte bancaire, l'adresse, le numéro de téléphone de son propriétaire. L'inconvénient majeur de ce passeport est qu'il est stocké par Microsoft de façon centralisée. La centralisation des données sensibles posent des problèmes de sécurité et il existe aussi un risque de monopolisation des informations par Microsoft. Il existe d'autres propositions pour l'authentification à identifiant unique comme le *YahooID* de Yahoo ou encore le *Google Account* de Google, qui donne l'accès aux services propres à Google tels que Blogger [22], YouTube [23] et Google Groups [24] en utilisant une adresse mail de type Gmail [25].

En réponse aux problèmes posés par l'authentification unique centralisée, l'authentification unique fédérée a été proposée. Le principe est que chaque service gère une partie des données d'un utilisateur. L'utilisateur peut donc disposer de plusieurs comptes, mais les différents services partagent les informations dont ils disposent sur l'utilisateur avec les services partenaires. Ainsi chaque service partenaire peut conserver la maîtrise de sa propre politique de sécurité. Un exemple de système d'authentification unique fédérée est le système Liberty Alliance [26], aussi connu sous le nom de Project Liberty. Il réunit des

21. `www.hotmail.com/`
22. `http://www.blogger.com/`
23. `http://fr.youtube.com/`
24. `http://groups.google.fr/`
25. `http://www.gmail.com/`
26. `http://www.projectliberty.org/`

acteurs des mondes industriel, informatique, bancaire et gouvernemental sous la forme d'un consortium. L'objectif est de définir des ensembles de spécifications de protocoles de fédération d'identité et de communication entre services Web. Les spécifications initiales du projet Liberty concernaient l'authentification unique fédérée et étaient basées sur la norme SAML [OAS05a]. Ces spécifications ne reposent aujourd'hui plus seulement sur SAML : elles s'appuient également sur un ensemble plus large de protocoles et de normes standards du W3C tels que HTTP [27] et SSL [28].

4.4.2 SAML

SAML [OAS05a] est l'acronyme de *Security Assertion Markup Language*. C'est un protocole proposé par le consortium OASIS. Il en est à sa version 2.0 qui a été approuvée en mars 2005. La première version de SAML s'est basée sur des travaux existants : *Security Services Markup Language* (S2ML) de Netegrity, *AuthXML* de Securant, *XML Trust Assertion Service Specification* (X-TASS) de Verisign et *Information Technology Markup Language* (ITML) de Jamcraker.

Le but de SAML est de proposer une norme pour répondre au problème de l'authentification unique. SAML se base sur les standards HTTP, XML, SOAP, XML Digital Signature et XML Encryption. Le principe de SAML est de représenter l'authentification et les décisions d'autorisation dans un format XML ouvert. Un utilisateur peut ouvrir une session avec un certificat numérique et naviguer ensuite sur des sites Web affiliés sans avoir à s'authentifier à nouveau. Pour ce faire, il est possible d'exprimer avec SAML les informations d'authentification à l'aide de déclarations sur des objets identifiés par le système de sécurité. Les déclarations définissent les propriétés et les contraintes qui s'appliquent sur les objets, ainisi que les droits de ces objets sur les ressources.

SAML propose un format pour les messages XML échangés entre plates-formes. En particulier, il donne un format pour les messages XML basés sur SOAP. Les informations SAML sont intégrées dans le corps du message `Body`.

L'avantage de SAML est qu'il n'est pas un nouveau système d'authentification. Il définit une syntaxe et un vocabulaire pour transmettre les informations qui composent une procédure d'authentification. Une solution concurrente à SAML est proposée par Microsoft, IBM et VeriSign ; elle s'appelle WS-Federation [L+06] et elle est présentée brièvement dans la partie 5.6.2 page 109.

27. Acronyme de *HyperText Transfer Protocol*.
28. Acronyme de *Secure Sockets Layers*.

4.5 Synthèse

Les différentes technologies présentées dans cette section adaptent les techniques de base de la sécurité à savoir la signature, le chiffrement et la gestion des clés et des mots de passe. Ces techniques assurent une partie des concepts de la sécurité qui sont l'authentification et l'autorisation, l'intégrité et la confidentialité. Le Tableau 3.4 récapitule pour chacune des technologies les concepts qui sont assurés.

	Authentification/ Autorisation	**Intégrité**	**Confidentialité**
XML Digital Signature	Identification	OUI	NON
XML Encryption	NON	NON	OUI
XKMS	OUI	NON	NON
XACML et XrML	OUI	NON	NON
Authentification unique	OUI	NON	NON
SAML	OUI	NON	NON

TABLE 3.4 – Récapitulatif des différentes technologies de sécurité.

Nous pouvons constater, d'après le Tableau ci-dessus, que chaque technologie assure, en général, une seule propriété. Il faut donc combiner plusieurs technologies pour pouvoir assurer les trois propriétés. Cependant, nous pouvons noter que chacune des technologies est à elle seule complexe à mettre en place. Il faut avoir des connaissances approfondies en sécurité mais aussi des connaissances du système auquel on applique la sécurité. En effet, la sécurité existe en fonction des caractéristiques métier du système.

5 WS-Security : technologie pour les services Web

5.1 Principes

WS-Security [OAS04b] est une recommandation proposée par le consortium OASIS en 2004. Cette recommandation propose une extension du protocole SOAP [W3C00] pour intégrer des contraintes de sécurité dans les messages échangés avec les services Web. XML Encryption [W3C02b] et XML Digital Signature [W3C08b] sont les bases de la spécifications WS-Security. Ces deux recommandations permettent respectivement de chiffrer et de signer des documents XML. Elles s'appliquent donc à des messages SOAP.

La recommandation WS-Security assure trois propriétés de sécurisation : l'authentification, l'intégrité et la confidentialité. Pour chacune de ces propriétés, elle propose une solution pour intégrer des techniques et/ou des technologies de sécurité existantes dans les messages SOAP. Le Tableau 3.5 ci-après récapitule toutes les techniques supportées par la recommandation WS-Security pour chacune des propriétés.

Propriétés	**Techniques**
Authentification	Nom d'utilisateur et mot de passe crypté ou non
	Certificat X.509
	Jetons Kerberos
	Jeton SAML
	Jetons REL
Intégrité	XML Digital Signature
Confidentialité	XML Encryption

TABLE 3.5 – Propriétés et techniques supportées par la recommandation WS-Security.

WS-Security essaie de regrouper pour l'authentification plusieurs standards : le nom d'utilisateur avec mot de passe, le certificat X.509, les jetons Kerberos [29], SAML ou REL [30]. Une spécification est proposée pour chacune de ses techniques d'authentification :

- [OAS06e] pour le nom d'utilisateur,
- [OAS] pour le certificat X.509,
- [OAS06b] pour les jetons Kerberos,
- [OAS06d] pour les jetons SAML,
- [OAS06c] pour les jetons REL.

29. Protocole d'authentification réseau créé au *Massachusetts Institute of Technology* (MIT), basé sur un système de jetons.
30. *Rights Expression Language*

L'intégrité et la confidentialité sont assurées respectivement par XML Digital Signature et XML Encryption. Ces standards sont quasiment intégrés tels quels dans la recommandation WS-Security.

Dans les trois sections suivantes, nous présentons comment l'authentification par nom d'utilisateur et mot de passe, la signature et le chiffrement avec un certificat X.509 sont intégrés concrètement dans l'en-tête d'un message SOAP. Pour rappel, un message SOAP se compose de deux parties : un en-tête, *header*, optionnel qui permet d'ajouter des extensions ; un corps, *body*, qui contient la méthode à invoquer ainsi que les paramètres si nécessaires.

5.2 Authentification avec WS-Security

L'authentification par nom d'utilisateur et mot de passe est la plus simple à mettre en place. La Figure 3.21 donne la structure de sécurité qui doit être ajoutée dans l'en-tête du message SOAP. Cette structure est composée :

- du nom de l'utilisateur, *Username*,
- du mot de passe, *Password*, et de son type qui a pour valeur chiffré ou non,
- de l'élément, *Nonce*, qui permet au récepteur de détecter les messages qui sont ré-envoyés lors des attaques par tromperie,
- de la date de création, *Created*, qui permet de gérer le temps de validité du nom d'utilisateur.

```
<wsse:UsernameToken wsu:Id="Example-1">
  <wsse:Username> ... </wsse:Username>
  <wsse:Password Type="..."> ... </wsse:Password>
  <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
  <wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

FIGURE 3.21 – Structure de l'authentification par nom d'utilisateur.

La méthode du nom d'utilisateur et du mot de passe est une méthode facile à mettre en place, cependant elle comporte des risques, notamment si le mot de passe est transmis en clair sur le réseau, celui-ci est facilement réutilisable par un pirate. La mise en place des éléments *Nonce* et *Created* permettent de prévenir d'un certain nombre d'attaques. Une autre méthode d'authentification un peu plus sûre est l'authentification par certificat X.509, qui est dérivée de la méthode de signature d'un message par certificat et qui sera présentée dans la section suivante.

5.3 Signature avec WS-Security

La signature permet d'assurer l'intégrité d'un message. La spécification WS-Security se base sur le standard XML Digital Signature pour ajouter la signature dans un message SOAP. La signature est faite avec un certificat X.509 qui est passé dans l'en-tête du message. Le résultat de la signature fait aussi partie de l'en-tête, comme l'illustre la Figure 3.22. La structure de la signature est quasiment identique à celle présentée dans la partie 4.1.1 sur XML Digital Signature, la différence se fait essentiellement au niveau des espaces de nommage.

```
...
<wsse:SecurityTokenReference wsu:Id="Str1">
...
</wsse:SecurityTokenReference>
...
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
  ...
    <ds:Reference URI="#Str1">
      <ds:Transforms>
        <ds:Transform Algorithm="...#STR-Transform">
          <wsse:TransformationParameters>
            <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2001/REC-xml1001c14n-20010315"/>
          </wsse:TransformationParameters>
      </ds:Transform>
      <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue></ds:SignatureValue>
</ds:Signature>
...
```

FIGURE 3.22 – Structure d'une signature dans un message SOAP.

La signature peut s'appliquer à un élément particulier du corps du message SOAP. Dans le cas, de l'intégrité, c'est en général la méthode qui est signée. Pour l'authentification, la solution proposée par la spécification WS-Security est la signature du corps du message au complet.

5.4 Chiffrement avec WS-Security

La spécification WS-Security permet de rendre confidentiel le contenu d'un message SOAP en se basant sur le standard XML Encryption. Grâce à XML Encryption, il est possible de choisir la partie d'un document que l'on souhaite chiffrer, dans notre cas le document XML est un message SOAP et la partie chiffrée est `Body`. Le chiffrement se fait avec un certificat X.509. La Figure 3.23 représente la structure d'un message chiffré.

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..."
 xmlns:ds="..." xmlns:xenc="...">
  <S11:Header>
    <wsse:Security>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#bodyID"/>
      </xenc:ReferenceList>
    </wsse:Security>
  </S11:Header>
  <S11:Body>
    <xenc:EncryptedData Id="bodyID">
      <ds:KeyInfo>
        <ds:KeyName>...</ds:KeyName>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>...</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  </S11:Body>
</S11:Envelope>
```

FIGURE 3.23 – Structure d'un message SOAP chiffré.

Le principe pour obtenir un message SOAP chiffré est donné par l'Algorithme 5.

Algorithme 5 : Construire un message SOAP confidentiel.

1 **début**
2 Créer une enveloppe SOAP ;
3 Créer dans l'en-tête *wsse :Security* ;
4 **pour chaque** *élément de l'enveloppe qui doit être chiffré* **faire**
5 Appliquer l'Algorithme 3 de chiffrement avec XML Encryption ;
6 Ajouter au message toutes les données non-chiffrées ;
7 **fin**

D'après la spécification, il est possible de choisir si l'on souhaite chiffrer tout le contenu (`Body`) du message ou seulement une partie ou plusieurs parties.

5.5 WSS4J : *Web Services Security for Java*

Dans cette partie, nous allons présenter la technologie WSS4J qui permet de sécuriser des messages SOAP en respectant la recommandation WS-Security. Cette technologie repose sur l'échange de messages SOAP entre le client et le fournisseur de services Web réalisé avec la technologie Apache Axis. Nous allons dans un premier temps présenter le fonctionnement de la bibliothèque Apache Axis avant de détailler les fonctionnalités de la bibliothèque Apache WSS4J.

5.5.1 Apache Axis et les services Web

Apache Axis [Apa06a] est une bibliothèque Java du projet *Apache Software Fundation*. Apache Axis est une implantation du protocole SOAP [W3C00], proposé par le W3C. Ce protocole a été présenté dans la section 3.4 page 46.

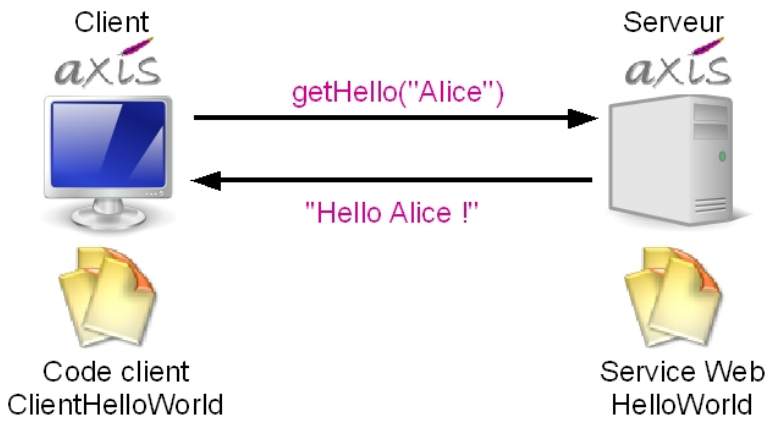


FIGURE 3.24 – Echange de messages avec Apache Axis.

La Figure 3.24 présente l'échange de messages entre un client et un service Web. Les messages échangés respectent le protocole SOAP. La Figure 3.25 est le message d'interrogation du service. Dans la partie `Body`, nous trouvons l'adresse du service appelé, la méthode, le paramètre et son type.

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getHello
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://hello">
      <in0 xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        Alice
      </in0>
    </ns1:getHello>
  </soapenv:Body>
</soapenv:Envelope>
```

FIGURE 3.25 – Requête SOAP d'interrogation du service Web.

La Figure 3.26 est le message de réponse du service Web. Il contient dans la balise `Body` les informations concernant la réponse et son type.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getHelloResponse
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://hello">
      <getHelloReturn xsi:type="soapenc:string"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        Hello Alice !
      </getHelloReturn>
    </ns1:getHelloResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

FIGURE 3.26 – Réponse SOAP du service Web.

5.5.2 Principes de WSS4J

Apache WSS4J [Apa06b], acronyme de *Web Services Security for Java*, est une implantation Apache de la spécification *Web Services Security* (WS-Security) [OAS04b] proposée par le consortium OASIS. C'est une bibliothèque Java qui permet de sécuriser des services Web déployés dans un serveur d'applications.

Cette bibliothèque Java est faite pour ajouter du code de sécurité dans les messages SOAP échangés entre le client et le fournisseur de service. Les messages peuvent être sécurisés avec un nom d'utilisateur et un mot de passe crypté ou non, soit alors avec un certificat. Le certificat est utilisé pour signer les messages ou authentifier l'expéditeur du message. Pour ce faire, Apache WSS4J se base sur l'implantation XML Security[31], qui fournit les standards de sécurité pour XML. XML Security repose sur les standards XML Signature [W3C08b] et XML Encryption [W3C02b] définis par le W3C.

La Figure 3.27 illustre le fonctionnement de la bibliothèque WSS4J. Cette bibliothèque est utilisée du côté client et du côté serveur. En effet, elle permet d'une part de sécuriser les informations à l'envoi, d'autre part de s'assurer de la validité de la sécurité à la réception. La Figure 3.27 détaille le fonctionnement d'un échange de message entre le client et le serveur. Nous prenons l'exemple d'un client qui s'authentifie avec un nom d'utilisateur et un mot de passe pour envoyer une requête au serveur et, en réponse, le serveur envoie un message non sécurisé.

Pour cet exemple, il faut configurer la sécurité côté client et côté serveur. Cette configuration se base sur la configuration classique d'Axis à savoir un client Axis côté client et un service Web avec son fichier de déploiement côté serveur. A ceci s'ajoute un fichier de configuration propre à WSS4J côté client. Les fichiers de configuration contiennent les informations concernant la sécurité, par exemple le nom d'utilisateur et le chemin du fichier contenant le mot de passe.

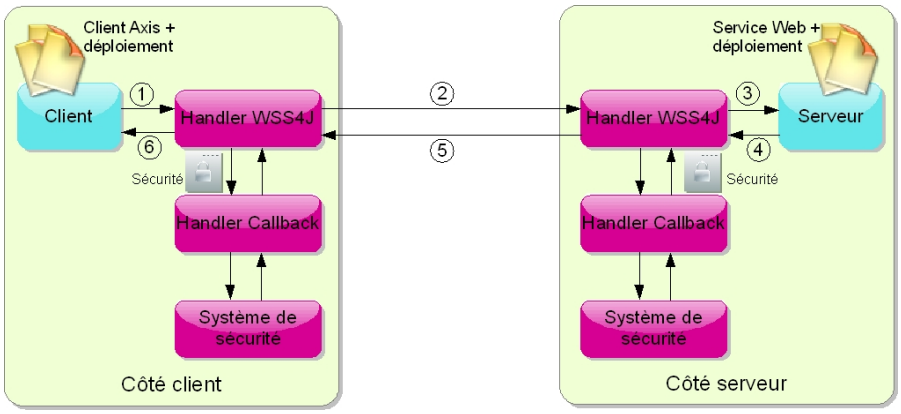


FIGURE 3.27 – Principe de sécurisation avec la bibliothèque WSS4J.

31. http://santuario.apache.org/

Les étapes de l'appel d'un service Web sécurisé, comme présentées dans la Figure 3.27, sont les suivantes :

1. Le client émet un message SOAP standard.
2. Le *handler* WSS4J ajoute les informations de sécurité au message initial, si nécessaire. Dans le cas où il y a une authentification par nom d'utilisateur et mot de passe, ces deux élements sont ajoutés dans l'en-tête du message SOAP. Le message est ensuite envoyé au serveur.
3. Le *handler* WSS4J côté serveur lit les informations de sécurité et valide l'accès au service Web. Dans notre exemple, il faut s'assurer que le nom d'utilisateur et le mot de passe sont valides. Le message SOAP, sans les éléments de sécurité, est finalement transféré au serveur qui n'a plus qu'à répondre au client.
4. Le serveur émet un message SOAP de réponse qui est transféré au *handler* WSS4J.
5. Dans notre cas, le message SOAP de réponse ne nécessite pas de sécurité, donc le *handler* WSS4J n'ajoute pas d'éléments de sécurité au message. Il est envoyé au client.
6. Côté client, le message SOAP est traité comme du côté serveur à l'étape 3. Pour l'exemple, le message est inchangé puisqu'il ne contient pas de sécurité.

Nous pouvons toutefois noter qu'une nouvelle bibliothèque nommée Rampart est proposée par la fondation Apache. Elle propose des fonctionnalités pour supporter les recommandations WS-Security 1.0 et 1.1, WS-Secure Conversation, WS-Policy 1.1 et 1.2 et WS-Trust. Cependant, cette bibliothèque ne fonctionne qu'avec Apache Axis 2.

5.6 Le *framework* WS-Security

Au-dessus de la recommandation WS-Security, de nouvelles recommandations ont été proposées ou sont en cours d'élaboration pour fournir un *framework* de sécurité pour les services Web. La Figure 3.28 représente les différentes recommandations.

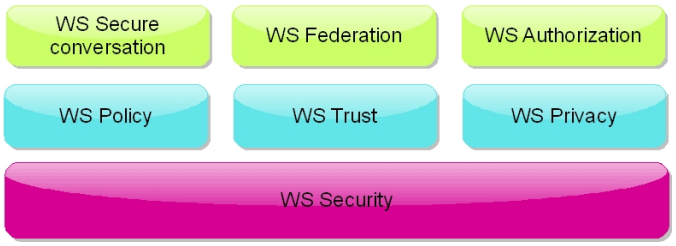


FIGURE 3.28 – Pile de standards.

Les deux sections suivantes présentent chacune des recommandations de la pile. Ces recommandations sont regroupées en deux couches : premier niveau est la couche politique et le deuxième niveau est la couche fédération.

5.6.1 Couche politique

La couche politique se compose de trois recommandations :

WS-Policy [W3C07] : est une recommandation du W3C qui propose de définir un langage pour décrire les contraintes et les règles de sécurité pour tous les participants à une communication avec des services Web.

WS-Trust [OAS07c] : est une recommandation proposée par le consortium OASIS, pour établir des relations de confiance entre différents services Web. Elle permet d'unir des modèles existants de confiance à partir d'un modèle de niveau de confiance, ainsi la validité des jetons de sécurité échangés peut être vérifiée. WS-Trust fournit un processus de communication pour demander la participation de tiers de confiance pour aider les autorités à la vérification.

WS-Privacy : propose un modèle d'expression de règles privées entre un service Web et ses clients. WS-Privacy est à utiliser en conjonction avec les recommandations WS-Policy et WS-Trust.

L'objectif de cette couche est de proposer des moyens pour avoir un niveau de confiance entre services Web. La recommandation WS-Privacy est encore à l'état de projet.

5.6.2 Couche fédération

La couche fédération se compose, elle aussi, de trois recommandations :

WS-SecureConversation [OAS07b] : est une recommandation du consortium OASIS. WS-SecureConversation propose de définir de façon formelle la création et l'échange de contexte de sécurité et des clés de session.

WS-Federation [L+06] : est une recommandation proposée par Microsoft, IBM, VeriSign, entre autres. Elle propose de normaliser la gestion des relations de confiance ainsi que l'identité des utilisateurs (ou des machines) dans un même domaine identifié, c'est-à-dire une fédération.

WS-Authorization : propose un standard pour gérer les informations utilisées pour l'autorisation et les politiques d'accès.

La couche fédération est au-dessus de la couche politique, elle permet d'unifier les différents domaines de confiance. Tout comme la couche politique, une recommandation est encore à l'état de projet : WS-Authorization.

6 Synthèse

Dans ce chapitre, nous avons commencé par mettre en évidence les failles de sécurité qui existent dans l'architecture à services. Les messages échangés sont une cible particulièrement intéressantes pour des personnes malveillantes. Les messages peuvent être écoutés, divulgués et/ou altérés. Des mécanismes de sécurité sont donc nécessaires pour que l'approche à services puisse être couramment utilisée dans les entreprises pour intégrer des applications.

La sécurité est un élément essentiel qu'il faut prendre en considération dès la conception en faisant une analyse des risques auxquels le système est exposé. Des politiques de sécurité permettent de définir des stratégies pour éviter les risques. Des solutions apportées en réponse aux politiques doivent être développées et suivies pour s'assurer que le niveau de sécurité attendu est obtenu. La sécurité est donc un élément qui intervient dans toutes les phases du cycle de vie d'un logiciel.

Nous avons identifié, dans ce chapitre, un ensemble de concepts, de techniques et de technologies qui permettent de répondre aux menaces détectées dans les architectures à services. Les principaux concepts sont l'authentification, la confidentialité et l'intégrité qui permettent de protéger les applications distribuées, telles que celles réalisées suivant l'architecture à services. Pour chacun de ces concepts, il existe des techniques de base qui leur sont associés. Le nom d'utilisateur avec un mot de passe est une technique pour authentifier un utilisateur, le certificat permet aussi son authentification. L'intégrité a pour but de protéger de l'information afin qu'elle ne soit pas altérée en la signant. Il est aussi possible de protéger de l'information en la rendant confidentielle avec des méthodes de chiffrement.

Avec le développement d'Internet et des applications pour le Web, des technologies mettant en œuvre les techniques de base de la sécurité ont été développées. Il existe de nombreuses technologies, mais les recommandations XML Digital Signature et XML Encryption sont deux technologies de référence pour signer et chiffrer des messages XML. Ces technologies ont été adaptées à la technologie des services Web pour signer et chiffrer des messages SOAP. Actuellement, la recommandation WS-Security est la plus avancée pour sécuriser les services Web. Mais, chacune des technologies à services présentées dans le chapitre précédent commencent à développer des standard de sécurité en adaptant les technologies existantes pour le Web. Par ailleurs, nous remarquons que les technologies de sécurité sont adaptées aux technologies à services. Une connaissance des deux domaines est alors nécessaire pour réaliser une application à base de services sécurisés.

Deuxième partie

Contribution

4

Proposition

Dans une première partie, nous allons mettre en évidence la problématique liée à la composition de services en prenant en considération des contraintes de sécurité. Ensuite, nous définirons les objectifs de cette thèse. La deuxième partie sera consacrée à une brève introduction aux notions de l'ingénierie dirigée par les modèles, qui seront utilisées par la suite dans notre approche. Dans une troisième partie, nous présenterons de façon globale notre approche puis nous détaillerons les deux parties qui la constituent : l'une sur la conception de la composition de services et l'autre sur l'exécution de cette composition.

1 Problématique et objectifs

L'état de l'art, que nous avons présenté dans la première partie, nous a permis de faire un certain nombre de constatations qui sont à la base de nos travaux :

- **il existe de nombreuses technologies à services.** Parmi ces technologies à services, rares sont celles qui proposent des environnements de composition de services.
- **la composition de services est complexe.** En particulier, il est nécessaire d'ajouter des opérations de médiation afin de rendre les services compatibles et d'assurer certaines propriétés non-fonctionnelles.
- **la composition de services hétérogènes est encore plus complexe.** Elle nécessite d'avoir une connaissance approfondie de chacune des technologies pour assurer des compositions effectives.
- **l'architecture à services comprend des failles de sécurité.** L'avantage de l'architecture à services est qu'elle permet d'intégrer des services distants en ayant uniquement connaissance de son interface. La distribution ouvre de nombreuses failles de sécurité du niveau physique jusqu'au niveau des messages échangés entre les différents acteurs.
- **il existe de nombreuses techniques et technologies de sécurité.** La sécurité est un vaste domaine de l'informatique ; de nombreuses solutions techniques et technologiques existent pour sécuriser des systèmes. Actuellement, ces techniques n'ont été adaptées qu'à quelques technologies à services.
- **le code de sécurité est très lié au code fonctionnel.** La mise en place d'un système sécurisé se fait toujours sur un système particulier, elle est faite de manière ad hoc. Le code de sécurité est alors ajouté au code fonctionnel du système.

Notre objectif est de simplifier la réalisation d'applications basées sur une orchestration de services sécurisés et hétérogènes. Nous pensons que l'ajout de propriétés de sécurité à une orchestration doit être simple, compréhensible et que le code de sécurité doit pouvoir être généré mais aussi maintenu et changé facilement.

Pour atteindre cet objectif, nous proposons un environnement de développement et d'exécution dans lequel le développeur peut :

1. **définir les spécifications fonctionnelles** de l'application sous forme d'orchestration de services abstraits pour cacher l'hétérogénéité des technologies des services concrets utilisés ;
2. **annoter les services avec des propriétés de sécurité** de façon simple et compréhensible par tous les intervenants lors de la phase de conception de l'application ;
3. **générer le code fonctionnel et celui de sécurité** nécessaires pour l'exécution de l'application.

Avant de présenter l'approche que nous avons mise en place pour répondre aux objectifs fixés, nous allons faire une brève introduction à l'ingénierie dirigée par les modèles, puisque notre proposition s'appuie sur une partie des paradigmes de ce domaine.

2 Introduction à l'Ingénierie Dirigée par les Modèles

L'Ingénierie Dirigée par les Modèles [1] (IDM) est une approche qui est définie ainsi par R. France et B. Rumpe :

> « *The term Model-Driven Engineering (MDE) is typically used to describe software development approaches in which abstract models of software systems are created and systematically transformed to concrete implementations.* » [FR07]

Le but de l'ingénierie dirigée par les modèles est de proposer un cadre pour le développements de logiciels en réduisant l'écart qui existe entre le problème et l'implantation logicielle de sa solution. Cette approche met à disposition un ensemble de technologies qui permettent des raffinements successifs de modèles représentant le système à différents niveaux d'abstraction. Les raffinements sont des transformations qui partent du niveau d'abstraction du problème jusqu'à l'implantation de sa solution. L'avantage de cette approche est qu'elle cache la complexité de l'implantation et qu'elle permet une meilleure communication entre les différents acteurs intervenant tout au long du développement du logiciel.

R. France et B. Rumpe [FR07] présentent une vision plus ambitieuse de l'IDM : les modèles sont utilisés pour le développement du logiciel mais aussi pendant sa phase d'exécution [BBF06]. Les modèles utilisés sont alors des modèles d'exécution. Dans ce cas, ils permettent une surveillance et une gestion du système pendant son exécution.

Dans cette section, nous allons présenter le concept de modèle qui est lié à la relation de représentation, et la relation de conformité qui nous permet de définir la notion de méta-modèle.

2.1 Concept de modèle

La notion de modèle est au centre de l'IDM. Il n'existe pas de consensus pour la définition de modèle, mais nous pouvons nous appuyer sur les quatre définitions suivantes :

> « *Un modèle est une abstraction... une représentation... une simplification d'un système étudié.* » [FEBF06b]

1. En anglais, *Model-Driven Engineering*.

> « *A model is a simplification of a system built with an intended goal in mind.* » [BG01]
>
> « *A model is a description or specification of that system and its environment for some certain purpose.* » [OMG]
>
> « *A model is an abstraction of some aspect of a system. The system described by a model may or may not exist at the time the model is created. Models are created to serve particular purposes, for example, to present a human understandable description of some aspect of a system or to present information in a form that can be mechanically analyzed.* » [FR07]

Ces définitions insistent sur la vocation simplificatrice du modèle qui permet de formaliser des propriétés d'un système en vue d'en faciliter la compréhension ou l'analyse. Un modèle est une représentation du système étudié. Un système peut être décrit par un ou plusieurs modèles, chacun présentant ou représentant un point de vue particulier du système à un certain niveau d'abstraction. Comme un modèle (re)présente un point de vue particulier, il est donc partiel. Notons que la modélisation d'un système peut être faite alors que celui-ci n'existe pas.

Pour faciliter la compréhension et la réalisation des modèles, la représentation sous forme de graphe est souvent utilisée (Figure 4.1 [Kuh06]). Un nœud représente un concept et les liens représentent les relations entre ces concepts. Les cardinalités précisent le nombre d'instances qui participent à une relation. Le diagramme de classes du langage UML[2] [OMG09] est un exemple de modélisation sous forme de graphe.

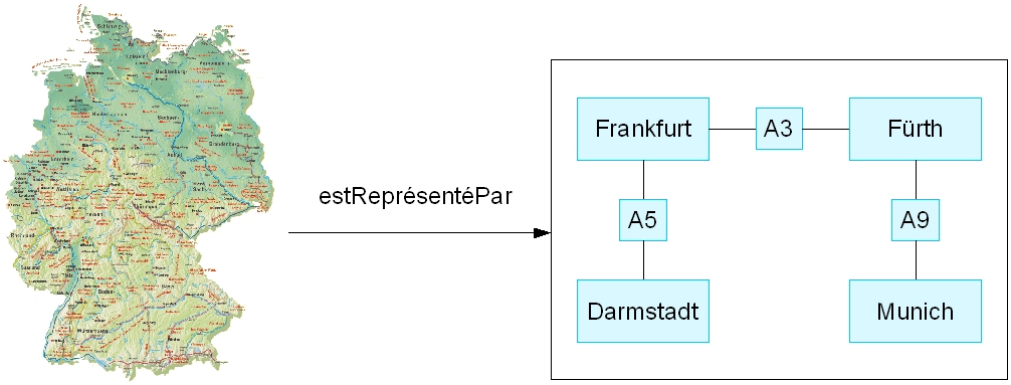


FIGURE 4.1 – Relation de représentation.

J. Ludewig [Lud03] apporte une précision sur la nature de la relation de représentation existante entre le modèle et le système. Il distingue le modèle descriptif du modèle

2. Le langage UML est géré par le consortium OMG (*Object Management Group*).

normatif. Le système descriptif décrit de façon abstraite le système modélisé ou un aspect particulier de ce système, alors que le modèle normatif spécifie certaines caractéristiques attendues du système modélisé. En général, le système normatif permet de définir des spécifications dans le but de réaliser un nouveau système ; le modèle descriptif est, quant à lui, construit à partir d'un système existant.

Les modèles peuvent être distingués en fonction de leur rôle. Une distinction entre modèle contemplatif et modèle productif a été proposée au début des années 2000. Les modèles contemplatifs sont utilisés uniquement pour une meilleure communication entre les différents acteurs. Ces modèles font partie des méthodes de modélisation antérieures à l'IDM comme Booch [BME+07] et OMT [3] [RBP+91]. Les modèles productifs permettent d'implanter des systèmes logiciels après un certain nombre de transformations. Une autre classification des modèles, proposée par S.J. Mellor *et al.* [MSUW04], distingue les modèles en fonction de leur niveau de précision. Un modèle peut être considéré comme un croquis sketch, un plan de travail blueprint ou alors un exécutable executable. M. Fowler *et al.* [FS99] classent aussi en trois catégories les modèles : modèles conceptuels, modèles de spécifications et modèles d'implémentation. Avec cette catégorisation, nous remarquons qu'il existe des modèles adaptés aux différentes phases du cycle de vie d'un logiciel.

2.2 Concept de méta-modèle

Pour que les modèles de l'IDM soient productifs, ils doivent pouvoir être manipulés par des machines. Pour ce faire, ces modèles doivent être formalisés dans un langage de modélisation clair, précis et non-ambigu. Tous les aspects d'un tel langage de modélisation peuvent alors être spécifiés, ce langage devient ainsi un sujet de modélisation. Cette idée introduit la notion de méta-modèle. Nous pouvons nous appuyer sur les deux définitions suivantes pour expliquer le terme de méta-modèle :

> « *A meta-model is a model that defines the language for expressing a model.* » [OMG02]

> « *Un méta-modèle est défini comme le modèle d'un langage de modélisation.* » [Fav04]

Un méta-modèle peut être, comme le modèle, exprimé sous forme de graphe. Dans le cas particulier du langage UML, un méta-modèle est alors représenté par un diagramme de classes.

Les méta-modèles servent à définir une grammaire et un vocabulaire afin de réaliser des modèles conformes et cohérents. Ceci permet d'assurer d'un point de vue théorique mais surtout opérationnel qu'un modèle est correctement construit et donc qu'il est envisageable de lui appliquer des transformations automatisées [FEBF06a]. La notion de

3. En anglais, *Object Modeling Technique*.

conformité (Figure 4.2) peut avoir plusieurs significations. Dans le cadre de notre étude, nous utilisons la conformité par instanciation. Si le méta-modèle contient des éléments instanciables, tels que des relations/liens ou encore des classes/objets, alors un modèle créé par instanciation des éléments du méta-modèle est conforme au méta-modèle. Cela définit une notion de conformité par rapport au méta-modèle.

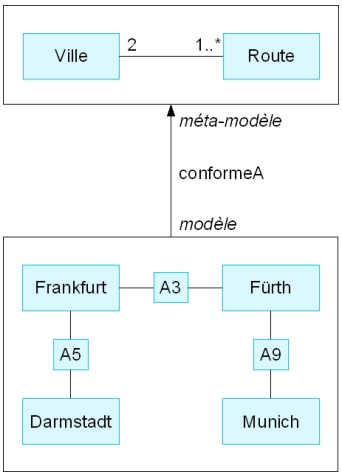


FIGURE 4.2 – Relation de conformité.

2.3 Composition de modèles

Précédemment, nous avons expliqué qu'un système pouvait être représenté par plusieurs modèles qui expriment des points de vue différents. Pour que ces modèles soient productifs, il est donc nécessaire de les composer pour avoir l'ensemble du système. Une composition de modèles est définie ainsi :

> « *Model composition is an operation that combines two or more models into a single one.* » [FBV06]

> « *Model composition in its simplest form refers to the mechanism of combining two models into a new one.* » [HHKR+07]

La composition de modèles peut évidemment se faire au niveau méta-modèle, puisqu'un méta-modèle est un modèle. La composition de méta-modèles est alors le résultat de l'union des méta-modèles et de liens supplémentaires entre des éléments de ces méta-modèles.

3 Notre approche

Dans cette partie, nous allons premièrement présenter d'un point de vue global notre approche. Puis, nous détaillerons chacune des parties, conception et exécution, dans les sections suivantes.

3.1 Présentation générale

Notre objectif est de simplifier la réalisation d'applications basées sur une orchestration de services sécurisés et hétérogènes. Nous souhaitons que la sécurité soit exprimée de façon explicite pour qu'elle soit compréhensible, analysable, évolutive, éventuellement réutilisable et applicable à différents types d'orchestration de services. Pour atteindre notre objectif, nous proposons une approche générative et outillée qui se divise en deux parties, comme illustrée dans la Figure 4.3 :

1. la première phase est la **modélisation** de l'orchestration de services en **séparant** les aspects contrôle et les aspects de sécurité ;
2. la deuxième phase est l'**exécution** du code de l'orchestration en fonction du modèle défini dans la phase précédente et des services disponibles répondant aux spécifications.

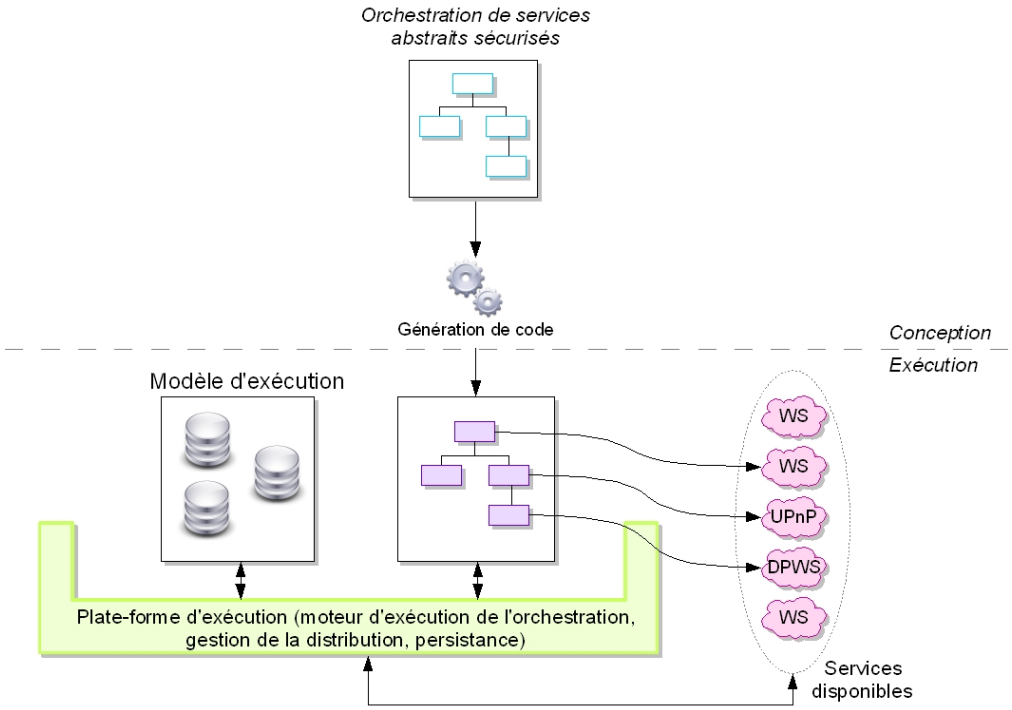


FIGURE 4.3 – Principes de notre approche.

Nous avons mis en application notre approche avec deux outils :

1. un **environnement de modélisation** de l'orchestration dans lequel le contrôle et la sécurité sont exprimés séparément ;
2. une **machine d'exécution** qui permet de gérer des services sécurisés et hétérogènes et l'exécution de l'orchestration abstraite, telle que modélisée par l'outil précédent.

Au niveau conception, nous nous plaçons dans le cadre d'une orchestration de services sécurisés et hétérogènes. Cette orchestration est définie de manière abstraite pour être indépendante des technologies à services et des services effectivement présents à l'exécution. L'orchestration de services abstraits est enrichie par des propriétés de sécurité, qui sont elles-mêmes présentées dans une vue abstraite séparée. La section 3.2 présente en détail le modèle d'orchestration et la solution utilisée pour enrichir ce modèle avec des propriétés de sécurité.

La partie exécution se divise en deux parties : une phase de génération de code générique en fonction de la technologie choisie et une deuxième phase de génération de code spécifique en fonction du service concret sélectionné. La première génération se fait à partir de la spécification de la composition de services sécurisés abstraits et avec des fichiers *templates* prédéfinis pour chacune des technologies à services supportées par la plate-forme. La deuxième étape consiste à compléter le code générique avec les informations spécifiques à l'appel du service concret correspondant aux spécifications. Pour ce faire, le service concret est sélectionné à partir d'un modèle d'exécution. Le modèle d'exécution contient les informations concernant les services concrets disponibles. Mais, il contient aussi des informations concernant les précédentes sélections et un historique des dernières opérations. Avec toutes ces informations, il est possible de choisir le service concret approprié et de compléter le code générique pour le rendre spécifique au service. Le code spécifique permet donc d'appeler le service concret. Le détail de chacun des éléments de cette phase sera développé dans la section 3.3.

3.2 Partie conception

Dans cette partie, nous nous intéressons uniquement au modèle utilisé pour la conception d'une orchestration de services abstraits avec des propriétés de sécurité. La montée en abstraction est la première phase de notre approche pour cacher l'hétérogénéité des technologies à services et la disponibilité évolutive. Nous proposons ensuite de séparer de manière explicite la partie orchestration de services abstraits de la partie sécurité. Puis, nous définissons un langage avec un vocabulaire et une grammaire pour définir l'orchestration de services abstraits et les propriétés de sécurité.

3.2.1 Abstraction

Le principe de la composition de services est de permettre de construire des applications à partir de services existants ou non. Le défi est de pouvoir composer ces services alors qu'ils sont implantés dans des langages différents. L'approche à services propose une solution pour cacher cette hétérogénéité d'implantation en utilisant des interfaces qui présentent les fonctionnalités du service.

Cependant, cette solution n'est pas suffisante pour composer facilement des services. Les descriptions de services dépendent de la technologie à services. En effet, le chapitre 2 page 29 a permis de mettre en évidence qu'il existe de nombreuses technologies à services, telles que les services Web [W3C04b], les services DPWS [Mic06], UPnP [UPn08],... Et, chacune de ces technologies décrit suivant son standard ses interfaces, par exemple avec un fichier WSDL pour les services Web et un fichier WSDL étendu pour les services DPWS. La Figure 4.4 montre les différents niveaux d'abstraction existant pour les services.

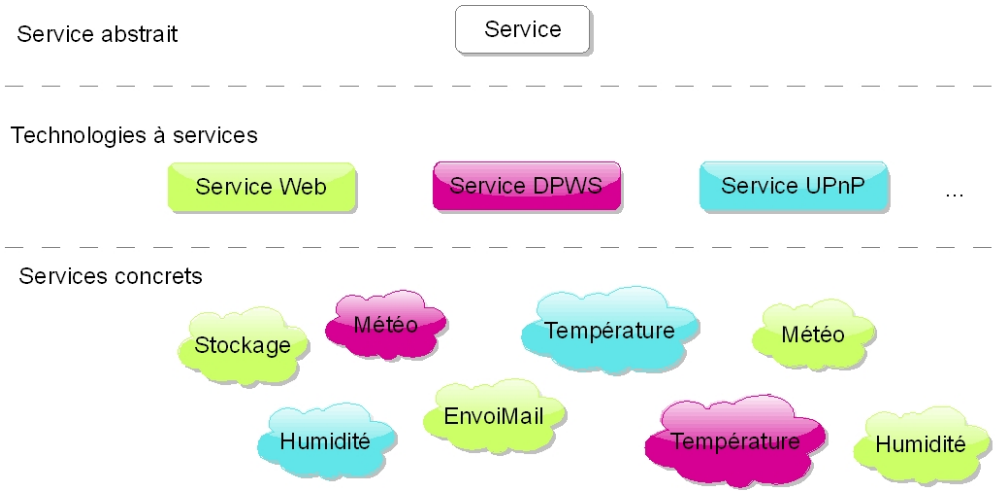


FIGURE 4.4 – Montée en abstraction pour les services.

Pour réaliser la composition de services en tenant compte de l'hétérogénéité des services et des technologies, nous proposons de définir de manière abstraite la composition de services. Cette technique d'abstraction est une solution usuelle de l'ingénierie dirigée par les modèles pour cacher la complexité des techniques et des technologies utilisées, mais aussi pour cacher la réalité de l'environnement à l'exécution. Elle permet de travailler au niveau modèle pour représenter le domaine au lieu d'utiliser directement les objets du domaine. Par exemple, il est plus simple de travailler avec un diagramme de classes UML [OMG09] plutôt qu'avec des objets Java.

Pour ce faire, nous devons définir la notion de service abstrait, en regroupant les caractéristiques communes aux services concrets ainsi qu'aux différentes technologies à services. Notre définition d'un service abstrait comprend les éléments suivants :

- une **interface fonctionnelle** qui spécifie les fonctionnalités fournies par le service concret,
- un **ensemble de propriétés de sécurité** qui décrivent les aspects de sécurité supportés par le service.

Pour notre proposition, nous nous intéressons à l'orchestration de services, qui est une solution pour la composition de services. La montée en abstraction pour les services nous impose aussi de définir l'orchestration de manière abstraite. Il faut définir un modèle d'orchestration abstraite ; ce modèle, Figure 4.5, est détaillé dans les sections suivantes. Par conséquent, pour passer de l'orchestration abstraite à l'orchestration concrète, qui correspond à l'appel des services concrets, il faut passer par plusieurs phases de transformation du modèle jusqu'à l'obtention du code exécutable. Ces étapes sont détaillées dans la section 3.3 page 127 traitant de la partie exécution de l'orchestration.

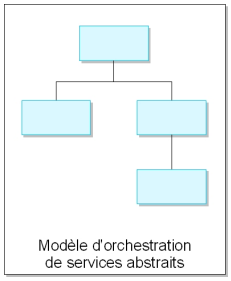


FIGURE 4.5 – Modèle d'orchestration de services abstraits.

L'avantage de cette montée en abstraction est que la communication entre les différents intervenants pour la composition de services est simplifiée. En effet, les détails techniques étant cachés, ils peuvent donc se concentrer sur les fonctionnalités attendues de l'orchestration. Le deuxième avantage est que la composition est définie au niveau modèle. La composition abstraite est donc réutilisable quel que soit le type de services concrets utilisés ainsi que les services.

3.2.2 Séparation des préoccupations

La séparation des préoccupations, en anglais *separation of concerns*, est un terme qui a été en premier introduit par E. W. Dijkstra [Dij74] en 1974 dans le domaine de l'informatique [4] :

> « *It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focusing one's attention upon some aspect" : it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.* »

Cependant, cette idée n'a été reprise et acceptée que quinze ans plus tard ; en 1989, C. Reade [Rea89] présente la séparation des préoccupations ainsi :

> « *The programmer is having to do several things at the same time, namely, 1. describe what is to be computed ; 2. organise the computation sequencing into small steps ; 3. organise memory management during the computation.* »

Le principe de la séparation des préoccupations est de diviser un problème en parties pour en faciliter sa résolution. Il a été repris dans les approches comme l'IDM ou encore l'AOP [5]. Une première difficulté est d'identifier les parties en fonction de leur taille, leur intérêt ou encore de leur point de vue. Chaque partie regroupe une préoccupation qui peut être une fonctionnalité ou un comportement particulier. Le modèle MVC [6] est un exemple de la séparation des préoccupations avec le modèle pour le comportement de l'application, la vue pour l'interface utilisateur et le contrôleur pour la gestion des événements. L'orchestration de services, présentée dans la partie 2.2 page 40, est un autre exemple de la séparation des préoccupations ; chaque service est une préoccupation.

Dans la section précédente, nous avons expliqué que la montée en abstraction permet de cacher les détails techniques et technologiques de la composition de services. Nous avons donc défini une orchestration de services abstraits hétérogènes. Ces derniers sont décrits par des propriétés fonctionnelles et des propriétés de sécurité.

Pour simplifier cette orchestration de services abstraits hétérogènes et sécurisés, nous proposons de séparer les propriétés de sécurité des propriétés fonctionnelles. Nous obtenons alors une décomposition pour l'orchestration de services abstraits sécurisés. Les

4. R. Descartes avait déjà utilisé en 1637 les principes de la séparation des préoccupations dans son *Discours de la méthode* [Des53] : « Le second, de diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait et qu'il serait requis pour mieux les résoudre. »
5. En anglais *Aspect-Oriented Programming*, programmation orientée aspect.
6. En anglais *Model View Controller*, Modèle Vue Contrôleur.

propriétés fonctionnelles sont représentées dans un modèle, c'est l'orchestration de services abstraits hétérogènes. La sécurité est, quant à elle, définie dans un modèle non-fonctionnel. Le modèle de la sécurité est lié au modèle de l'orchestration de services abstraits. La Figure 4.6 illustre le principe de la séparation des modèles.

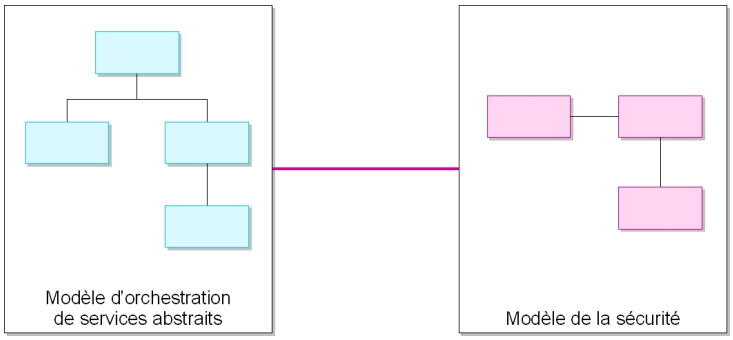


FIGURE 4.6 – Séparation des modèles.

Une telle séparation des modèles permet de rendre modulaire les différents éléments qui constituent la composition de services abstraits. Le modèle de l'orchestration n'est alors pas surchargé. De plus, grâce à cette modularité, chaque élément peut être facilement remplacé. Par exemple, il est possible de choisir un modèle d'orchestration APEL [DEA98] ou bien WS-BPEL [OAS07a] comme modèle d'orchestration. Le modèle de la sécurité peut alors rester inchangé mais il peut s'adapter grâce à ses liens avec le modèle d'orchestration (APEL ou WS-BPEL par exemple). A l'inverse, il est possible de changer le modèle de la sécurité en gardant un modèle d'orchestration. De cette propriété découle le principe de réutilisabilité des différents modèles.

Cette séparation a pour autre avantage de faciliter la maintenance. En effet, un modèle peut évoluer avec un impact nul ou contrôlé sur l'autre modèle. Le modèle maintenu est alors modifié et son (ses) lien(s) avec l'autre modèle peu(ven)t alors être changé(s).

La séparation des préoccupations a pour conséquence de faire intervenir de nombreux acteurs. Chaque modèle est réalisé par un expert de son domaine. Ainsi, les connaissances approfondies dans un domaine de chaque intervenant permettent de réaliser une meilleure composition de services abstraits. Les communications entre les différents intervenants sont par ailleurs facilitées.

Après avoir divisé un problème en parties pour en faciliter sa résolution, il faut recomposer les parties pour avoir une solution globale. C'est là que se trouve la deuxième difficulté : comment les parties doivent-elles être assemblées ? Une solution proposée par l'AOP est le tissage (*weaving*) des différents aspects, dans notre cas l'orchestration et la sécurité.

3.2.3 Composition de méta-modèles

Pour s'assurer de la cohérence et de la validité du modèle, nous montons d'un niveau d'abstraction, c'est-à-dire que nous passons au niveau méta-modèle. Dans la partie précédente, nous avons présenté notre modèle, qui se compose d'un modèle fonctionnel et d'un modèle de la sécurité. Le modèle de la sécurité est lié au modèle fonctionnel. Nous proposons de réaliser un méta-modèle pour l'orchestration de services et un méta-modèle de la sécurité. La Figure 4.7 représente cette montée en abstraction. Au niveau méta-modèle, nous obtenons une composition de méta-modèles. Le méta-modèle de la sécurité est lié au méta-modèle de l'orchestration par des liens.

Chaque méta-modèle contient les concepts de son domaine, il définit un langage pour que les modèles associés soient interprétés de façon non-ambiguë. Le méta-modèle fonctionnel contient les concepts associés à l'orchestration comme la notion de service, ou encore les entrées et les sorties requises pour un service. Le méta-modèle de la sécurité contient, par exemple, les concepts d'authentification, d'intégrité ou encore de confidentialité. Les liens définissent les dépendances qui existent entre les modèles au niveau modèle.

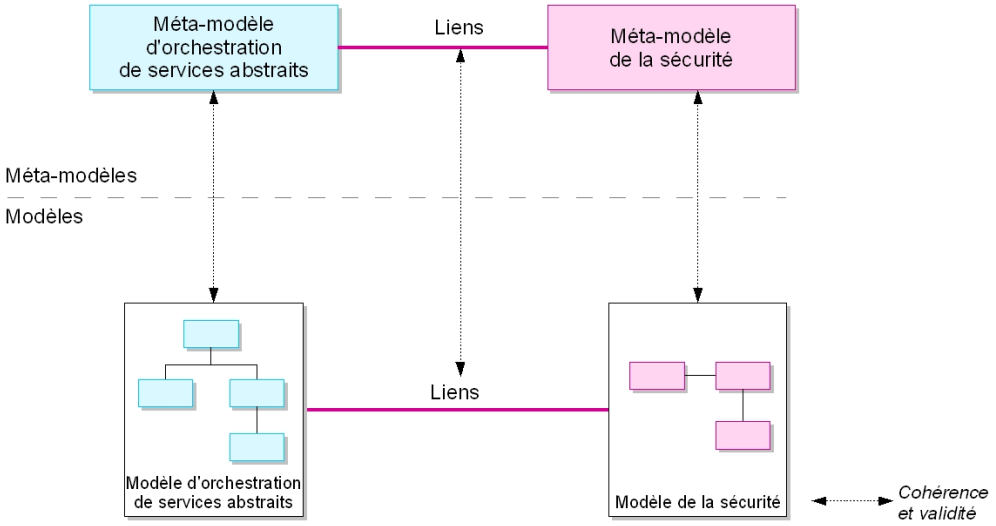


FIGURE 4.7 – Composition des méta-modèles.

Cette montée en abstraction nous permet par conséquent de nous assurer d'une cohérence entre les liens et les modèles ainsi que de définir les concepts qui peuvent être utilisés au niveau modèle. Lors de la création de la composition de services, l'utilisateur peut ainsi être guidé dans ses choix grâce à la grammaire définie par le méta-modèle complet de la composition de services.

Un méta-modèle est défini par un expert métier du domaine. Par sa connaissance de

son domaine, il est capable d'identifier quels sont les concepts et les liens entre ceux-ci. Les liens sont, quant à eux, définis par tous les experts. Cette approche permet donc une meilleure communication entre les différents experts.

3.2.4 Conclusion de la conception

Nous synthétisons les différents éléments de la partie conception dans la Figure 4.8. Nous apportons :

1. une **modélisation des services et de l'orchestration** qui permet de cacher l'hétérogénéité des technologies et des techniques à services.
2. une **modélisation de la sécurité** de façon séparée du modèle d'orchestration. Le modèle de sécurité permet d'annoter le modèle d'orchestration.
3. des **méta-modèles pour l'orchestration et la sécurité** mis en relation avec des liens. Nous obtenons ainsi un moyen d'assurer la cohérence du modèle global.

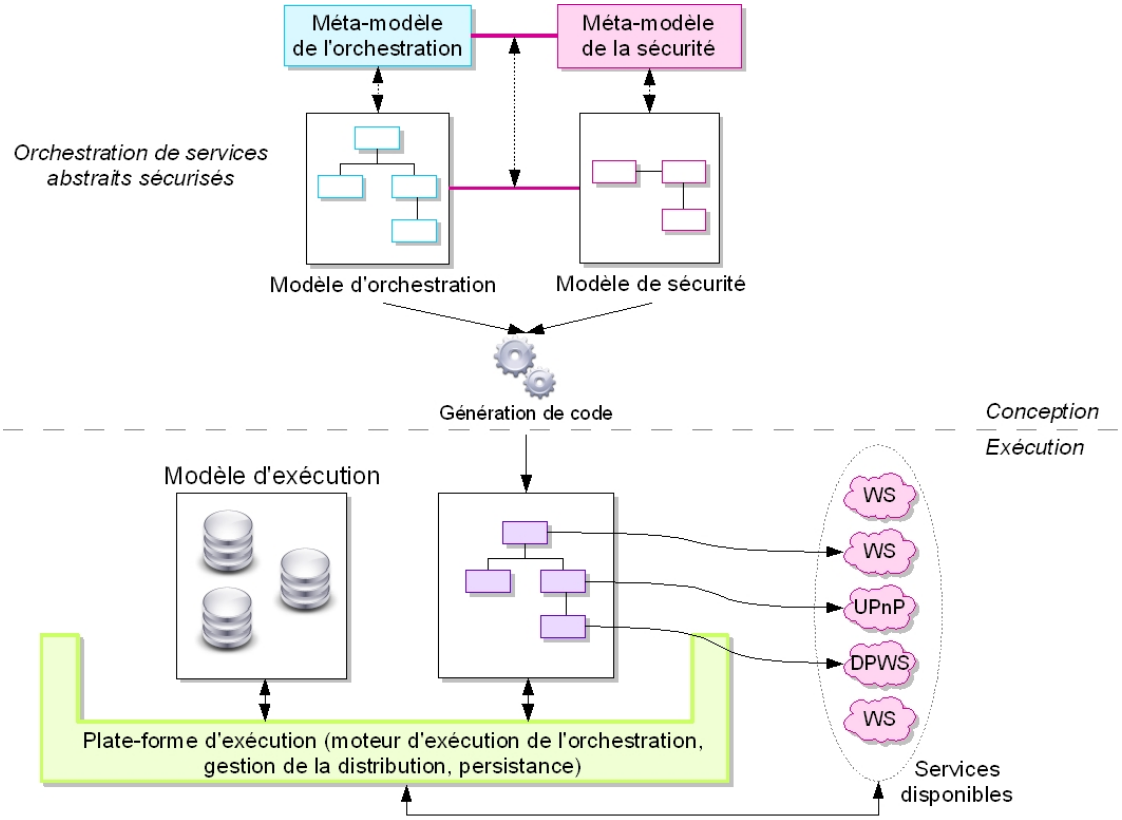


FIGURE 4.8 – Synthèse de la partie conception.

3.3 Partie exécution

Après la phase de conception de l'orchestration de services abstraits hétérogènes et sécurisés, nous passons à la phase d'exécution en fonction des services disponibles répondant aux spécifications du modèle de l'orchestration. Dans une première partie, nous allons présenter le modèle d'exécution, puis le mécanisme de sélection requis pour obtenir un service approprié. La dernière partie concerne la génération du code qui intervient à deux moments dans notre approche : à la fin de la conception et après la sélection du service concret.

3.3.1 Le modèle d'exécution

Le but du modèle d'exécution est de pouvoir maintenir l'état de l'environnement d'exécution ; il a une connaissance des services disponibles et de leurs propriétés concrètes. Notre approche est mise en application dans un Intranet pour simplifier la gestion de la sécurité mais aussi des services disponibles. L'ajout du modèle d'exécution dans notre architecture permet de garder la propriété de liaison à retardement de l'approche à services. Cette propriété est très importante puisqu'elle permet de s'adapter au dynamisme des services. En effet, les services peuvent être indisponibles pour cause de pannes, d'arrêt ou encore pour avoir évolué ; tous ces problèmes peuvent être en partie évités si le client choisit son service juste avant de l'appeler. Evidemment, même si la sélection est faite juste avant l'appel du service, il se peut que le service soit quand même indisponible, et il est donc nécessaire de gérer l'erreur.

Le modèle d'exécution prend le rôle de l'annuaire, qui est un des acteurs de l'architecture à services. Les services disponibles s'enregistrent auprès de cet annuaire et les clients recherchent le service approprié en fonction de leurs besoins. Dans notre cas, nous créons un client par service abstrait de l'orchestration. La durée d'exécution d'un service ne pouvant être connue à l'avance, il est important d'avoir un modèle d'exécution maintenu à jour en temps réel pour que l'orchestration puisse être exécutée correctement.

Néanmoins, le modèle d'exécution est plus qu'un annuaire ; il comprend des informations concernant les services concrets disponibles mais aussi un historique des invocations antérieures. La définition de l'orchestration abstraite de services est réalisée en définissant évidemment le contrôle mais aussi le type des services qui seront interrogés. L'architecte a une connaissance métier de l'application qui sera réalisée par une orchestration. Par exemple, si un architecte veut faire une remontée de données de capteurs pour une application domotique, il sait que les technologies à services les plus appropriées sont UPnP et DPWS. Par conséquent, la sélection des services devra prendre en considération cette information qui doit aussi être présente dans le modèle d'exécution. La structure nécessaire pour supporter plusieurs technologies à services est d'avoir un annuaire par technologie. Le modèle d'exécution est alors un ensemble d'annuaires évolués. Le contenu de ces annuaires est explicité dans la section suivante qui explique le mécanisme de sélection en fonction des spécifications et des services disponibles.

3.3.2 La sélection de services concrets

Pour passer d'une spécification de l'orchestration abstraite à l'exécution, il faut trouver des services concrets qui correspondent aux services abstraits ; c'est la phase de sélection des services. Le principe est que, pour chacun des services abstraits, il faut sélectionner le « bon » service ; en effet, c'est un service concret ayant les caractéristiques suivantes :

- il doit être **disponible**, c'est-à-dire présent et en fonctionnement ;
- il doit être dans la **technologie à services** requise ;
- il doit **assurer exactement les fonctionnalités** définies dans la spécification ;
- et il doit aussi **respecter les contraintes de sécurité** définies dans la spécification.

Pour trouver le « bon » service, il faut utiliser le modèle d'exécution qui comprend toutes les informations concernant les services disponibles. Ces informations doivent pouvoir répondre aux spécifications, il faut donc avoir le type de la technologie ainsi que les caractéristiques fonctionnelles et non-fonctionnelles supportées par le service. Le modèle d'exécution est donc composé, comme pour la partie spécification, de deux parties : une partie fonctionnelle et une partie sécurité. La Figure 4.9 illustre cette composition des deux modèles. La partie fonctionnelle du modèle d'exécution doit comprendre au moins le nom du service concret et son adresse physique pour pouvoir concrètement l'interroger. La partie sécurité doit, quant à elle, présenter les propriétés de sécurité de telle sorte qu'un client puisse rechercher facilement le service approprié.

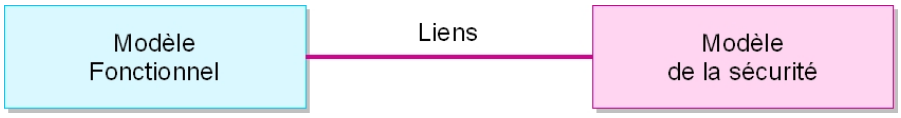


FIGURE 4.9 – Modèle d'exécution composé de deux modèles.

Dans la recherche du « bon » service, il est important de savoir définir ce qu'est un « bon » service, voire le « meilleur » service, qui réponde aux spécifications. En effet, plusieurs choix peuvent être possibles : plusieurs services concrets peuvent répondre exactement ou partiellement aux spécifications. Pour faire un choix, il est nécessaire que toutes les fonctionnalités soient exactement réalisées. Par contre, pour les propriétés de sécurité, il faut au moins que celles définies lors de la spécification soient supportées. Cependant, il est possible que le service supporte des propriétés supplémentaires.

Pour choisir le « meilleur » parmi les « bons » services, d'autres critères peuvent entrer en jeu comme le temps de recherche, la qualité du service et sa réputation, son temps de réponse, son coût,... Par exemple, dans le cas du temps de recherche, le « bon » service correspond au premier service trouvé qui remplit exactement les critères. Ces informations ne sont pas toutes des informations données par le service concret. Elles proviennent de l'expérience obtenue après des utilisations précédentes des services.

Pour conclure, le service qui est sélectionné doit répondre aux critères suivants dans cet ordre :

1. le type du service doit être conforme à celui de la spécification ;
2. les caractéristiques fonctionnelles doivent être conformes à celles de la spécification ;
3. les caractéristiques de sécurité requises doivent être remplies ;
4. les autres propriétés non-fonctionnelles doivent être minimisées ;
5. les propriétés de qualité de service doivent être remplies au mieux ;
6. le service choisi peut être préféré aux autres parce qu'il a déjà été utilisé valablement.

3.3.3 La génération de code

La génération du code est une phase qui se déroule une fois que l'orchestration de services abstraits a été spécifiée et avant son exécution. Le but de la génération est de rendre exécutable l'orchestration en faisant le lien entre la spécification abstraite et les services concrets sélectionnés.

La génération se décompose en deux étapes, comme l'illustre la Figure 4.10 :

1. la **génération de fichiers** pour chacune des activités à partir de fichiers *templates* prédéfinis ;
2. la **spécialisation des fichiers générés** en les paramétrant en fonction des données récupérées lors de la sélection.

Les fichiers *templates* sont des fichiers prédéfinis à l'avance en fonction des technologies à services supportées. Ils correspondent aux fichiers client d'un service. Pour chacune des activités, il faut générer les fichiers client. Cependant, au moment de la génération du code, les services concrets ne sont pas encore sélectionnés ; les fichiers client ne sont donc pas complets.

La sélection permet à partir des propriétés fonctionnelles et non-fonctionnelles de sélectionner le service le plus approprié aux spécifications. La sélection retourne les informations concrètes nécessaires à l'appel du service, comme par exemple son adresse physique. Ces informations sont utilisées comme paramètre pour spécialiser les fichiers incomplets à l'appel d'un service concret particulier.

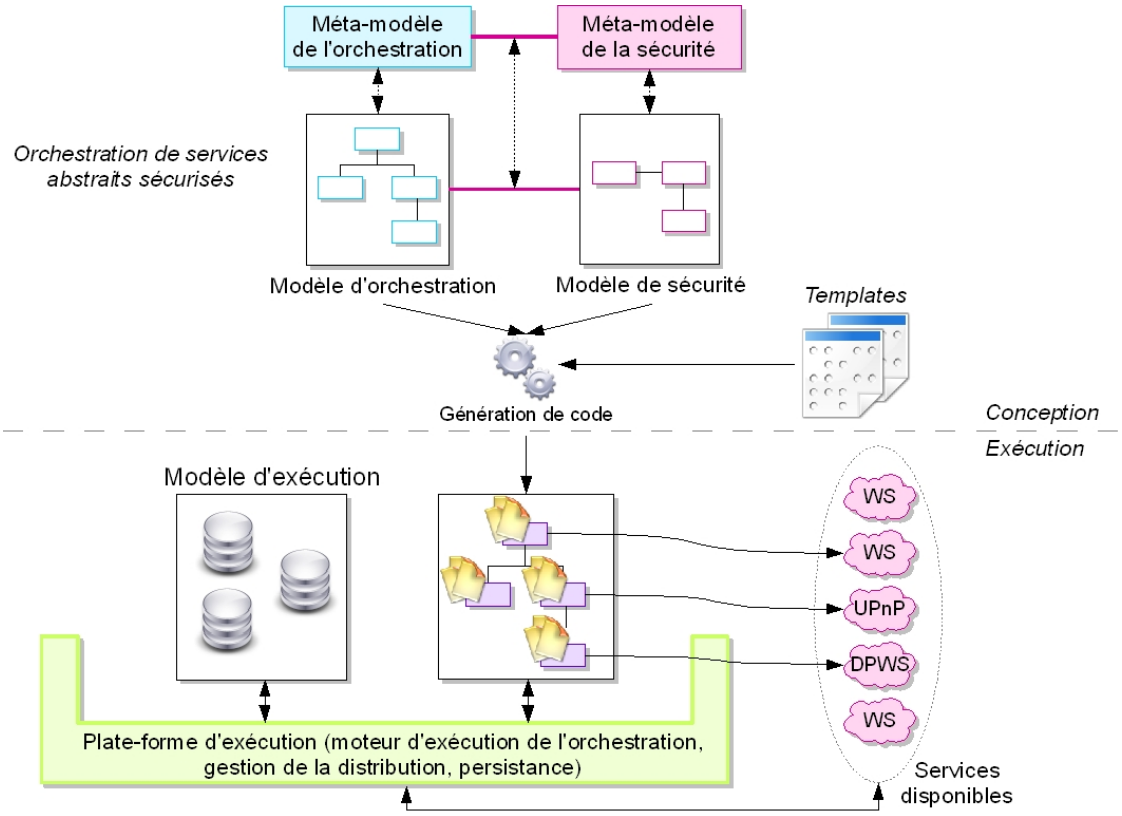


FIGURE 4.10 – Génération de code en deux phases.

3.3.4 Conclusion de l'exécution

Nous pouvons récapituler les apports de la phase d'exécution de notre approche ainsi :

1. une **modélisation** des services concrets disponibles en fonction de leur type. Le modèle comprend également les fonctionnalités fournies par les services ainsi que leurs caractéristiques de sécurité.
2. un **mécanisme de sélection** qui permet de choisir le service concret le plus approprié aux spécifications grâce au contenu du modèle d'exécution.
3. un **processus de génération** de code en deux étapes : la première étape permet de générer des fichiers génériques en fonction des technologies à services choisies ; la deuxième étape permet de spécialiser ces fichiers avec les informations concrètes nécessaires à l'appel du service.

L'ajout de ces éléments a été fait dans le but de gérer le dynamisme de l'environnement d'exécution, plus précisément, celui des services.

5

Méta-modèles et modèles

Dans le chapitre précédent, nous avons présenté notre approche pour la réalisation d'applications basées sur une orchestration de services hétérogènes et sécurisés. Pour mettre en place notre approche, nous avons proposé de définir deux méta-modèles liés entre eux : l'un pour l'orchestration de services hétérogènes et l'autre pour la sécurité. Ces méta-modèles permettent de définir les concepts qui sont utilisables au niveau modèle pour spécifier, de manière abstraite, l'orchestration.

Ce chapitre a pour but de présenter la mise en application de notre approche en présentant les différents éléments de la conception nécessaire à la génération du code exécutable. La Figure 5.1 récapitule les différents éléments de notre approche. Dans les deux premières parties, nous présentons les méta-modèles : d'une part, celui de l'orchestration de services hétérogènes et, d'autre part, celui de la sécurité. La partie suivante est consacrée aux liens qui nous ajoutons entre ces deux méta-modèles. Dans la quatrième section, nous passons au niveau modèle avec l'expression des orchestrations de services sécurisés. Puis, nous expliquons le code qui doit être généré à partir des modèles.

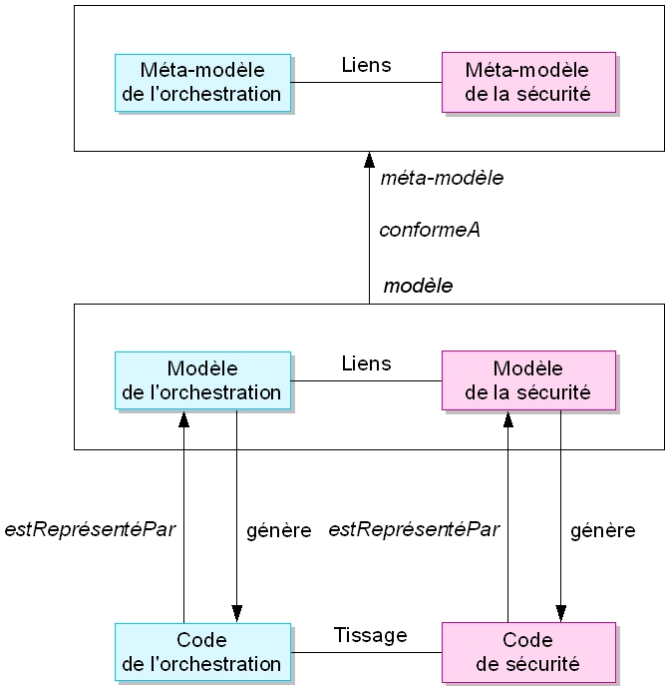


FIGURE 5.1 – Mise en application de notre approche.

1 Méta-modèle de l'orchestration de services hétérogènes

La Figure 5.2 présente le diagramme de classes de notre méta-modèle de l'orchestration de services abstraits hétérogènes. Ce méta-modèle étend le méta-modèle du langage APEL [1] [DEA98]. APEL est un langage qui a été développé dans l'équipe de recherche ADELE et qui permet d'exprimer des procédés logiciels [2].

Le concept d'*Activité* est l'élément central du langage APEL qui comprend les concepts suivants :

- une *Activité* est une tâche qui peut être réalisée pendant l'exécution du procédé. Une *Activité* peut être composée de sous-activités, elle est alors dite composite.
- un *Port* est une interface de communication pour une *Activité*.
- un *Produit* est une entité produite, transformée et/ou consommée par les activités.
- un *Type de produit* permet d'exprimer sous la forme d'une chaîne de caractères le type de chacun des produits qui circulent entre les activités.

1. *Abstract Process Engine Language*
2. Les procédés sont définis comme étant une suite d'étapes réalisées dans un but donné : « *a sequence of steps performed for a given purpose* » [Ei90]. Les procédés logiciels (*Software Process*) servent à gérer et assister le développement logiciel [Jam05].

– un *Flux de données* exprime de façon explicite la circulation des données entre les activités et il est possible d'en déduire le flux de contrôle entre les activités.

L'avantage du langage APEL est qu'il est structuré à partir d'un nombre minimal de concepts, ce qui facilite son utilisation. Cependant, la nature des activités, comme manuelles ou automatisées, n'est pas précisée. Le langage APEL peut être étendu pour répondre à des problématiques particulières comme c'est le cas dans notre approche. Une activité peut représenter une activité humaine ou alors un service abstrait.

La Figure 5.2 est composée à gauche des concepts du langage APEL. A ces concepts, nous avons ajouté le concept de *Service abstrait* qui représente toute activité automatisée, et le concept d'*Activité humaine*. Ces deux concepts sont liés au concept d'*Activité*. Une *Activité* peut être réalisée par un *Service abstrait* ou par un ensemble de *Services abstraits*, dans ce cas on parle de service **composite**. Plus précisément, un *Service abstrait* est défini par une *signature* qui correspond :

– au **nom de la méthode** qui réalise l'activité,

– et aux **paramètres d'entrée et de sortie** nécessaires à la méthode.

Ces informations concernant la signature ne sont pas exprimées de la même manière dans chacune des technologies à services (Figure 5.3), c'est pourquoi nous avons défini quatre spécialisations pour la définition d'un service abstrait. Le code Java est une technique particulière pour réaliser un service abstrait ; une méthode Java a aussi des paramètres d'entrée et de sortie.

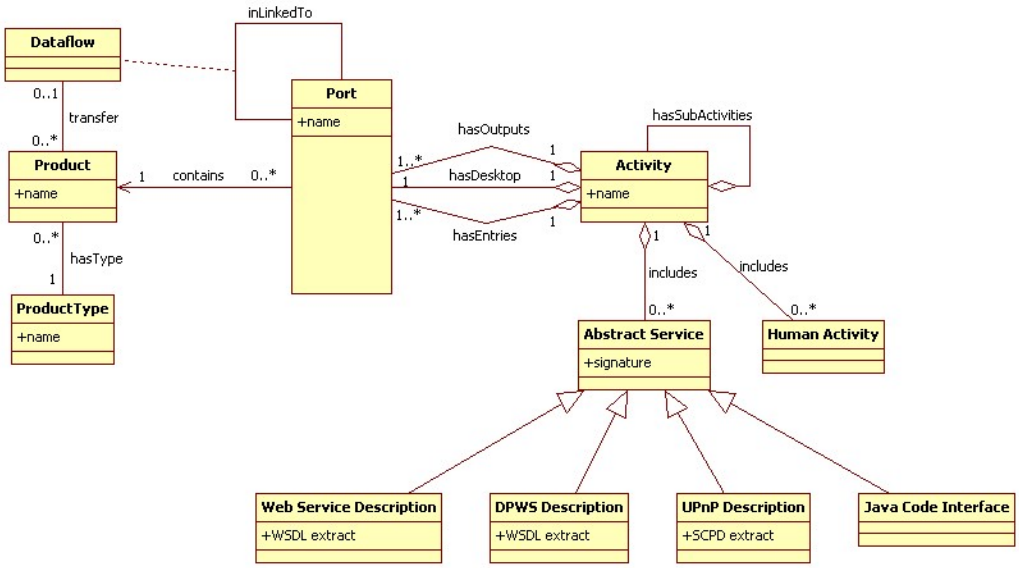


FIGURE 5.2 – Méta-modèle de l'orchestration de services abstraits hétérogènes.

De plus, la définition d'une orchestration de services abstraits se fait en fonction des technologies que l'architecte souhaite utiliser. L'architecte qui définit une orchestration sait quelle technologie est la plus appropriée pour récupérer, par exemple, des données d'un capteur. Les technologies UPnP et DPWS sont les technologies à services les plus utilisées dans ce domaine. Alors que pour faire une analyse de données, il semble plus pertinent d'utiliser un service Web. Nous ne cachons donc pas l'hétérogénéité des technologies à services mais nous cachons l'hétérogénéité des services concrets disponibles.

La définition d'un service abstrait permet à l'application de s'adapter au dynamisme de l'environnement à l'exécution. En effet, un service peut être indisponible ou disparaître à n'importe quel moment pour différentes raisons que ce soit un problème technique, une mise à jour ou encore un arrêt définitif. Mais, en ayant spécifié à un haut niveau les fonctionnalités attendues, il est possible de choisir un autre service concret disponible qui remplisse ces fonctionnalités au moment où l'on souhaite l'utiliser, c'est un mécanisme de liaison à retardement.

Service Web	`<wsdl:portType name="nmtoken">*`
	`  <wsdl:documentation .... />?`
	`  <wsdl:operation name="nmtoken">*`
	`  <wsdl:documentation .... /> ?`
	`  <wsdl:input name="nmtoken"? message="qname">?`
	`  <wsdl:documentation .... /> ?`
	`  </wsdl:input>`
	`  <wsdl:output name="nmtoken"? message="qname">?`
	`  <wsdl:documentation .... /> ?`
	`  </wsdl:output>`
	`  <wsdl:fault name="nmtoken" message="qname"> *`
	`  <wsdl:documentation .... /> ?`
	`  </wsdl:fault>`
	`  </wsdl:operation>`
	`</wsdl:portType>`

UPnP	`<action>`
	`  <name>actionName</name>`
	`  <argumentList>`
	`  <argument>`
	`  <name>formalParameterName</name>`
	`  <direction>in xor out</direction>`
	`  <retval />`
	`  <relatedStateVariable>stateVariableName</relatedStateVariable>`
	`  </argument>`
	`  Declarations for other arguments (if any) go here`
	`  </argumentList>`
	`</action>`

FIGURE 5.3 – Exemple de description de service au format WSDL et UPnP.

L'expression des procédés logiciels dans le langage APEL peut aussi se faire de manière graphique. En effet, le langage APEL a aussi une syntaxe graphique. Une activité est

représentée par un rectangle. Les ports sont des petits carrés comprenant des triangles et les flux de données sont exprimés par un trait entre les ports des activités. La Figure 5.4 est un exemple avec deux actitivés et un flux de données qui va de l'une à l'autre.

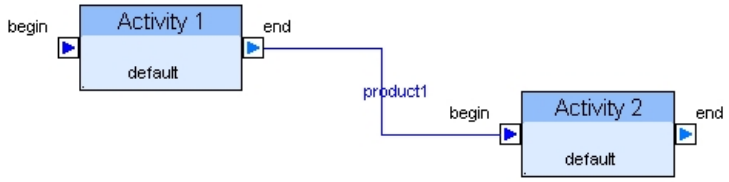


FIGURE 5.4 – Expression d'un procédé en langage APEL.

Nous avons choisi que notre extension du méta-modèle d'APEL n'implique pas de modification de sa syntaxe graphique. Notre extension ne porte que sur le fonctionnement de l'activité. Nous pensons qu'il n'est pas nécessaire de le faire apparaître directement dans le langage pour éviter de le surcharger.

A la différence du langage WS-BPEL, le langage APEL nous permet d'exprimer sous forme d'orchestration des compositions de services hétérogènes de manière abstraite. Le langage WS-BPEL ne permet de réaliser que des orchestrations de services Web. De plus, le langage WS-BPEL permet d'exprimer directement le contrôle pour la composition de services ; dans le langage APEL, le contrôle est implicite puisqu'il découle de l'expression des flux de données entre les activités. Une autre différence entre les deux langages est l'expression des structures de contrôle. Dans WS-BPEL, ces structures sont exprimées explicitement alors que, dans APEL, elles le sont de manière implicite.

2 Méta-modèle de la sécurité

La sécurité est une propriété non-fonctionnelle complexe, qui s'applique dans de nombreux domaines et en particulier pour les services. Nous avons choisi d'étudier cette propriété en se mettant uniquement dans le cadre d'un utilisateur d'un service sécurisé et non d'un fournisseur de service sécurisé. Cette restriction implique que nous n'allons pas gérer les droits et les accès des utilisateurs aux services, comme c'est le cas avec les méthodes de type RBAC [3] [FK92, SCFY96].

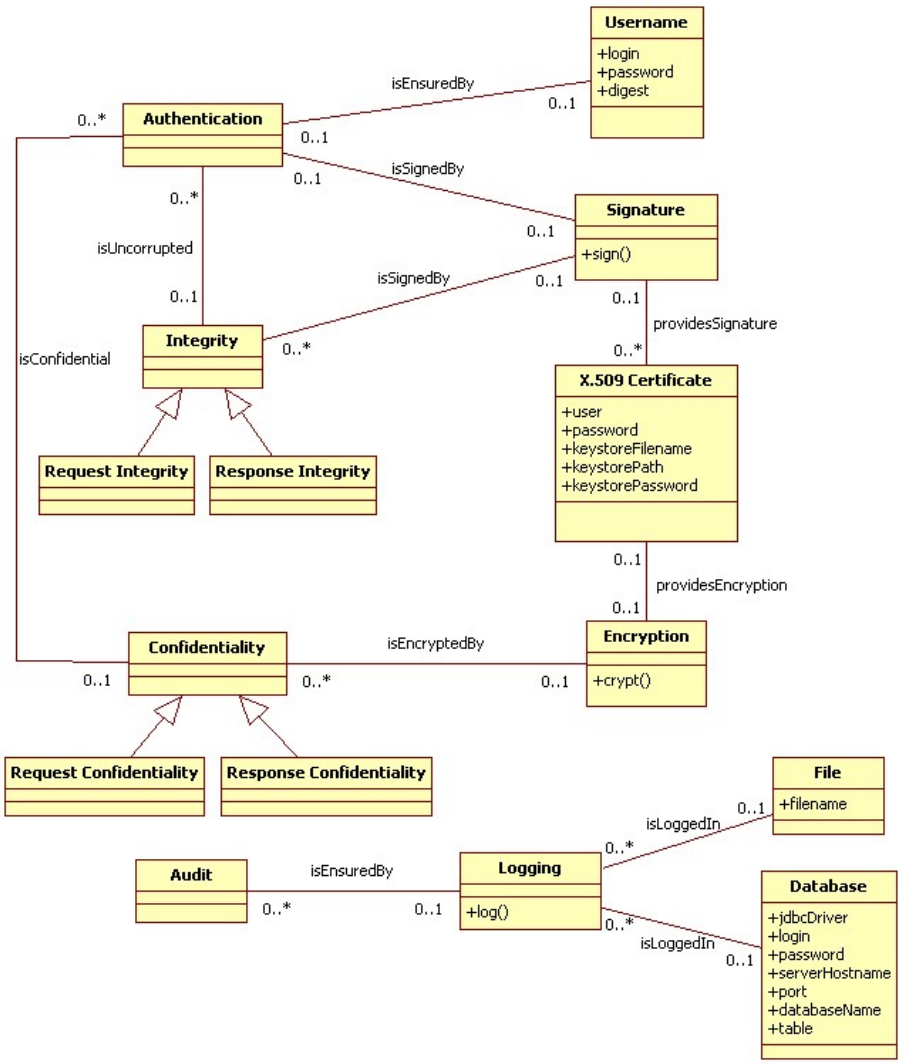


FIGURE 5.5 – Méta-modèle de la sécurité.

3. En anglais, *Role-Based Access Control* pour contrôle d'accès à base de rôles.

Le méta-modèle de la sécurité, présenté dans la Figure 5.5, sert à définir les concepts qui peuvent être utilisés au niveau modèle. Nous avons choisi de simplifier l'ajout de la sécurité pour l'orchestration de services hétérogènes. Pour ce faire, nous voulions que les architectes puissent exprimer leurs contraintes de sécurité avec des termes clairs et appropriés au domaine. Notre méta-modèle de sécurité contient donc les principaux concepts de sécurité, à savoir l'authentification, l'intégrité, la confidentialité et l'audit qui seront utilisés par les architectes lors de la conception de l'orchestration de services abstraits sécurisés.

Cependant, ces quatre concepts de sécurité ne peuvent pas être mis directement en application. Pour ce faire, nous avons exprimé dans le méta-modèle les solutions techniques appropriées au domaine des services. Nous avons choisi d'exprimer ces solutions au niveau méta-modèle, pour que l'expert en sécurité puisse déterminer de façon plus précise les solutions qui sont envisageables et adéquates pour l'application. Les solutions techniques, que nous proposons, sont celles qui ont été définies dans la partie 2.2 page 79. L'authentification et l'intégrité peuvent être assurées par une signature ; la confidentialité, quant à elle, peut être assurée par le chiffrement. Pour l'audit, nous avons choisi de mettre en place une journalisation des événements. Les événements recueillis servent de base à un audit mais ne sont évidemment pas suffisants. L'audit nécessite des analyses avec des informations plus complètes sur le système global.

Les solutions techniques sont ensuite implantées sous plusieurs formes. La principale est le certificat X.509 qui permet d'assurer l'authentification, l'intégrité et la confidentialité en l'utilisant correctement selon les besoins, c'est-à-dire en chiffrant ou en signant les messages émis et/ou en déchiffrant ou en validant les signatures pour les messages reçus. Le nom d'utilisateur et le mot de passe, chiffré ou non, sont aussi des éléments nécessaires pour une authentification de type *Username*. Dans le cas de l'audit, nous proposons un système de journalisation des événements dans un fichier ou dans une base de données.

Ce méta-modèle est construit par niveau (concepts/techniques/technologies) ce qui permet de gérer plus facilement l'évolutivité des technologies. Les concepts et les techniques de la sécurité existent déjà depuis de nombreuses années et par conséquent ils n'évoluent pas ou peu. A l'inverse les technologies de sécurité évoluent assez vite dans le temps suite, par exemple, à des découvertes de failles dans les algorithmes de chiffrement. Pour ajouter une nouvelle technologie, il suffit donc d'étendre le méta-modèle en intégrant ses caractéristiques.

3 Composition des méta-modèles

Nous avons défini dans les deux sections précédentes le méta-modèle de l'orchestration de services abstraits hétérogènes et le méta-modèle de la sécurité. Ces deux méta-modèles permettent de définir quels sont les concepts qui pourront être utilisés au niveau modèle. Nous avons choisi de mettre en relation ces deux méta-modèles avec des liens pour spécifier l'orchestration de services abstraits hétérogènes et sécurisés. Dans la suite, nous présentons les liens que nous avons définis pour mettre en relation les deux méta-modèles.

Le but de notre approche est de simplifier l'ajout de propriétés de sécurité dans une orchestration de services hétérogènes. C'est pour cette raison que nous avons choisi que l'architecte sécurise son orchestration avec des concepts de sécurité simples et connus de tous, ce qui facilite les communications entre les différents intervenants. Les techniques et les technologies de sécurité utilisées doivent être cachées. Les liens partiront des concepts *Authentification*, *Intégrité*, *Confidentialité* et *Audit* du méta-modèle de sécurité.

De l'autre côté, nous avons choisi de rattacher tous ces liens avec le concept d'*Activité*. Ainsi, par définition, une activité peut être sécurisée suivant différentes techniques selon des contraintes de sécurité prédéfinies. La Figure 5.6 exprime les liens entre les deux méta-modèles.

Les liens ont une cardinalité 0 ou 1, ce qui signifie que :

- une activité peut être sécurisée ou non,
- si une activité est sécurisée, :
  - elle peut nécessiter une authentification,
  - elle peut communiquer avec des messages intègres,
  - elle peut communiquer avec des messages confidentiels,
  - ses messages peuvent être tracés.

Toutes ses possibilités peuvent se décliner sur les différents types de services, comme les services Web, les services DPWS et UPnP, mais aussi sur les activités humaines. La cardinalité 0 permet de ne pas imposer d'éléments de sécurité comme, par exemple, une authentification sur du code Java.

Du côté de l'orchestration, les liens s'appliquent au concept d'*Activité*. Cependant, il existe deux types d'activités : simple et composite. Les activités composites comprennent un ensemble d'activités simples. Lorsque l'on annote une activité composite avec un concept de sécurité, toutes les sous-activités de l'activité composite sont alors annotées avec ce concept de sécurité.

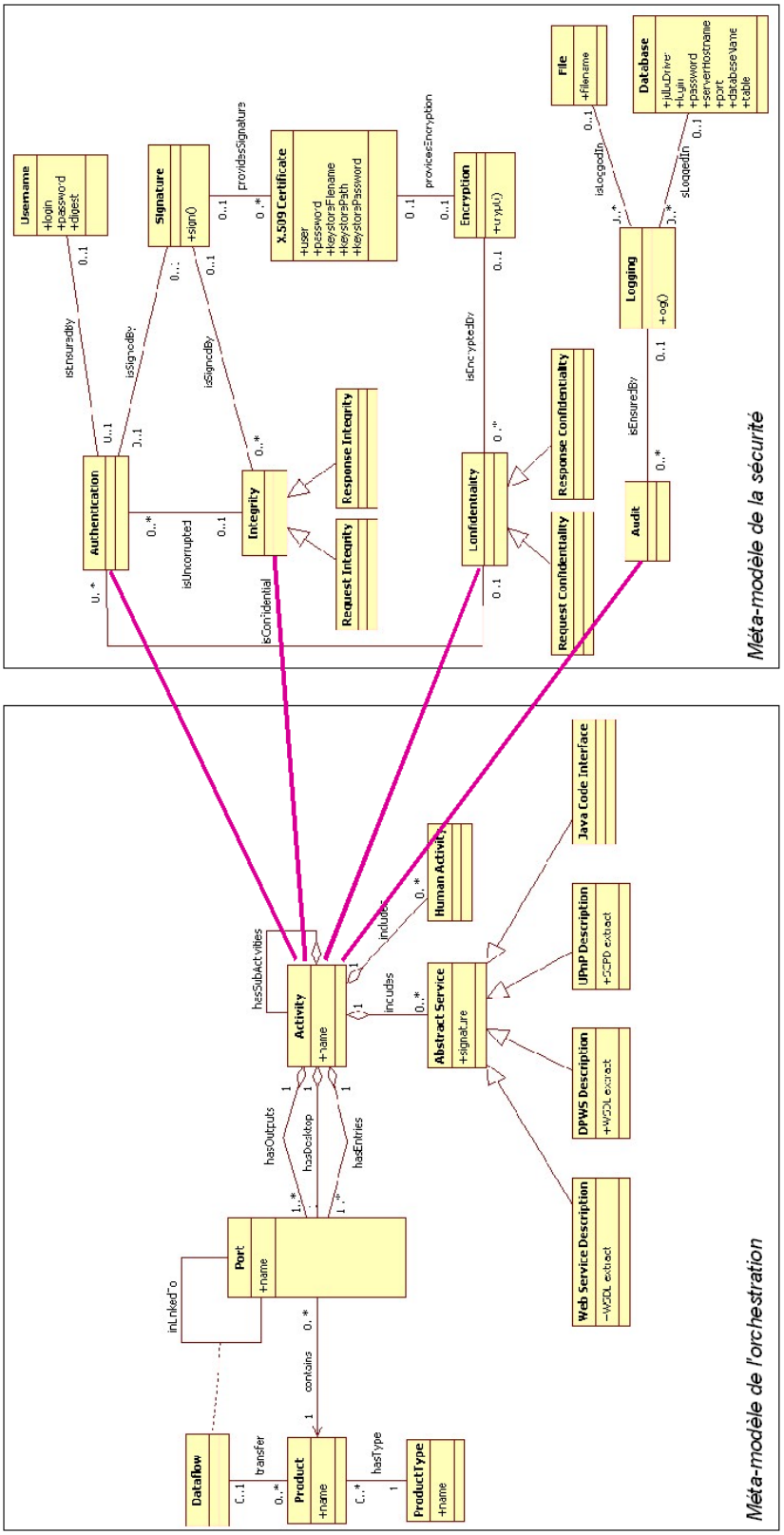


FIGURE 5.6 – Composition des deux méta-modèles.

4 Modèles de l'orchestration et de la sécurité

Dans cette partie, nous nous intéressons aux modèles que nous pouvons réaliser en conformité par rapport aux méta-modèles définis précédemment. Nous définissons des modèles d'orchestration de services hétérogènes et des modèles de sécurité par instanciation du méta-modèle global. Le méta-modèle global comprend le méta-modèle de l'orchestration de services, celui de la sécurité ainsi que les liens que nous avons définis entre eux.

Les modèles d'orchestration ont pour but de définir l'enchaînement des services de façon abstraite. Ces modèles sont définis par des experts du métier. L'utilisation de la syntaxe graphique de APEL simplifie la visualisation de l'orchestration. La Figure 5.7 est un exemple de modèle d'orchestration.

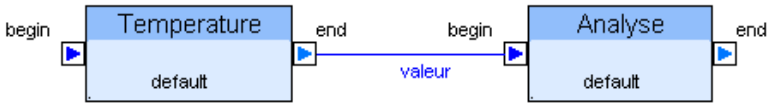


FIGURE 5.7 – Un modèle d'orchestration selon la syntaxe graphique de APEL.

A ces activités sont associés des services abstraits dans des technologies particulières, proposées dans le méta-modèle de l'orchestration. Ces caractéristiques des services sont ajoutées en suivant le principe d'instanciation du méta-modèle. La Figure 5.8 est un exemple de modèle d'orchestration de services hétérogènes.

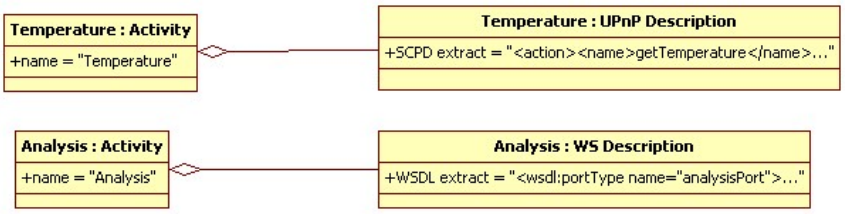


FIGURE 5.8 – Un modèle d'orchestration de services hétérogènes.

Comme pour le modèle d'orchestration, il est possible de réaliser des modèles de sécurité conformes au méta-modèle de la sécurité. La Figure 5.9 est un exemple de modèle de la sécurité qui exprime une authentification par nom d'utilisateur et mot de passe non chiffré.

FIGURE 5.9 – Un modèle de sécurité.

L'authentification définie précédemment s'applique à une activité, plus particulièrement à un service abstrait. La Figure 5.10 montre un lien qui peut être défini entre ces concepts en étant conforme aux liens du niveau méta-modèle.

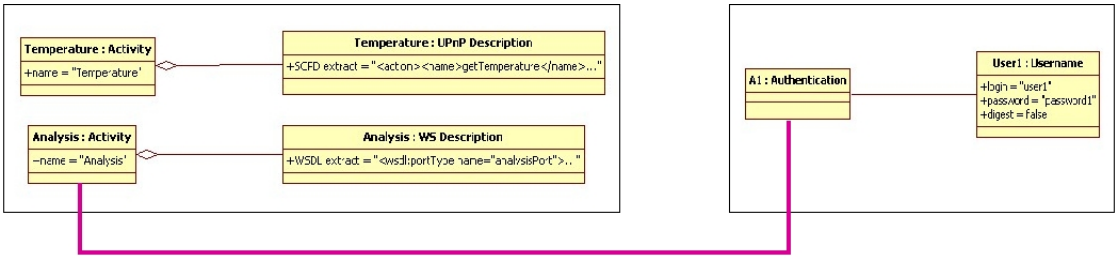


FIGURE 5.10 – Composition des modèles.

Dans le cas des activités composites, il est également possible de leur appliquer des propriétés de sécurité. Pour rappel, une activité composite contient un ensemble de sous-activités simples ou composites. Nous avons choisi que lorsque l'on applique une propriété de sécurité à une activité composite, cette propriété est alors appliquée à toutes ses sous-activités. La Figure 5.11

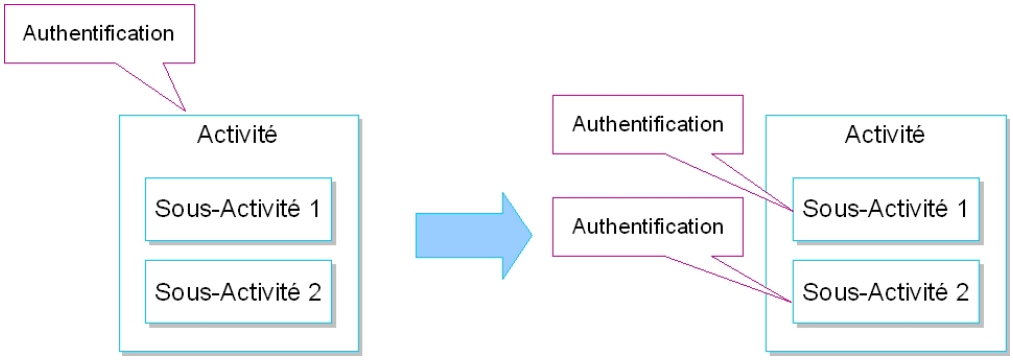


FIGURE 5.11 – Règles d'implication pour les activités composites.

5 Génération de code

La génération du code exécutable se fait après la définition d'un modèle d'orchestration de services sécurisés. Il faut sélectionner les services concrets correspondant aux spécifications définies dans le modèle, générer le code pour appeler les services et pour appliquer, si nécessaire, la sécurité dans les communications. Nous avons choisi de générer le code en deux étapes : la première étape consiste à générer le code fonctionnel pour chaque service en fonction des technologies ; la deuxième étape est une spécialisation de ce code générique avec les paramètres physiques, tels que l'adresse du service, qui ont été récupérés suite à la sélection du service concret approprié.

5.1 Génération du code fonctionnel

Comme nous l'avons expliqué dans l'approche, la génération du code exécutable se fait à partir des spécifications et d'un ensemble de fichiers *templates*. A partir de ces informations, il est possible de générer des fichiers génériques pour appeler les services concrets ; ces fichiers correspondent aux *proxies*.

Nous générons, dans un premier temps, des fichiers génériques en fonction des technologies choisies. Une fois que le service concret qui correspond aux spécifications a été sélectionné, nous complétons les fichiers génériques avec les informations concrètes, comme l'adresse physique. Cette génération en deux temps permet de s'adapter à l'environnement et de supporter la propriété de liaison à retardement.

5.2 Génération du code de sélection

Le code de sélection doit être généré et tissé avec le code fonctionnel. La sélection permet de déterminer à partir des spécifications quel service est le plus approprié. Nous proposons un modèle d'exécution qui contient les informations sur les services concrets disponibles. Le modèle d'exécution est un annuaire amélioré qu'il faut appeler avant l'appel du service. La Figure 5.12 est le diagramme de classes du modèle d'exécution. Il présente les éléments qui sont nécessaires à la sélection de services concrets hétérogènes et sécurisés.

La sélection d'un service se fait sur le type du service qui a été choisi au niveau de la conception de l'orchestration, donc les types des services doivent être présents dans le modèle d'exécution. De plus, les fonctionnalités requises au niveau de la conception doivent être impérativement respectées. La classe *Service* présente les fonctionnalités du service et son adresse physique pour l'interroger.

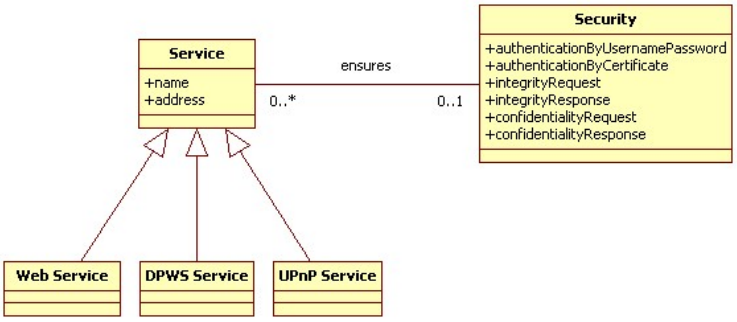


FIGURE 5.12 – Modèle d'exécution.

Le méta-modèle de la sécurité présente les informations de sécurité du point de vue du client. Dans le modèle d'exécution, les propriétés de sécurité doivent être présentées selon le point de vue du service. Concrètement, les informations données doivent permettre de déterminer si le service supporte ou non les propriétés de sécurité définies pendant la phase de conception de l'orchestration. La classe *Security* présente chacune des propriétés déclinées suivant les techniques possibles : authentification par nom d'utilisateur et mot de passe, authentification par certificat, intégrité sur la requête ou sur la réponse, confidentialité sur la requête ou la réponse. L'audit n'impacte pas la sélection des services puisque c'est un mécanisme qui est ajouté côté client et qui est transparent pour le service. Toutes ces propriétés de sécurité sont de type booléen ; le service supporte ou non la propriété.

5.3 Génération du code de sécurité

La génération du code de sécurité se fait une fois que le service concret approprié aux spécifications a été sélectionné. Ce code contient les éléments qui permettent d'assurer les différentes propriétés de sécurité.

L'authentification. Nous proposons deux solutions pour réaliser de l'authentification : nom d'utilisateur/mot de passe ou par certificat électronique X.509. Pour ces deux types d'authentification, le client doit envoyer ses informations personnelles permettant de l'authentifier, soit son nom d'utilisateur et son mot de passe, soit son certificat. La Figure 5.13 illustre l'échange des informations d'authentification entre le client et le serveur.

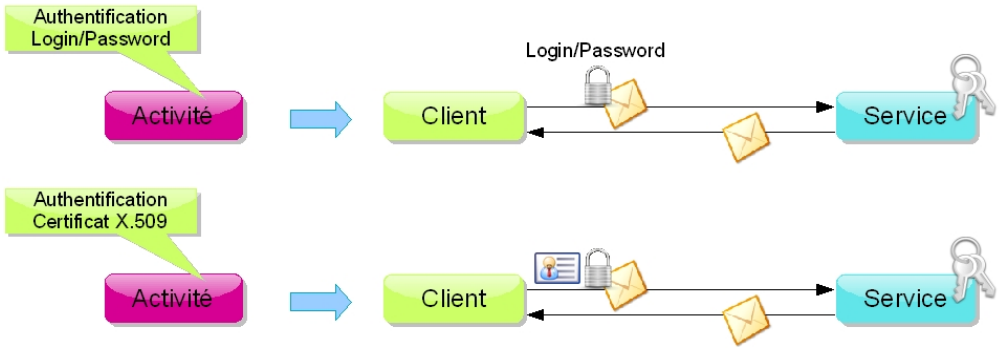


FIGURE 5.13 – L'authentification dans les messages.

Seule la requête contient des informations sécurisées. La réponse n'est pas impactée ; par conséquent, le client n'a pas de traitement à faire sur la réponse. La génération de code se limite à l'ajout du nom d'utilisateur/mot de passe ou du certificat dans le message.

L'intégrité. Comme nous l'avons définie dans le méta-modèle de la sécurité, Figure 5.5, l'intégrité peut être appliquée sur une requête, sur sa réponse ou les deux. Pour la requête, c'est au client de réaliser la signature ; alors que pour la réponse, le client doit s'assurer que la signature est correcte. La Figure 5.14 montre les trois possibilités concernant l'intégrité des messages, de haut en bas : intégrité de la requête, intégrité de la réponse puis intégrité des deux.

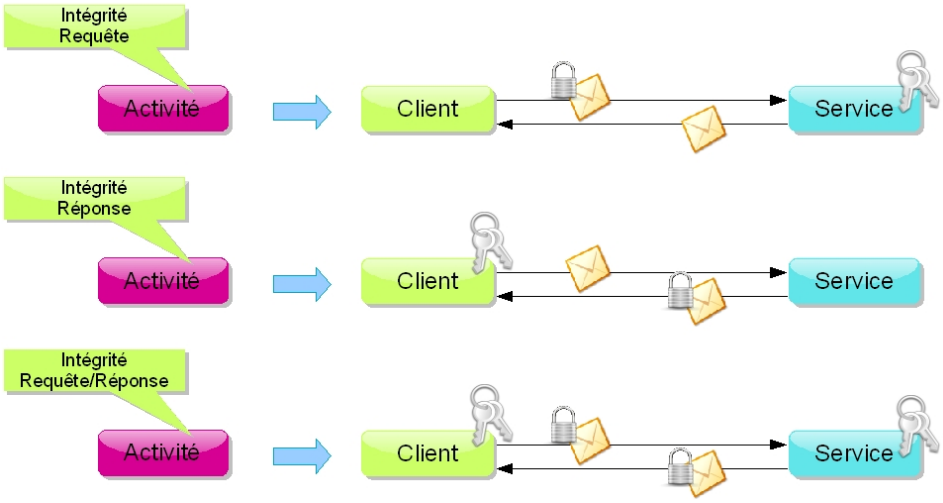


FIGURE 5.14 – L'intégrité dans les messages.

Selon le rôle du client, émetteur ou récepteur d'un message sécurisé, les actions à réaliser sont différentes : signature du message ou vérification de la signature. Par conséquent, la génération du code doit être adaptée à l'intégrité choisie (requête et/ou réponse).

La confidentialité. Tout comme l'intégrité, la confidentialité s'applique soit à la requête, soit à la réponse, soient aux deux. Les données échangées doivent être chiffrées du côté de l'émetteur et déchiffrées du côté du récepteur. La génération du code est, comme pour l'intégrité, adaptée au rôle du client.

L'audit. L'audit consiste à récupérer les informations qui concernent l'échange de messages entre le client et le service. Les informations concernent la date d'émission, de réception ou encore le contenu. Nous proposons deux solutions pour l'audit : dans un fichier ou dans une base de données. La Figure 5.15 illustre le traçage des messages.

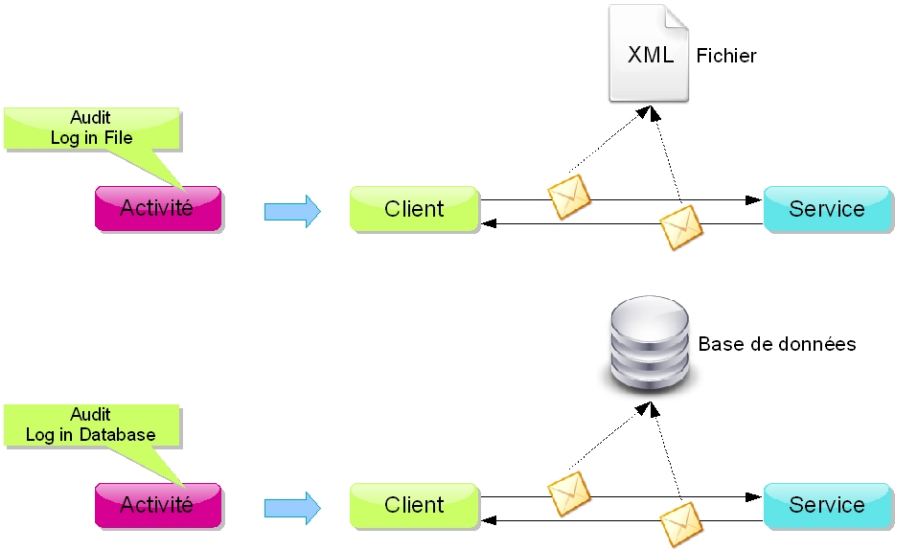


FIGURE 5.15 – L'audit des messages.

Les éléments de sécurité, telles que les certificats, les noms d'utilisateur et les mots de passe, doivent être ajoutés dans les messages échangés entre le client et le fournisseur. Ces informations sont alors tissées avec le code fonctionnel pour y être intégrées. Dans le cadre des services Web et de la technologie WSS4J, les données de sécurité sont définies dans un fichier de déploiement du client et, lors de l'appel du service, ce fichier est passé en paramètre.

6 Synthèse

Dans ce chapitre, nous avons proposé deux méta-modèles pour mettre en application notre approche présentée dans le chapitre précédent. Le premier méta-modèle définit les concepts nécessaires à l'expression d'orchestrations de services hétérogènes. Le deuxième méta-modèle présente les principaux concepts de la sécurité. Ces concepts sont ceux qui sont utilisés au niveau modèle.

Pour exprimer la composition de services abstraits hétérogènes et sécurisés, nous avons définis des liens entre ces deux méta-modèles. Ces liens permettent d'exprimer quels concepts de l'orchestration peuvent être annotés avec des propriétés de sécurité. Pour résumer, un service peut être annoté par les concepts d'authentification, d'intégrité, de confidentialité et d'audit.

A partir des deux méta-modèles et de leurs liens, il est possible de définir des modèles d'orchestration de services abstraits hétérogènes et sécurisés. De plus, pour rendre ces modèles exécutables, nous générons le code fonctionnel, de sélection de services concrets et de sécurité nécessaire à l'exécution. Les codes de sélection et de sécurité sont tissés dans le code fonctionnel.

6

Réalisation et expérimentations

Dans la première partie de ce chapitre, nous allons présenter la plate-forme FOCAS, qui est un environnement de développement et d'exécution d'orchestration de services abstraits. La deuxième section est consacrée aux extensions que nous avons apportées à FOCAS pour intégrer les principes de notre approche. Nous avons étendu FOCAS en Secure FOCAS pour concevoir et exécuter des orchestrations de services abstraits hétérogènes et sécurisés. Dans la troisième section, nous détaillons un cas d'utilisation et son développement avec la plate-forme Secure FOCAS. La quatrième partie de ce chapitre est consacrée aux expérimentations que nous avons réalisées et aux résultats obtenus.

1 Plate-forme FOCAS

Dans cette partie, nous présentons la plate-forme FOCAS qui a été développée dans l'équipe ADELE, du Laboratoire d'Informatique de Grenoble (LIG). Nous avons apporté à cette plate-forme une extension pour permettre la sélection dynamique de services. Nous l'avons également étendue pour qu'elle supporte, à la conception et à l'exécution, les propriétés de sécurité que nous avons présentées précédemment dans le chapitre 5.

Présentons tout d'abord la plate-forme FOCAS dont le but est de permettre la conception d'orchestration de services abstraits, en particulier des services Web, et de réaliser son exécution. Cette plate-forme se divise ainsi en deux parties :

- un **environnement de conception** d'orchestration de services abstraits,
- une **plate-forme d'exécution** de l'orchestration de services, réalisée suivant une architecture client/serveur.

L'environnement de conception est basé sur la plate-forme Eclipse [GHM+05], plus concrètement c'est un plugin Eclipse. Cet environnement permet de définir l'orchestration d'activités abstraites avec le langage APEL [DEA98], qui est un langage de procédés. Dans cet environnement, l'utilisateur définit l'enchaînement des activités ainsi que les données et leur type qui sont utilisés par les activités. Toutes ses informations sont regroupées par domaine dans des modèles différents [PF09, PE08] suivant le principe de la séparation des préoccupations. Les domaines définis par défaut dans FOCAS sont le domaine de contrôle, le domaine des données et le domaine des services. Ces domaines sont composés avec des liens. Ces liens entre domaines permettent de passer des activités abstraites aux services Web concrets en définissant quelles sont les données qui sont échangées.

La Figure 6.1 est une capture d'écran de l'interface de l'environnement de conception. La partie centrale permet d'éditer l'orchestration sous forme graphique. Les propriétés des activités et des données sont définies dans l'onglet *Properties>Basic* en dessous.

Pour passer de la partie conception à la partie exécution, il y a une phase de génération de code qui permet de rendre exécutable l'orchestration abstraite. Cette phase permet de générer les différents *proxies* nécessaires à l'appel des services concrets. C'est l'instanciation du modèle d'orchestration. Pour la technologie des services Web, nous réalisons des *proxies* Apache Axis [Apa06a] à partir d'un fichier *template*. Nous reviendrons plus en détail sur ses fichiers dans la section traitant de l'extension de sécurité que nous avons apportée.

La plate-forme d'exécution est composée d'un serveur qui comprend un moteur d'exécution d'APEL basé sur Mélusine [Jam05]. Ce moteur se charge de l'exécution des instances des orchestrations définies dans l'outil de conception. La plate-forme d'exécution est aussi composée d'un client qui permet de se connecter au serveur et de choisir l'orchestration que l'on souhaite exécuter. La Figure 6.2 est une capture d'écran de l'interface d'exécution. Dans cette interface, on a au centre une instance de l'orchestration modé-

lisée précédemment. Cette orchestration n'est pas encore exécutée, les activités sont de couleur jaune. Les activités en cours d'exécution sont vertes et elles sont rouges lorsqu'elles sont terminées.

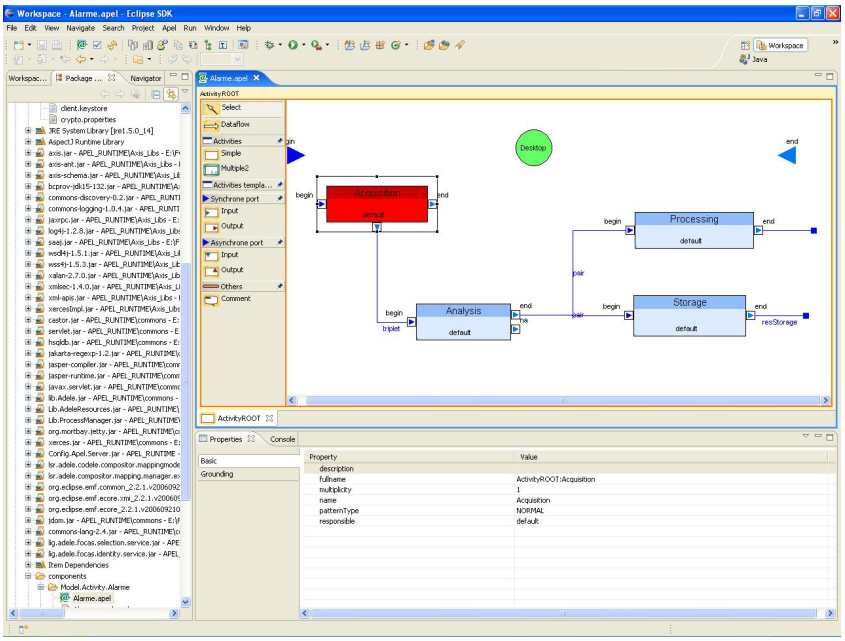


FIGURE 6.1 – Environnement de conception FOCAS.

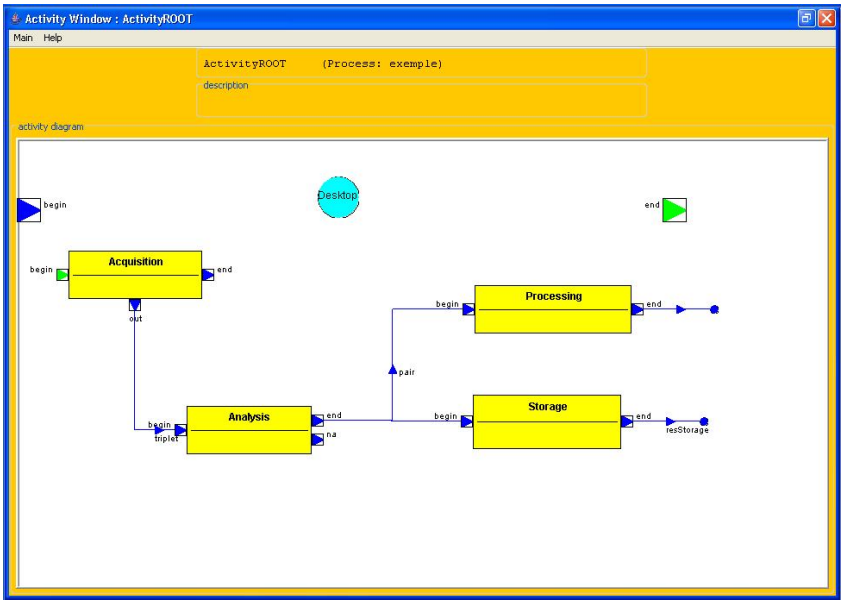


FIGURE 6.2 – Interface pour l'exécution.

En conclusion, l'outil FOCAS est un outil implanté dans notre équipe par G. Pedraza [PF09]. Il permet de concevoir des orchestrations de services abstraits sous forme de définition de procédés avec le langage APEL. Par la suite, nous allons montrer comment nous avons étendu cet outil pour supporter les extensions que nous avons apportées au langage APEL.

L'outil FOCAS permet de spécifier et/ou de visualiser les propriétés fonctionnelles qui sont associées à chacune des activités dans l'onglet *Properties*. Pour ne pas surcharger l'orchestration centrale, il nous a semblé intéressant d'ajouter les propriétés de sécurité dans ce même onglet, comme expliqué dans la section suivante.

2 Extension de sécurité pour FOCAS : Secure FOCAS

Dans cette section, nous présentons l'extension que nous avons apportée à FOCAS pour mettre en application notre approche.

2.1 Présentation générale

Tout d'abord, précisons que nous avons choisi d'utiliser la plate-forme FOCAS pour deux raisons essentielles. En premier lieu, FOCAS est un outil qui permet de réaliser des orchestrations d'activités abstraites et de générer le code nécessaire à son exécution. Ceci est en cohérence totale avec notre propre approche ; la sécurité est ajoutée à une orchestration de services abstraits et le code doit aussi être généré pour l'exécution. La deuxième raison est que notre connaissance privilégiée de l'outil nous a permis de l'étendre facilement. Les éléments de l'orchestration peuvent être étendus autant au niveau de l'interface qu'au niveau de la génération du code. Nos travaux sur la sécurité se placent exactement dans ce contexte. Nous pouvons voir les éléments de sécurité comme des extensions apportées à la définition des services abstraits.

Les extensions, que nous avons ainsi apportées à FOCAS, sont les suivantes :

- la **prise en compte des éléments de sécurité** au niveau de l'interface et de la génération de code,
- la **sélection retardée** des services sécurisés ou non.

Notre plate-forme étendue se nomme Secure FOCAS [1]. Elle est, comme FOCAS, basée sur la plate-forme Eclipse. Les éléments de sécurité et de sélection ont été ajoutés sous la forme de deux *plugins* qui comprennent chacun une interface graphique et le code nécessaire à la génération du code pour la phase d'exécution. La Figure 6.3 est une capture d'écran de Secure FOCAS.

1. `http://cadse.imag.fr/securefocas`

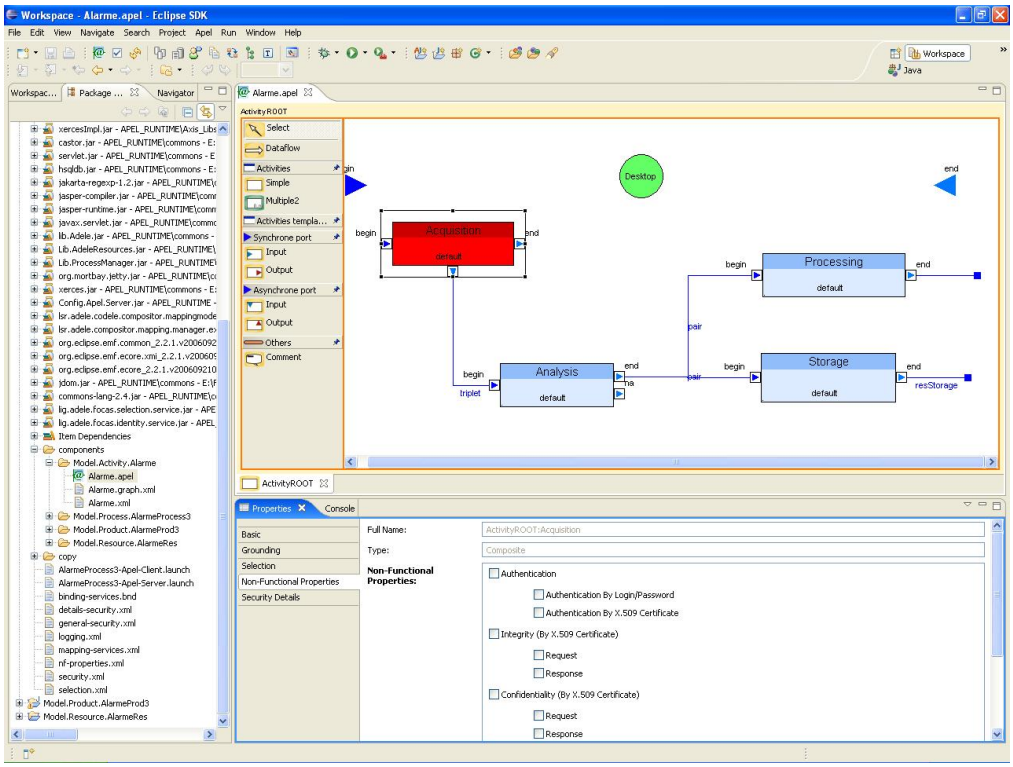


FIGURE 6.3 – Capture d'écran du logiciel Secure FOCAS.

2.2 Extension de sécurité

L'extension majeure, que nous avons ajoutée à FOCAS, est l'ajout de la sécurité au niveau :

- de l'**interface** pour la spécification de l'orchestration de services hétérogènes abstraits et sécurisés,
- de la **génération du code** pour émettre et/ou recevoir des messages sécurisés d'un service concret sécurisé.

2.2.1 Interface utilisateur : les onglets

Nous avons ajouté au niveau de l'interface des onglets dans la partie *Properties* en bas à droite dans la Figure 6.3. Nous avons défini deux onglets : *Non-Functional Properties* et *Security Details*. Le premier onglet *Non-Functional Properties* permet de définir la partie abstraite de la sécurité alors que l'onglet *Security Details* permet de donner les détails techniques nécessaires pour les propriétés de sécurité définies dans l'onglet *Non-Functional Properties*.

Onglet *Non-Functional Properties* L'onglet *Non-Functional Properties*, présenté dans la Figure 6.4, est un onglet qui peut être rempli pour chacune des activités de l'orchestration. Cet onglet comprend le nom de l'activité et son type (simple ou composite) ainsi que les propriétés de sécurité qui peuvent être sélectionnées pour cette activité. Les propriétés de sécurité proposées sont celles de la partie de gauche du méta-modèle de la sécurité (Figure 5.5 page 136), c'est-à-dire uniquement les concepts de sécurité : authentification, intégrité, confidentialité et audit.

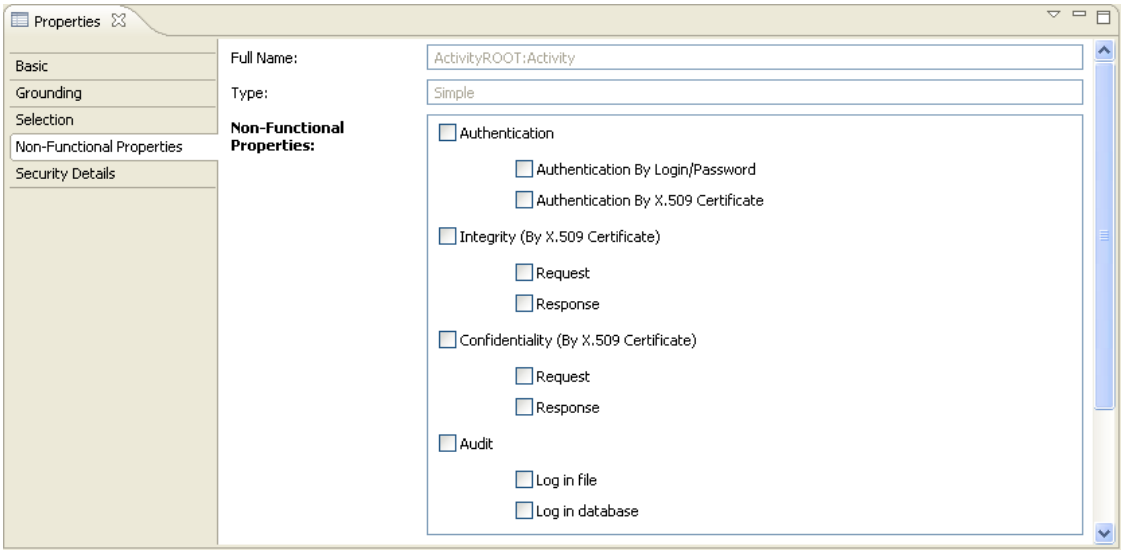


FIGURE 6.4 – Capture d'écran de l'onglet *Non-Functional Properties*.

Il est possible d'affiner son choix pour chacun des concepts de sécurité globaux. En effet, dans cet onglet, l'utilisateur peut choisir entre une authentification par nom d'utilisateur et mot de passe ou alors par certificat. L'intégrité et la confidentialité sont des propriétés qui peuvent s'appliquer soit sur la requête émise, soit sur la requête reçue soit sur les deux. Tous ces choix peuvent être faits en validant les cases à cocher proposées dans l'interface. Il en est de même pour l'audit ; la journalisation des événements peut se faire soit dans un fichier soit dans une base de données. A part pour l'authentification, les choix ne sont pas exclusifs.

Onglet *Security Details* Le deuxième onglet *Security Details* permet à l'utilisateur d'entrer les informations concrètes nécessaires à la mise en place des mécanismes de sécurité définis dans la partie droite du méta-modèle de la sécurité (Figure 5.5 page 136). La Figure 6.5 est une capture d'écran de cet onglet. Comme pour l'onglet précédent, cet onglet est disponible si une activité (simple ou composite) a été sélectionnée dans l'orchestration. Il rappelle le nom de l'activité et ensuite il se divise en quatre parties :

- *Login/Password* qui permet d'entrer les informations concernant le nom d'utilisateur, le mot de passe et son type (chiffré ou non) ;
- *X.509 Certificate* qui permet d'entrer les informations nécessaires à l'utilisation d'un certificat de type X.509 ;
- *Logging File* qui permet d'entrer les informations concernant le nom du fichier dans lequel les traces vont être enregistrées ;
- *Logging Database* qui permet d'entrer les informations nécessaires à l'enregistrement des traces dans une base de données.

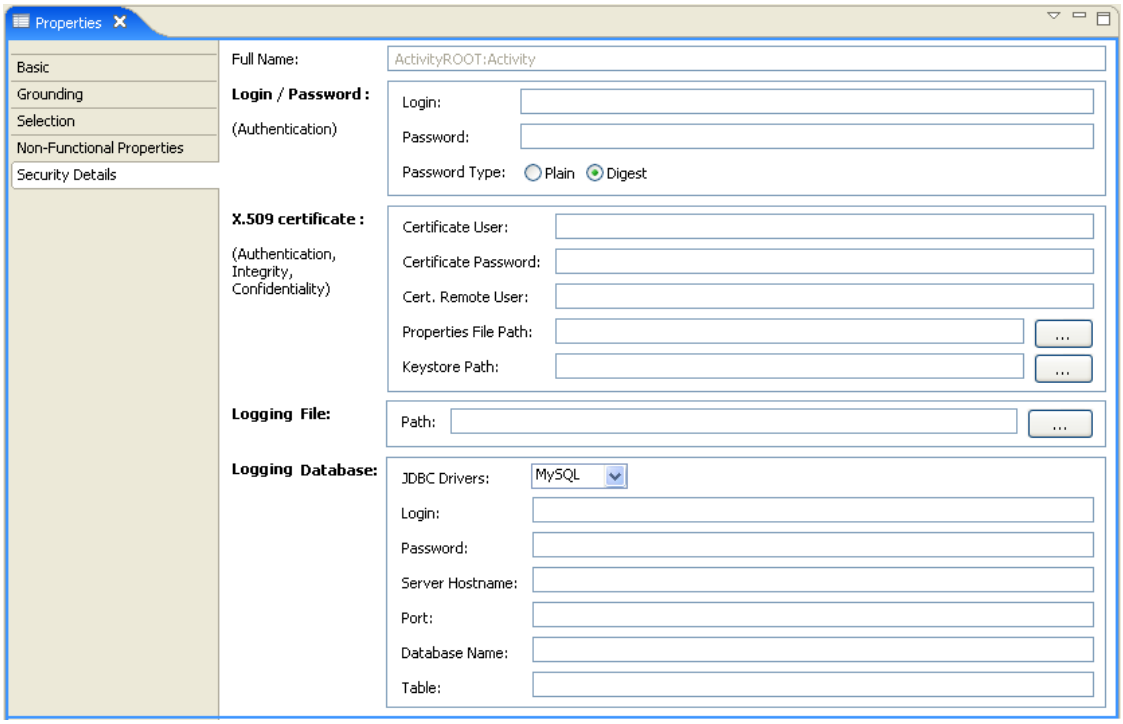


FIGURE 6.5 – Capture d'écran de l'onglet *Security Details*.

La première partie doit être remplie quand la case *Authentication by Login/Password* est cochée dans l'onglet *Non-Functional Properties*. La deuxième partie sert pour l'authentification par certificat, pour l'intégrité et la confidentialité. Les deux parties suivantes servent pour l'audit.

2.2.2 Génération du code : *Templates* JET

Comme expliqué dans l'approche, la génération du code se fait en deux phases. La première phase est la génération des *proxies* pour l'appel des services concrets. Pour ce faire, nous avons dû implanter des fichiers *templates* paramétrables. Ces paramètres sont remplis lors de la deuxième phase de la génération de code qui dépend de la phase de sélection du service concret approprié.

Les fichiers *templates* doivent être écrits en fonction des types de technologies à services. Si nous prenons l'exemple des services Web, nous avons écrit un fichier *template* pour la technologie Apache Axis[2] [Apa06a]. Ce fichier *template* correspond au client d'un service Web.

Pour ajouter la sécurité, nous avons utilisé la bibliothèque Apache WSS4J [Apa06b]. Cette bibliothèque est faite pour sécuriser des messages SOAP et elle requiert l'utilisation de Apache Axis pour la partie métier. Pour sécuriser un service Web, il faut le client standard ainsi qu'un fichier particulier permettant de préciser les informations de sécurité de l'utilisateur. La Figure 6.6 est un extrait de ce fichier de sécurité pour le client, c'est un fichier de déploiement.

```
...
<% if (isAuthenticationLogin.booleanValue()) { %>
  <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
    <parameter name="action" value="UsernameToken"/>
    <parameter name="user" value="<%=authenticationLoginLogin%>"/>
    <parameter name="passwordCallbackClass"
               value="<%=_package%>.service.<%=className%>_PWCallback" />
    <parameter name="passwordType" value="<%=authenticationLoginPasswordType%>"/>
  </handler>
<% } %>
<% if (isAuthenticationCertificate.booleanValue()) { %>
  <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
    <parameter name="passwordCallbackClass"
               value="<%=_package%>.service.<%=className%>_PWCallback"/>
    <parameter name="action" value="Signature"/>
    <parameter name="signaturePropFile" value="<%=fileNameCryptoProperties%>"/>
    <parameter name="user" value="<%=authenticationCertLogin%>"/>
    <parameter name="signatureKeyIdentifier" value="DirectReference"/>
    <parameter name="signatureParts"
               value="{}{http://schemas.xmlsoap.org/soap/envelope/}Body"/>
  </handler>
<% } %>
...
```

FIGURE 6.6 – Extrait du code du fichier de déploiement pour le client.

2. Apache Axis est une bibliothèque Java qui a une API pour développer et interroger des services Web.

Pour réaliser ces fichiers *templates*, nous avons utilisé la technologie JET [3] [Ecl]. Cette technologie supporte en particulier l'écriture de fichiers Java et XML sous forme de *template*. Les éléments qui sont compris entre les signes <% et %> sont des paramètres. Pour le fichier de sécurité, les paramètres sont remplis avec les informations fournies dans les onglets de l'interface. Cette première phase de génération ne permet pas d'interroger directement un service Web concret puisqu'il n'a pas encore été sélectionné et par conséquent son adresse physique n'est pas connue ; elle ne sera effectivement connue qu'à l'exécution.

2.3 Extension de sélection retardée

Jusqu'à présent dans FOCAS, la sélection des services concrets se faisaient de manière statique, c'est-à-dire que les services concrets étaient choisis avant l'exécution, à l'issue de la conception. Nous avons donc ajouté comme fonctionnalité la sélection dynamique. Il est vrai que l'utilisateur peut vouloir un service particulier pour une tâche précise. Mais, si l'utilisateur n'a pas de préférence particulière, il est préférable que le service soit choisi à l'exécution, le service a plus de chance d'être disponible et de correspondre aux attentes de l'utilisateur. Une fois l'orchestration conçue, son(ses) exécution(s) ne se déroule(nt) pas forcément immédiatement après ; le temps écoulé entre ces deux activités peut être assez long, ce qui rend possible la disparition ou l'évolution d'un service. De plus, une exécution de services est réitérable dans le temps, l'évolution des technologies et des besoins peut entraîner l'évolution d'un service, ce qui pose aussi problème si la sélection est faite lors de la conception.

L'ajout d'une fonctionnalité de sélection dynamique permet de garder les propriétés de faible couplage et de liaison retardée offertes par l'architecture à services. L'idée est d'ajouter un registre des services effectivement disponibles lors de l'exécution. Dans notre approche, nous avions représenté ce registre sous forme d'un modèle d'exécution.

Cet ajout intervient à deux niveaux dans FOCAS :

- au niveau de l'**interface** des propriétés pour chaque activité ;
- au niveau **exécution**, il faut en premier sélectionner le service le plus approprié selon les fonctionnalités définies dans l'orchestration abstraite. Puis, il faut interroger le service concret avec les informations récupérées dans l'annuaire.

La Figure 6.7 est une capture d'écran de l'onglet de sélection que nous avons ajouté. Cet onglet laisse la possibilité à l'utilisateur de choisir son type de sélection (statique ou dynamique). Dans le cas d'une sélection dynamique, l'utilisateur doit préciser l'adresse du registre.

3. *Java Emitter Templates*

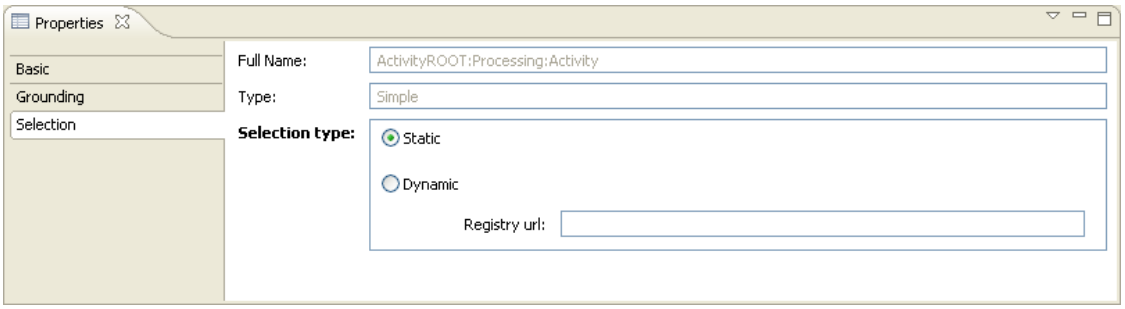


FIGURE 6.7 – Capture d'écran de l'onglet de sélection.

L'architecture mise en place pour la sélection dynamique est présentée dans la Figure 6.8. Nous retrouvons dans cette architecture les trois acteurs de l'architecture à services. Le fonctionnement de cette sélection est donnée par l'Algorithme 6.

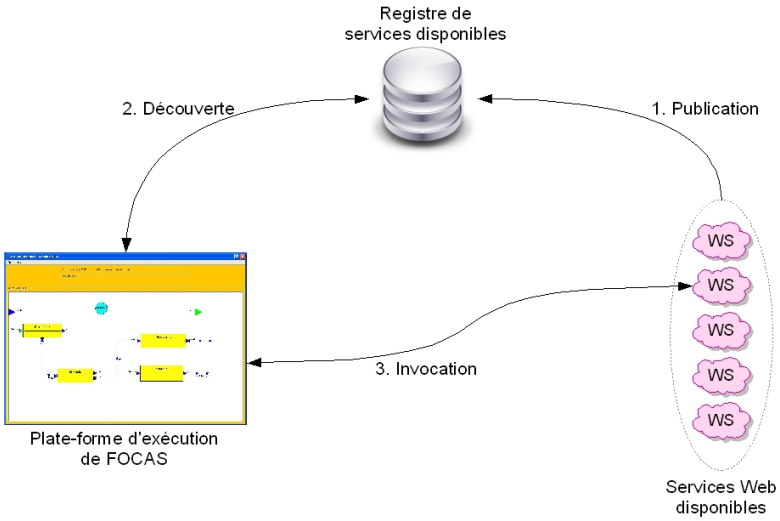


FIGURE 6.8 – Architecture pour la sélection dynamique de services.

Algorithme 6 : Algorithme de sélection dynamique à l'exécution.

1 **début**
2 **pour chaque** *activité* **faire**
3 **si** *l'activité requiert une sélection dynamique* **alors**
4 Chercher dans le registre le "bon" service ;
5 Récupérer les informations telles que son adresse pour l'interroger ;
6 Compléter le template Axis avec les informations récupérées ;
7 Appeler le service Web ;
8 **fin**

Nous avons également enrichi cette extension de sélection pour une recherche de services sécurisés. L'impact de l'ajout de la sécurité est faible puisque les propriétés de sécurité sont ajoutés dans les paramètres permettant la recherche du « bon » service. L'algorithme 7 présente le fonctionnement de la sélection dynamique d'un service sécurisé.

Algorithme 7 : Algorithme de la sélection d'un service sécurisé.

```
1 début
2     pour chaque activité faire
3         si l'activité requiert une sélection dynamique alors
4             Chercher dans le registre le "bon" service sécurisé ;
5             Récupérer les informations telles que son adresse pour l'interroger ;
6             Compléter le template Axis avec les informations récupérées ;
7         Appeler le service Web ;
8 fin
```

Au départ l'extension de la sélection dynamique a été réalisée uniquement pour la technologie des services Web pour FOCAS. Pour supporter plusieurs technologies à services, il faut que le modèle d'exécution contienne les services disponibles triés par type et que, lors de la recherche du service approprié à la spécification, le type du service soit aussi passé en paramètre.

3 Cas d'utilisation

Notre approche a été expérimentée sur un cas d'utilisation issu du projet Européen SODA [4]. Ce cas d'utilisation a été plus particulièrement formalisé par l'entreprise Thales. Il concerne la mise en place d'une chaîne d'acquisition de données pour une entreprise. Cette chaîne d'acquisition recupère des données de capteurs, ensuite ces valeurs sont récupérées et analysées. Le résultat de l'analyse est alors stocké dans une base de données. En fonction du résultat de l'analyse, un mail ou un SMS est envoyé au responsable de la chaîne d'acquisition et une action sur la chaîne peut-être exécutée.

3.1 Présentation générale

Selon notre approche, nous devons présenter cette chaîne d'acquisition sous forme d'une orchestration de services abstraits. La Figure 6.9 représente notre cas d'utilisation.

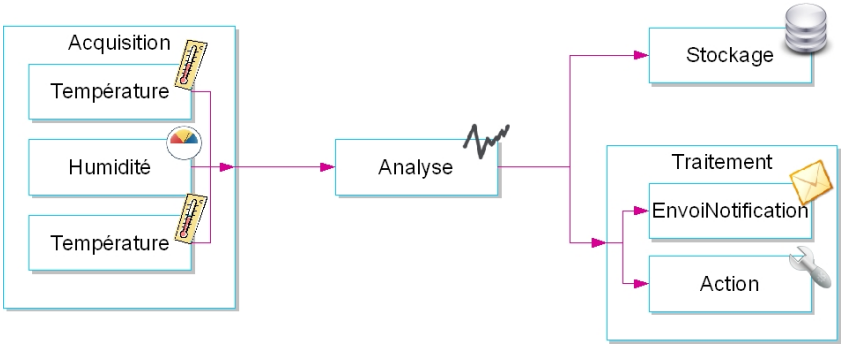


FIGURE 6.9 – Spécification de la chaîne d'acquisition.

Les détails de la chaîne d'acquisition sont les suivants :

- la première partie est l'acquisition des données, qui sont des températures et le taux d'humidité pour une machine ;
- ensuite les données collectées sont analysées par lot de dix valeurs, l'analyse consiste à calculer la moyenne des températures et agréger le taux d'humidité ;
- les données analysées sont à la fois stockées pour garder un historique de ce qui s'est passé dans l'usine pour être traitées. Le traitement consiste en l'envoi d'un mail au responsable du site et en une action sur la machine de l'usine (arrêt, ralentissement...).

Dans ces informations, nous avons des détails concernant les activités que requiert ce système d'alarme mais aussi des informations sur les données qui doivent être échangées entre ces activités. Toutes ces informations seront utilisées par la suite pour la définition abstraite de l'orchestration dans la plate-forme FOCAS.

4. Acronyme de *Service Oriented Device and Delivery*, `http://www.soda-itea.org/`

3.2 En langage APEL

La Figure 6.10 représente une vue globale du modèle de contrôle du système d'alarme. Ce modèle comprend quatre activités : *Acquisition*, *Analysis*, *Processing* et *Storage*. Seule l'activité *Storage* est une activité simple, les autres activités sont des activités composites ; elles comprennent des sous-activités.

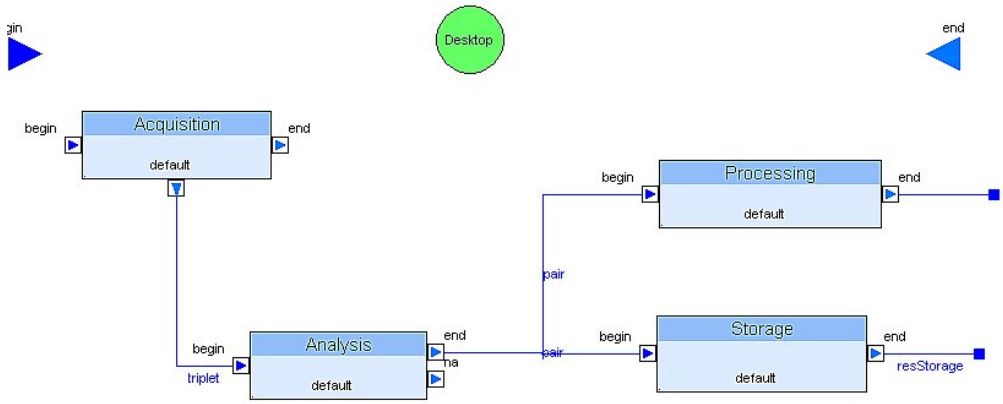


FIGURE 6.10 – Chaîne d'acquisition exprimée avec APEL.

L'activité *Acquisition* a pour but de récolter les données reçues par les capteurs de température et d'humidité. Elle se compose donc de deux activités *Temperature1* et *Temperature2* ainsi que d'une activité *Humidity*. L'acquisition des données est asynchrone. Nous avons ajouté à cette activité composite *Acquisition* une activité *Aggregator* qui permet de créer un tuple comprenant deux valeurs de température et une valeur pour le taux d'humidité. Ce tuple est ensuite envoyée à l'activité *Analysis*. La Figure 6.11 présente les sous-activités qui composent l'activité *Acquisition*.

L'activité *Analysis* a pour but de calculer la moyenne des températures et celle du taux d'humidité. Ces moyennes sont calculées à partir de dix tuples. La première partie de l'activité consiste en l'agregation des données en un tableau de tuples, la deuxième partie est l'appel d'un service avec la sous-activité *Call_Analysis* qui est capable de calculer les moyennes. Ces moyennes sont ensuite envoyées à l'activité *Acquisition*.

Le contenu de l'activité *Processing* est présenté dans la Figure 6.12. Elle se compose de trois activités : *ProcAnalysis*, *SendNotification* et *Action*. L'activité *ProcAnalysis* analyse les valeurs obtenues de l'activité *Analysis* et s'assure que ces valeurs sont « normales » et, si c'est le cas l'activité se termine par son port *none*. Si les valeurs sont dans un intervalle critique, l'activité *ProcAnalysis* se termine par le port *mail* et l'activité *SendNotification* est activée, elle envoie un message au responsable du système d'alarme. Si les mesures montrent que la situation est dangereuse, le port *both* est activé ; un message est envoyé au responsable et une action est réalisée sur une machine avec l'activité *Action*.

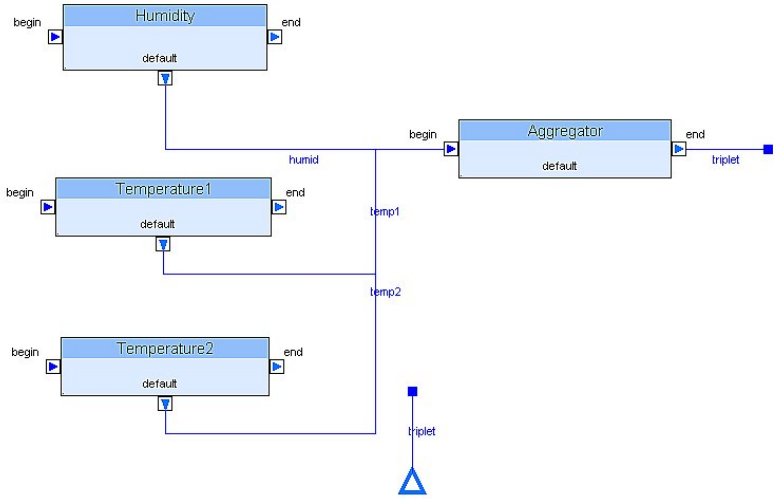


FIGURE 6.11 – Détail de l'activité *Acquisition*.

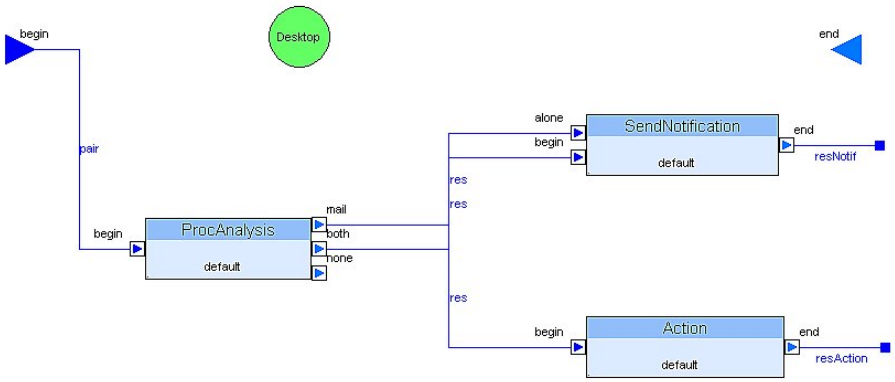


FIGURE 6.12 – Détail de l'activité *Processing*.

Nous pouvons noter que les structures de contrôle (*while* et *if/then/else*) sont exprimées dans le code des activités et non graphiquement au niveau du contrôle.

Les types des données, qui circulent dans cette orchestration, sont exprimés dans le modèle de données. Pour faciliter la composition du modèle de contrôle avec celui de données, nous avons créé un type de produit dans le modèle de contrôle correspondant à chaque type du modèle de données ; en plus les types ont les mêmes noms dans les deux modèles. Les types utilisés sont donc les suivants :

– ***DoubleA*** : type complexe pour représenter un *double*. Ce type est utilisé pour les données des capteurs.

- ***Measure*** : type complexe représentant un tuple qui comprend deux températures et un taux d'humidité soit trois *DoubleA* (`<DoubleA, DoubleA, DoubleA>`).
- ***LotMeasures*** : type complexe contenant un ensemble de dix *Measure*.
- ***MessageData*** : type complexe contenant les informations nécessaires pour envoyer un message (adresse, sujet, texte du message).
- ***BooleanA*** : type complexe pour représenter un *booléen*. Ce type est utilisé pour indiquer l'action qui doit être effectuée en fonction de l'analyse des moyennes.

3.3 Cas d'utilisation sécurisé

Notre cas d'utilisation sécurisé est une extension du cas d'utilisation précédent. Pour le système d'alarme, les propriétés de sécurité sont spécifiées ainsi :

- les données venant des capteurs doivent être confidentielles ;
- l'analyse ne peut pas être réalisée par n'importe qui, elle nécessite donc un mécanisme d'authentification ;
- l'analyse doit se faire sur des données non-altérées, il faut donc qu'elles soient intègres pour la requête. La réponse attendue doit également être non-altérée, donc intègre.
- le stockage des informations nécessite une authentification ;
- les traitements qui sont réalisés sur les moyennes calculées nécessite une authentification. N'importe qui ne peut pas exécuter des traitements sur des données.
- les données communiquées pour l'envoi d'une notification doivent être confidentielles ;
- les données qui sont reçues pour déclencher une action doivent être intègres.

La Figure 6.13 illustre les contraintes de sécurité que nous ajoutons à la chaîne d'acquisition.

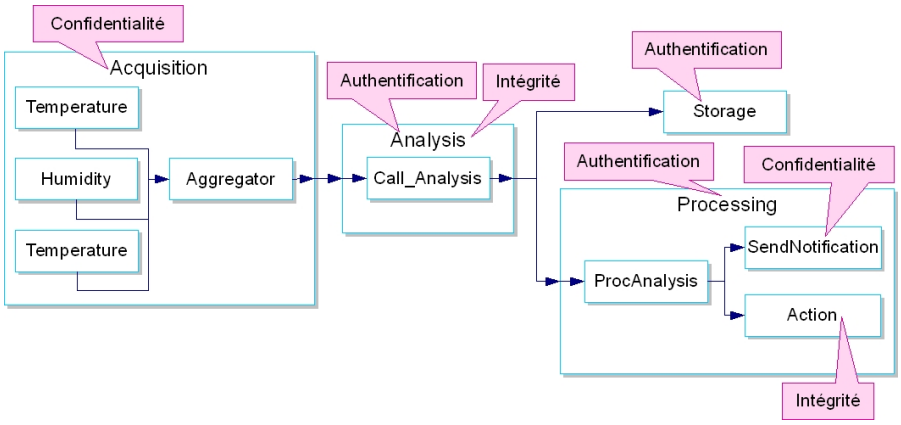


FIGURE 6.13 – Spécification de la chaîne d'acquisition sécurisée.

3.4 Dans Secure FOCAS

Ces propriétés de sécurité doivent être ajoutées pour chacune des activités dans l'interface de l'environnement de conception de Secure FOCAS. Ces propriétés sont ajoutées dans les deux onglets *Non-Functional Properties* et *Security Details*. Si nous prenons l'exemple de l'activité *Analysis*, elle nécessite l'authentification par nom d'utilisateur et mot de passe, ainsi que l'intégrité sur la requête et la réponse pour les données échangées. Le premier onglet *Non-Functional Properties* est donc rempli comme dans la Figure 6.14.

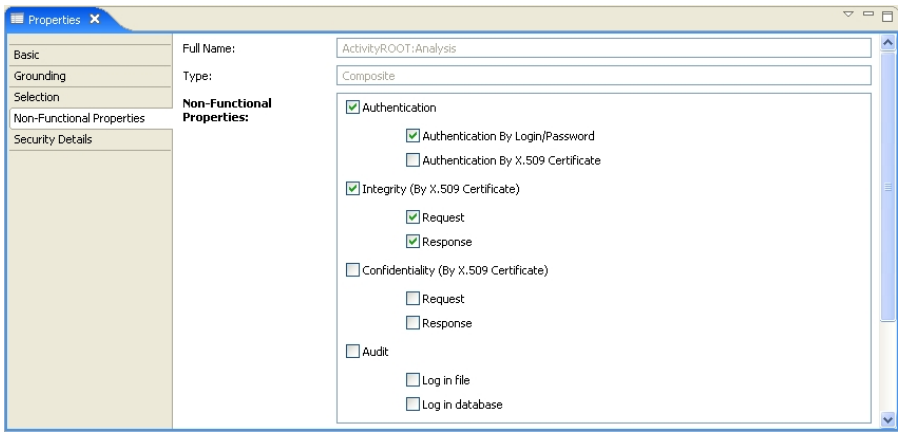


FIGURE 6.14 – Onglet *Non-Functional Properties* pour sécuriser l'activité *Analysis*.

Les informations concrètes concernant la mise en place de l'authentification par nom d'utilisateur et mot de passe ainsi que la signature par certificat X.509 pour l'intégrité doivent être entrées par l'utilisateur dans l'onglet *Security Details*. La Figure 6.15 présente cet onglet avec les informations concrètes pour l'activité *Analysis*. Il est à noter que la génération du code et l'exécution du système d'alarme ne nécessitent pas d'intervention humaine à part pour les déclencher, l'utilisateur n'interagit qu'avec ces deux onglets.

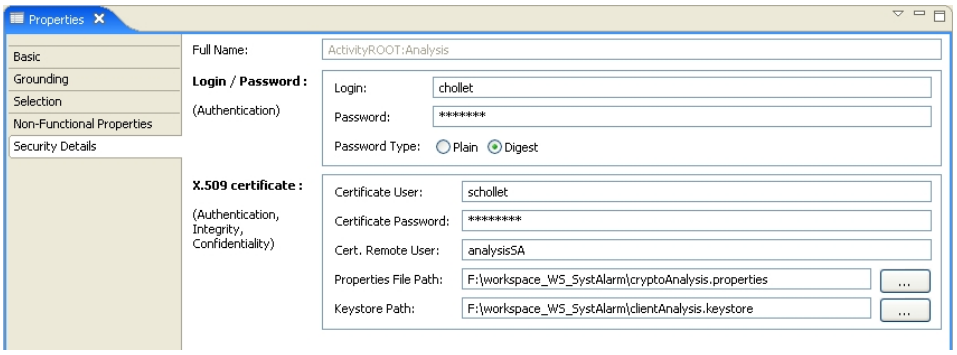


FIGURE 6.15 – Onglet *Security Details* pour sécuriser l'activité *Analysis*.

4 Expérimentations

Dans cette section, nous présentons les résultats que nous avons obtenus avec Secure FOCAS. Nos résultats se concentrent uniquement sur l'orchestration statique et dynamique et sur la sécurité. Pour ces deux aspects, nous avons voulu tester la productivité des développeurs en mesurant le nombre de lignes de code générées et écrites. Pour l'aspect de sécurité, nous avons testé le temps d'apprentissage des utilisateurs avec et sans notre outil.

4.1 L'orchestration statique et dynamique

Nous avons testé le nombre de lignes de code qui sont générées et écrites pour réaliser une sélection statique et une sélection dynamique pour le même scénario. Ce scénario de test est celui du système d'alarme, pour la première expérience il ne comprend aucune propriété de sécurité. Le Tableau 6.1 présente les résultats que nous avons obtenus.

	Sélection statique	**Sélection dynamique**
Lignes de code générées	1145	1215
Lignes de code écrites	152	152
TOTAL	**1297**	**1367**

TABLE 6.1 – Nombre de lignes de code générées et écrites pour les deux types de sélection.

Il est intéressant de noter que la sélection dynamique nécessite plus de lignes de code, un peu plus de 6%, pour la mettre en place pour chacune des activités. Par contre ces lignes supplémentaires sont uniquement des lignes générées ; l'utilisateur n'a pas à développer de code en plus. Notre extension permet donc de profiter des propriétés des services, telle que la liaison à retardement, sans impacter le travail du développeur.

Si nous détaillons les lignes de code qui sont générées, nous constatons que le code généré supplémentaire pour la sélection dynamique se trouve dans le composant de liaison. Ce code est celui qui permet de gérer la logique pour l'invocation des services. Il a en charge l'instanciation des *proxies* et l'invocation des opérations. Il représente environ 540 lignes de code. Dans le cas de la sélection dynamique, ce composant de liaison doit en plus invoquer le composant de sélection des services, il comprend alors 610 lignes de code, soit environ 13% de code supplémentaire.

4.2 La génération du code métier et de sécurité

Les lignes de code étudiées dans la section précédente ne sont que des lignes de code qui implantent la partie métier de l'orchestration. Nous avons aussi étudié le nombre de lignes de code générées et écrites si on ajoute des propriétés de sécurité à l'orchestration. La Figure 6.16 montre le cas d'utilisation que nous avons utilisé pour réaliser les tests. Les activités sont annotées au plus d'une propriété de sécurité et nous avons un exemple de tous les types de propriétés de sécurité. Le Tableau 6.2 présente les mesures que nous avons obtenues.

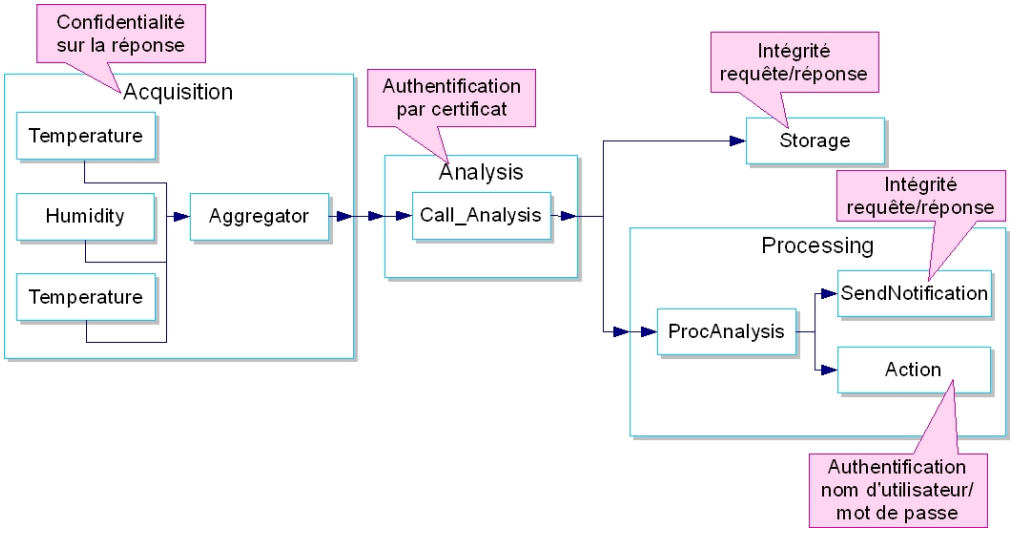


FIGURE 6.16 – Scénario de test.

		Sélection statique	**Sélection dynamique**
Lignes de code générées	Métier	1468	1539
	Sécurité	323	324
Lignes de code écrites	Métier	152	152
	Sécurité	0	0
TOTAL		**1943**	**2015**

TABLE 6.2 – Nombre de lignes de code générées et écrites pour les propriétés de sécurité.

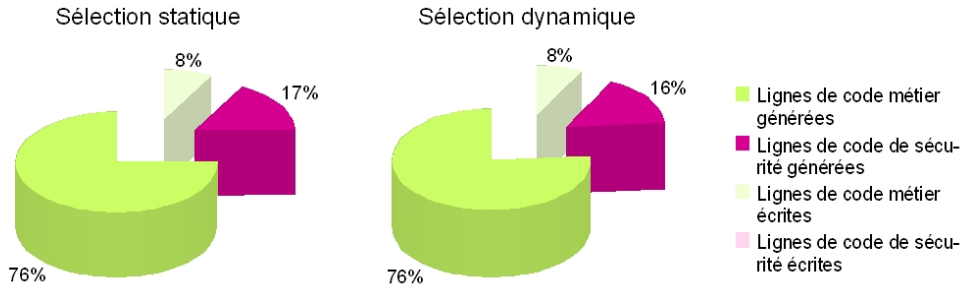


FIGURE 6.17 – Pourcentage de lignes de code générées et écrites.

Nous constatons, d'après les résultats obtenus, que les lignes de code de sécurité sont exclusivement des lignes de code générées. Ceci permet d'améliorer la productivité du développeur, il n'a pas à ajouter des lignes de sécurité dans le code fonctionnel de l'orchestration.

La deuxième remarque, que nous pouvons faire, est que le nombre de lignes de code générées est le même quelque soit le type de sélection. Les propriétés de sécurité sont seulement ajoutées en tant que paramètres pour la sélection d'un service sécurisé approprié à la spécification.

4.3 L'apprentissage des utilisateurs

Nous avons ensuite chercher à savoir combien de temps était nécessaire pour prendre en main notre extension de sécurité dans l'outil FOCAS. En effet, nous avions remarqué précédemment que la mise en place (apprentissage de la technologie, développement et tests) de la sécurité avec Apache WSS4J [Apa06b] était d'environ une semaine pour des scénarios du type du système d'alarme. Cette mesure avait été prise pour des utilisateurs d'un niveau expert en Apache Axis [Apa06a], technologie utilisée pour implanter des services Web, et d'un niveau débutant pour Apache WSS4J.

Dans cette partie, nous souhaitons vérifier que le temps d'apprentissage dans notre outil est diminué par rapport aux résultats obtenus sans notre outil. Les scénarios de tests se basent sur les activités du système d'alarme et ils sont présentés dans le Tableau 6.3.

Les scénarios présentés sont des scénarios simples qui proposent pour une partie des activités une propriété de sécurité. Le testeur doit remplir les informations concernant uniquement la sécurité dans les deux onglets *Non-Functional Properties* et *Security Details*. Pour réaliser les tests, les utilisateurs avaient pour document le protocole de tests et le manuel d'utilisation de Secure FOCAS.

Activités	**Scénario 1**	**Scénario 2**	**Scénario 3**
Acquisition			
Temperature 1	Confidentialité	Confidentialité	Aucune
Temperature 2	Aucune	Aucune	Aucune
Humidity	Confidentialité	Confidentialité	Aucune
Analysis	Authentification	Authentification	Authentification
Storage	Intégrité	Aucune	Intégrité
Processing			
SendNotification	Intégrité	Intégrité	Aucune
Action	Authentification	Confidentialité	Authentification

TABLE 6.3 – Les scénarios de tests.

Le Tableau 6.4 récapitule les temps que nous avons obtenus avec notre outil pour chacun des testeurs pour les trois scénarios présentés précédemment.

Testeurs	**Scénario 1**	**Scénario 2**	**Scénario 3**
T1	07:20	05:07	01:55
T2	17:58	09:42	04:08
T3	14:39	08:39	04:08
T4	17:10	09:25	03:00
T5	11:24	07:37	02:28
T6	14:45	09:01	04:39
T7	13:30	07:34	04:38
T8	09:59	05:32	02:06
T9	10:25	07:31	02:40
T10	08:54	05:21	02:34
Moyenne	**12:36**	**7:32**	**3:13**

Table 6.4: Temps obtenus par les testeurs pour chacun des scénarios.

La Figure 6.18 est la représentation sous forme de graphique du Tableau 6.4.

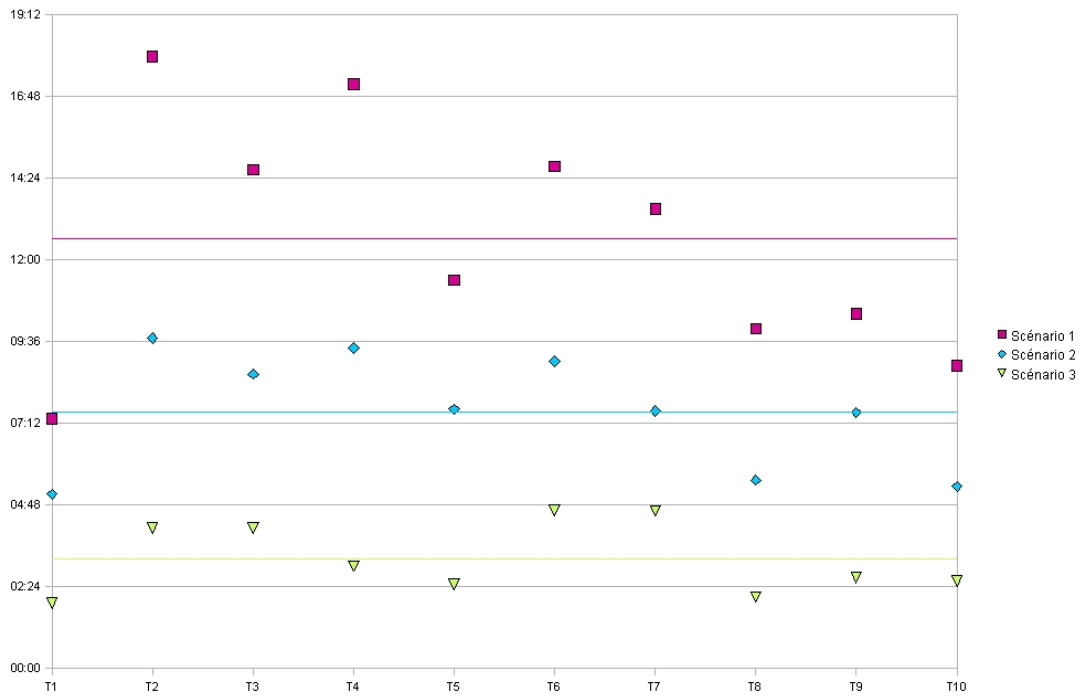


FIGURE 6.18 – Représentation des différents temps obtenus pour les trois scénarios.

Nous remarquons que le temps d'apprentissage pour les différents scénarios est très inférieur au temps obtenus sans notre outil, de plusieurs jours à quelques minutes. Les utilisateurs n'ont pas eu besoin d'apprendre les technologies utilisées pour écrire les clients des services Web et pour les sécuriser. L'interface, que nous avons proposée, diminue fortement le temps d'apprentissage de la sécurisation des services pour les utilisateurs. Les utilisateurs ne se concentrent que sur les informations utiles pour la sécurisation sans se soucier de l'implantation.

Nous avons choisi de refaire des tests avec, cette fois-ci, les mêmes utilisateurs devenus plus familier avec notre outil. La réalisation sans notre outil pour sécuriser le système d'alarme est d'environ un ou deux jours selon les scénarios. Les résultats obtenus avec des experts de notre outil pour les trois scénarios sont donnés dans le Tableau 6.5.

Testeurs	Scénario 1	Scénario 2	Scénario 3
T1	03:40	04:05	01:27
T2	06:20	06:19	02:51
T3	05:45	04:53	02:28
T4	05:14	04:48	02:25
T5	05:40	07:02	01:53
T6	05:53	07:13	02:27
T7	05:43	05:58	02:31
T8	04:34	04:10	01:18
T9	04:20	05:14	01:45
T10	04:14	03:31	01:34
Moyenne	**05:08**	**05:19**	**02:03**

Table 6.5: Temps obtenus par les testeurs experts pour chacun des scénarios.

La première constatation est que notre outil permet une diminution sensible du temps nécessaire à la sécurisation des services autant pour des débutants que des experts.

Les Figures 6.19, 6.20 et 6.21 montrent les écarts de temps pour l'utilisation en tant que débutant et qu'expert de notre outil pour les trois scénarios.

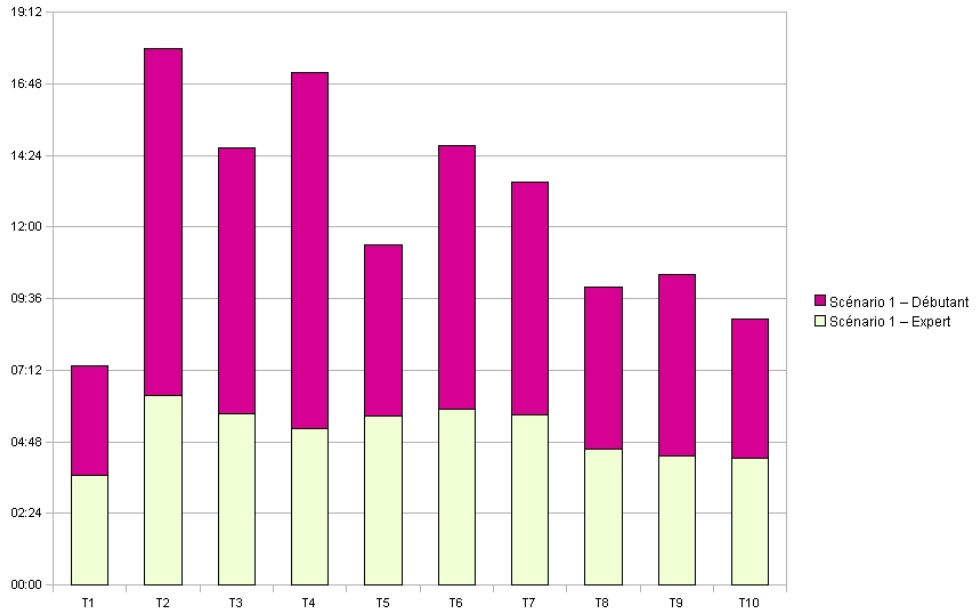


FIGURE 6.19 – Temps obtenus pour le scénario 1 pour des testeurs débutants et experts.

FIGURE 6.20 – Temps obtenus pour le scénario 2 pour des testeurs débutants et experts.

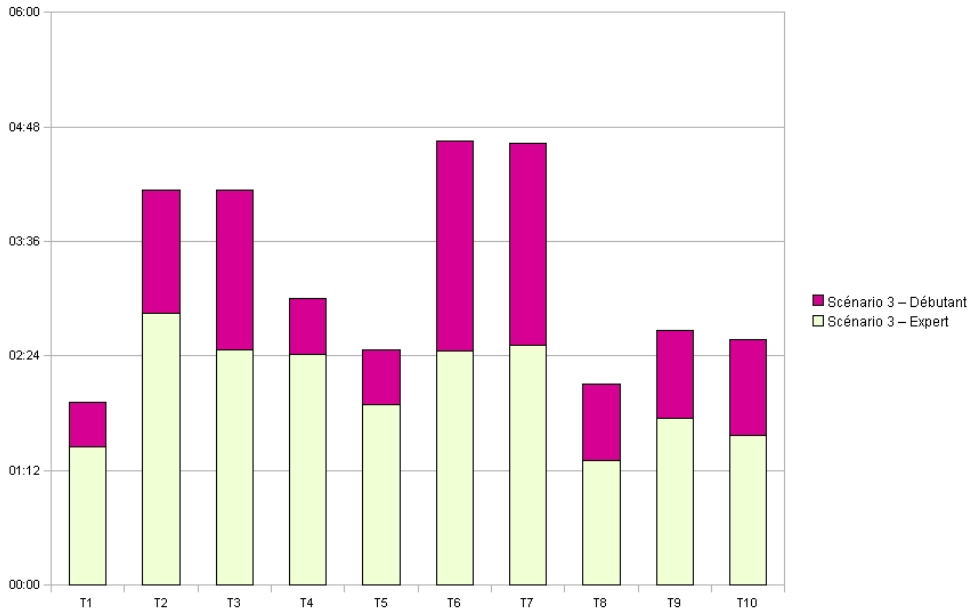


FIGURE 6.21 – Temps obtenus pour le scénario 3 pour des testeurs débutants et experts.

L'écart le plus significatif entre les utilisateurs débutant et expert a été obtenu pour le scénario 1. Nous pouvons expliquer ce résultat par le fait que le scénario 1 a permis aux utilisateurs de découvrir l'environnement de conception. Même si les scénarios 2 et 3 ont été fait par des utilisateurs débutants, ils ont été réalisés après le scénario 1. Les utilisateurs ne maîtrisaient pas l'environnement mais l'avait déjà découvert grâce au scénario 1.

5 Synthèse

Dans ce chapitre, nous avons présenté la plate-forme FOCAS à laquelle nous avons ajouté un ensemble de fonctionnalités. Notre nouvelle plate-forme se nomme Secure FOCAS, c'est un environnement de développement et d'exécution d'orchestration de services abstraits hétérogènes et sécurisés. Nos principaux apports sont :

- la définition des **propriétés de sécurité** au niveau de **l'interface**, puis **la génération du code** associé à ces propriétés de sécurité.
- la définition de **la sélection retardée** des services concrets sécurisés ou non au niveau de **l'interface**, puis **la génération du code** associé.

Pour ces deux extensions, nous avons présenté les onglets que nous avons ajoutés à l'interface :

- un onglet *Non-Functional Properties*, dans lequel l'utilisateur choisit les propriétés de sécurité qu'il souhaite appliquer aux services abstraits ;
- un onglet *Security Details*, dans lequel l'utilisateur précise les éléments concrets nécessaires à l'utilisation des technologies à services qui permettent d'assurer les propriétés de sécurité choisies dans l'onglet *Non-Functional Properties* ;
- un onglet *Selection*, dans lequel l'utilisateur choisit le type de sélection - statique ou dynamique - qu'il a choisi pour les services abstraits.

La génération du code pour la sécurité et la sélection des services se font avec la technologie JET. Nous avons réalisé des fichiers *templates* qui permettent de paramétrer la génération du code en fonction des informations données par l'utilisateur dans les différents onglets.

Ce chapitre présente également un cas d'utilisation, que nous avons développé en collaboration avec Thales pour le projet Européen ITEA SODA. Ce cas d'utilisation représente une chaîne d'acquisition de données. Nous avons réalisé des expérimentations sur ce cas d'utilisation pour valider notre approche. Les résultats de nos expérimentations montrent que la productivité des utilisateurs est améliorée puisque :

- le code nécessaire à la sélection dynamique est entièrement généré ;
- le code de sécurité est également entièrement généré ;
- le temps d'apprentissage de l'outil est bien inférieur au temps d'apprentissage des technologies à services et de sécurité.

7

Extensions et perspectives

Dans une première partie, nous présentons une première extension qui permet de gérer les identités des utilisateurs de l'environnement Secure FOCAS. Nous détaillons ses principes puis son implantation dans notre outil. Dans la deuxième partie, nous développons notre deuxième perspective, qui concerne l'ajout de plusieurs propriétés non-fonctionnelles. Nous détaillons les principes de cet ajout dans notre approche, puis son implantation dans notre outil. Chacune de ces deux parties se terminent par une synthèse.

1 Gestion des identités

Nous proposons une extension de sécurité à notre outil Secure FOCAS : un système de gestion des identités des utilisateurs. Nous allons détailler dans les deux sections suivantes son intégration dans notre approche et sa mise en application dans notre outil. Nous terminerons par les résultats de nos expérimentations.

1.1 Dans notre approche

La gestion des identités, en anglais *Identity Management* [LCGSG09], est une extension que nous avons ajoutée à notre plate-forme Secure FOCAS. La gestion des identités est un élément-clé de la sécurité ; les éléments d'identification des utilisateurs sont gérés en fonction des ressources. Le principe d'un gestionnaire d'identités est de déterminer qui a le droit de faire quoi dans le système. L'utilisateur justifie de son identité en début de session en présentant un élément d'authentification et cette identité lui est suffisante pour accéder aux différentes ressources. Le gestionnaire d'identités masque l'utilisation d'identités différentes pour un même utilisateur pour chacun des sytèmes.

Il existe sur le marché de nombreuses offres de gestionnaire d'identités comme Identity Management [1] d'Oracle, Identity Integration Server [2] de Microsoft, Identity Manager [3] de Novell ou encore Tivoli Identity Manager [4] de IBM. Ces systèmes sont généralement mis en place dans des entreprises pour cacher l'hétérogénéité des systèmes d'identification aux utilisateurs. Ils permettent de gérer les identifications d'un point de vue global par rapport à l'entreprise.

Dans le cadre de notre approche, il est intéressant de mettre en place un tel système puisque les utilisateurs accèdent à des services concrets sécurisés. De plus, nous pouvons noter, que suite aux tests que nous avions réalisés, entrer plusieurs fois des données concernant les mots de passe et les certificats était source d'erreur. Il est difficile pour un utilisateur de retenir beaucoup d'identités, c'est-à-dire ses noms d'utilisateur, ses mots de passe ainsi que ses certificats pour plusieurs systèmes. La deuxième source d'erreur est la faute de frappe dans les données entrées par l'utilisateur. Ce type d'erreur est difficile à détecter.

La Figure 7.1 présente l'architecture utilisée pour la gestion des identités. Les services et les utilisateurs s'enregistrent dans le registre d'identités en donnant les informations de sécurité nécessaires à l'utilisation du service. Ces informations sont identifiées par le couple service/utilisateur. La deuxième étape est l'exécution de l'orchestration dans la plate-forme d'exécution Secure FOCAS. L'exécution d'une orchestration nécessite que l'utilisateur s'identifie auprès de la plate-forme avec son nom d'utilisateur et son mot

1. http://www.oracle.com/technology/products/id_mgmt/index.html
2. https://www.microsoft.com/france/miis/home.mspx
3. http://www.novell.com/products/identitymanager/
4. http://www-01.ibm.com/software/tivoli/products/identity-mgr/

de passe. C'est ce nom d'utilisateur qui est ensuite utilisé pour rechercher ses certificats dans le registre d'identités pour chacun des services appelés lors de l'exécution de l'orchestration. La dernière phase est l'invocation du service concret avec les bons éléments d'identification de l'utilisateur.

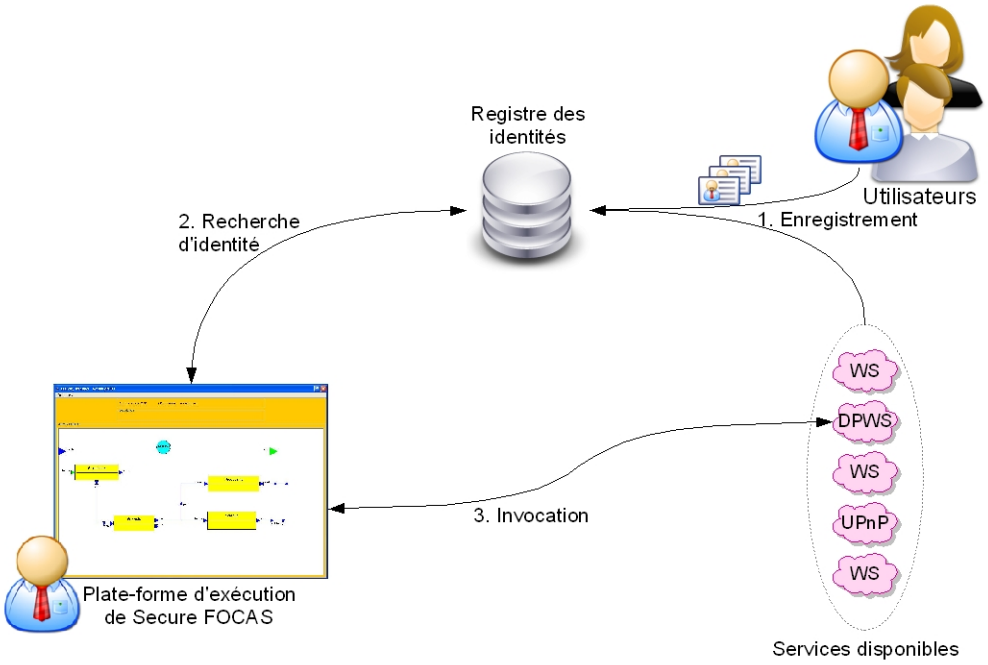


FIGURE 7.1 – Architecture pour la gestion des identités des utilisateurs.

Nous avons implanté un système de gestion des identités basiques. Ce système est un service Web, qui a deux fonctionnalités :

- l'**enregistrement** des services et des utilisateurs,
- la **recherche** des données nécessaires pour prouver l'identité d'un utilisateur particulier pour un service donné.

L'échange des données entre le client Secure FOCAS et le service Web de gestion des identités est particulièrement critique. En effet, les données échangées sont les identités des utilisateurs. Ces échanges peuvent poser des problèmes de sécurité. Nous avons donc choisi de mettre en place des communications sécurisées en suivant la recommandation WS-Security. Nous rendons ces communications confidentielles en les chiffrant.

Dans la suite, nous allons montrer comment ce registre d'identités est utilisé à partir de notre outil Secure FOCAS. En effet, l'utilisateur doit choisir s'il souhaite ou non utiliser le module de gestion des identités, ce qui impacte l'interface de Secure FOCAS ainsi que la génération du code.

1.2 Dans l'outil Secure FOCAS

La gestion des identités a pour but d'éviter à l'utilisateur d'entrer les paramètres de sécurité requis pour utiliser, par exemple, un certificat. Le principe de la gestion des identités est d'ajouter un module qui récupère l'identité de l'utilisateur de l'orchestration et d'utiliser en fonction des besoins ses certificats appropriés aux services appelés. L'ajout d'un tel module impacte l'interface de conception de Secure FOCAS ainsi que la génération du code pour l'exécution. Les deux sections suivantes détaillent cette extension.

1.2.1 L'interface de conception

La gestion des identités nécessite d'ajouter dans l'onglet *Security Details* le choix d'utiliser ce module. L'utilisation de ce module implique que l'utilisateur ne doit pas entrer d'autres informations que l'adresse du registre des identités. La Figure 7.2 est une capture d'écran de l'onglet *Security Details* avec l'extension de gestion des identités.

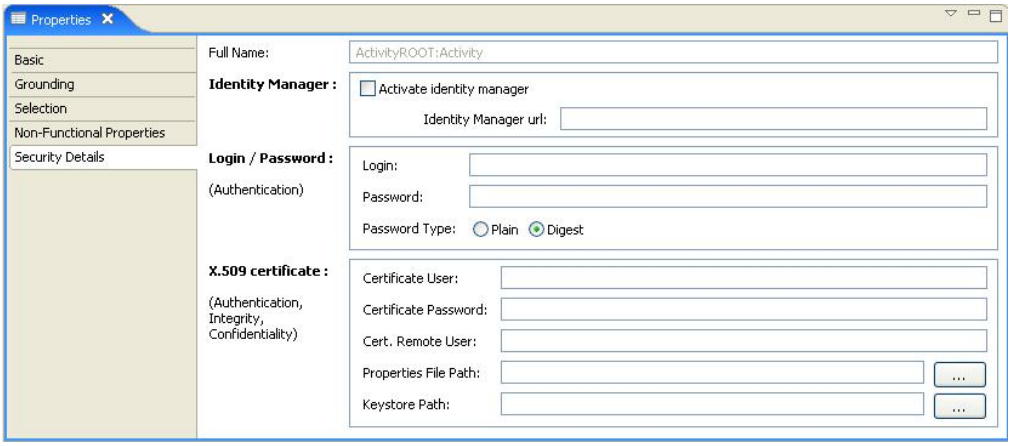


FIGURE 7.2 – Capture d'écran de l'onglet de sécurité détaillée avec la gestion des identités.

1.2.2 La génération du code

L'ajout d'un gestionnaire d'identités impacte la génération du code. En effet, nous devons appeler le gestionnaire d'identités pour chacune des activités qui nécessite une gestion automatisée des identités. Cet appel se fait une fois que le service a été sélectionné. Les informations qui sont retournées par le gestionnaire d'identités sont ensuite utilisées pour configurer les fichiers *proxies*. Nous avons donc dû adapter nos fichiers *templates*, écrits avec la technologie JET, pour ajouter la gestion des identités. La Figure 7.3 est un extrait du code nécessaire pour la recherche des paramètres d'identification et leur utilisation.

```
<% if(isAuthenticationLogin.booleanValue()) { %>
        String kind = lig.adele.focas.identity.service.CallService.LOGINPASS;
<% } else { %>
        String kind = lig.adele.focas.identity.service.CallService.CERTIFICAT;
<% } %>
lig.adele.focas.identity.service.CallService identityCaller =
  new lig.adele.focas.identity.service.CallService(
  "<%=concretePort%>", serviceUrl.toString(), user, kind,
  "<%=identityManagerUrl%>");

identityCaller.findSecuData();
//Config WSDD File
Remote remote = serviceLocator.getPort(
  <%=concreteServicePackage%>.<%=concretePortType%>.class);
Stub axisPort = (Stub) remote;
axisPort = addConfig(axisPort, identityCaller);
service =
  (<%=concreteServicePackage%>.<%=concreteServiceName%>SoapBindingStub)axisPort;
```

FIGURE 7.3 – Extrait du code du fichier *template* pour la recherche d'identité.

1.3 Expérimentations

Nous avons fait le choix d'implanter le registre d'identités avec un service Web. Ce service Web ne comprend que deux fonctionnalités basiques d'enregistrement et de recherche d'identités mais il représente environ 1500 lignes de code.

Par ailleurs, nous avons mesuré le nombre de lignes de code nécessaire pour l'utilisation de ce module de gestion des identités dans la plate-forme Secure FOCAS. Les résultats, que nous avons obtenus pour le cas d'utilisation du système d'alarme, sont présentés dans le Tableau 7.1.

		Sélection statique	**Sélection dynamique**
Lignes de code générées	Métier	1468	1539
	Sécurité	323	324
	Gestion des identités	162	162
Lignes de code écrites	Métier	152	152
	Sécurité	0	0
	Gestion des identités	0	0
TOTAL		**2105**	**2177**

TABLE 7.1 – Lignes de code générées et écrites avec la gestion des identités.

Nous constatons, d'après les résultats obtenus, que les lignes de code pour utiliser le service de gestion des identités sont exclusivement générées. Ceci permet d'améliorer la productivité du développeur qui n'a pas à ajouter l'appel au registre d'identités dans le code fonctionnel.

Le deuxième point intéressant est que le nombre de lignes de code générées est le même quelque soit le type de sélection. Une fois que le service est sélectionné de façon statique ou dynamique, la recherche des éléments d'identification peut être réalisée. La recherche se fait en fonction de l'utilisateur et du service à l'exécution.

1.4 Synthèse

Dans cette partie, nous avons proposé d'étendre notre outil Secure FOCAS en ajoutant un système de gestion d'identités. Avec cette extension, nous avons montré que notre outil s'intègre facilement dans le système d'information d'une entreprise. Il suffit d'adapter l'accès au système de gestion d'identités en fonction de la technologie utilisée.

De plus, l'ajout d'un tel système n'impacte pas le travail des développeurs. Le code qui permet l'accès au gestionnaire d'identités est entièrement généré ; ainsi les développeurs gagnent du temps de développement. Un autre avantage de la génération du code est la diminution du nombre de tâches répétitives souvent source d'erreurs, comme l'appel au gestionnaire d'identités pour chacune des activités qui le requiert.

2 Ajout de nouvelles propriétés non-fonctionnelles

D'après notre expérience pour l'ajout de la sécurité dans la composition de services suivant une approche générative dirigée par les modèles, nous nous sommes intéressé à la possibilité d'appliquer cette même approche à l'ajout de plusieurs propriétés non-fonctionnelles. En effet, nous nous sommes rendu compte que le modèle d'exécution comprenant uniquement des propriétés fonctionnelles et de sécurité n'était pas suffisant pour faire le choix du service concret le plus approprié. Des propriétés de qualité de service peuvent permettre d'affiner le choix lors de la phase de sélection. Nous pensons alors qu'il doit être possible d'exprimer ce type de propriétés au niveau de la spécification.

Dans une première partie, nous présentons une extension de notre approche en prenant en considération plusieurs propriétés non-fonctionnelles. Ensuite, dans une seconde partie, nous détaillons la mise en application de cette approche avec deux nouvelles propriétés non-fonctionnelles : la journalisation d'événements et la qualité de service. La troisième partie concerne l'ajout de ces propriétés dans la plate-forme Secure FOCAS. Dans une dernière partie, nous concluons sur les résultats que nous avons obtenus et les problèmes ouverts que cette approche soulève.

2.1 Dans notre approche

Le but de cette extension de notre approche est de simplifier l'ajout de propriétés non-fonctionnelles pour la composition de services. Suivant notre approche, l'ajout de la sécurité se fait lors de la spécification de la composition et le modèle d'exécution est étendu pour prendre en considération les contraintes de sécurité. Nous allons donc ajouter les propriétés non-fonctionnelles à la spécification de la composition de services et au modèle d'exécution.

Comme pour la sécurité, le code correspondant aux propriétés non-fonctionnelles d'une application est souvent appelé *glue code*. En effet, il est ajouté au code fonctionnel de façon ad hoc. De ce fait, il n'est pas réutilisable. Notre idée est alors de présenter ces propriétés au niveau modèle pour le générer et l'insérer automatiquement au bon endroit dans le code fonctionnel. De plus, en montant en abstraction au niveau modèle, cela nous permet de cacher les détails techniques et technologiques.

2.1.1 Séparation des préoccupations

Nous appliquons le principe de la séparation des préoccupations à notre extension ; nous obtenons alors une composition de modèles. Nous avons ainsi un modèle d'orchestration des services et des modèles de propriétés non-fonctionnelles. Ces modèles sont tous définis séparément. Seuls les modèles des propriétés non-fonctionnelles sont liés au modèle d'orchestration. La Figure 7.4 illustre cette composition.

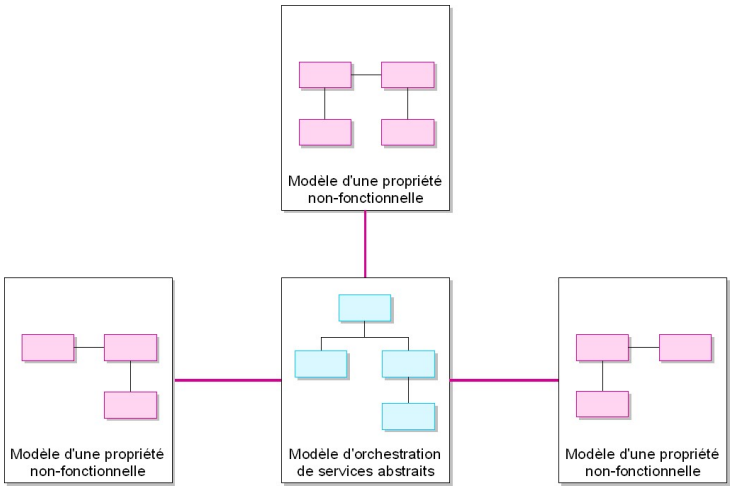


FIGURE 7.4 – Séparations des modèles.

La séparation des préoccupations fait que chaque modèle peut être réalisé par un expert du domaine. La communication entre experts permet de déterminer quels sont les liens possibles entre les modèles. La qualité de la composition est alors améliorée.

Un autre avantage de cette séparation est que les modèles peuvent être maintenus facilement. En effet, l'évolution d'un modèle peut avoir un impact nul ou contrôlé sur les autres modèles. Seuls ses liens avec le modèle d'orchestration doivent aussi évoluer.

2.1.2 Niveau méta-modèle

Pour s'assurer de la cohérence et de la validité de la composition des différents modèles précédemment définis, nous définissons des méta-modèles pour chacun des modèles. Chaque méta-modèle comprend les concepts de son domaine. Le méta-modèle de l'orchestration de services est central, chaque méta-modèle non-fonctionnel est lié par des liens au méta-modèle de l'orchestration. La Figure 7.5 illustre cette composition de méta-modèles.

Les modèles et les liens définis dans la section précédente doivent donc être conformes aux méta-modèles et à leurs liens que nous venons de définir. Les environnements de développement qui prennent en entrée des méta-modèles peuvent guider les développeurs pour la réalisation d'applications. Dans notre cas, les développeurs seront guidés pour la réalisation d'une application basée sur une composition de services avec des propriétés non-fonctionnelles.

L'avantage certain d'une telle séparation des préoccupations est que chaque expert d'un domaine définit un méta-modèle. Les liens entre méta-modèles sont alors spécifiés suite à une communication entre les différents experts. L'expertise de chacun permet d'obtenir des méta-modèles et des liens de bonne qualité.

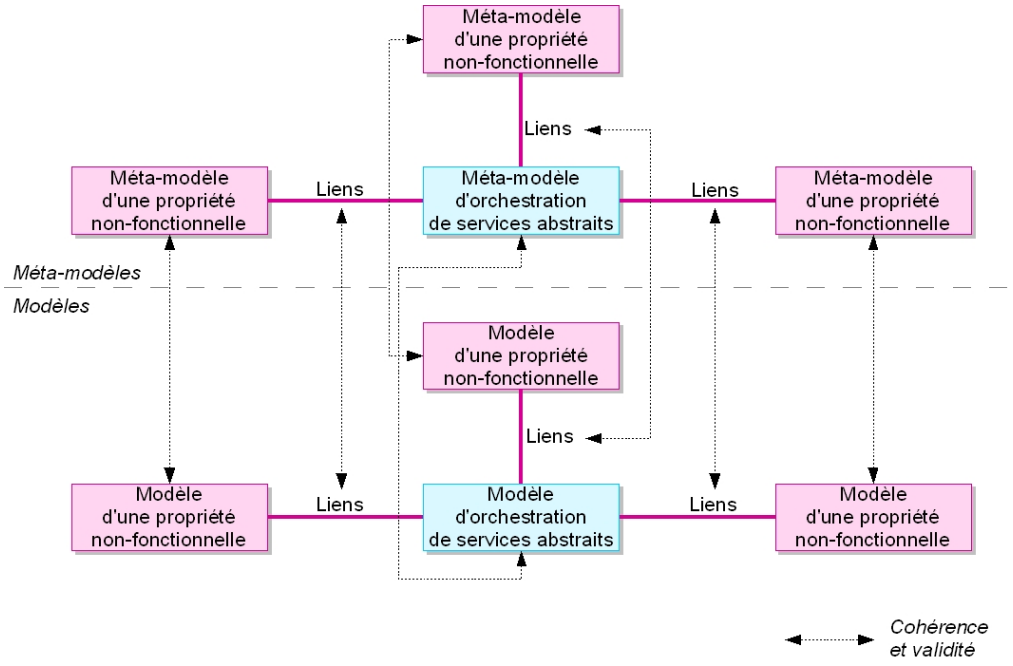


FIGURE 7.5 – Composition des méta-modèles.

2.1.3 Modèle d'exécution

Dans notre approche, le modèle d'exécution est utilisé pour sélectionner le service approprié parmi les services concrets disponibles. Les spécifications comprennent à présent les propriétés fonctionnelles et un ensemble de propriétés non-fonctionnelles. Pour qu'à la sélection le service concret choisi réponde aux spécifications, il faut que les informations concernant les propriétés fonctionnelles et non-fonctionnelles soient contenues dans le modèle d'exécution. Par conséquent, le modèle d'exécution est constitué d'un modèle central comprenant les données fonctionnelles et de plusieurs modèles pour définir les propriétés non-fonctionnelles supportées et/ou requises par le service concret.

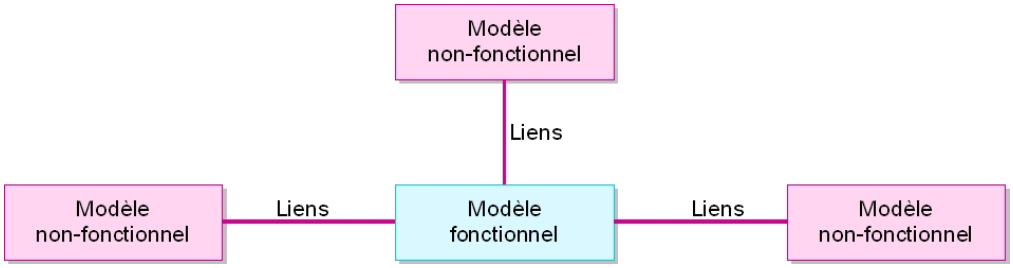


FIGURE 7.6 – Modèle d'exécution.

2.2 Mise en application avec deux propriétés non-fonctionnelles

Nous nous proposons d'appliquer cette extension à deux propriétés non-fonctionnelles supplémentaires que sont la journalisation d'événements et la qualité de service.

2.2.1 Journalisation d'événements

La journalisation d'événements, appelée en anglais *logging*, est une propriété non-fonctionnelle très utilisée lorsque l'on souhaite avoir une connaissance de ce qui se passe globalement ou en un point particulier d'un système. Elle est plus spécifiquement utilisée lorsque l'on souhaite faire de l'audit d'un système. Nous avons d'ailleurs proposé dans le méta-modèle de la sécurité, Figure 5.5 page 136, les solutions techniques de la journalisation comme un moyen de faire de l'audit d'une activité.

Nous présentons, dans la Figure 7.7, la journalisation avec deux implantations possibles, qui sont la journalisation d'événements dans un fichier et dans une base de données.

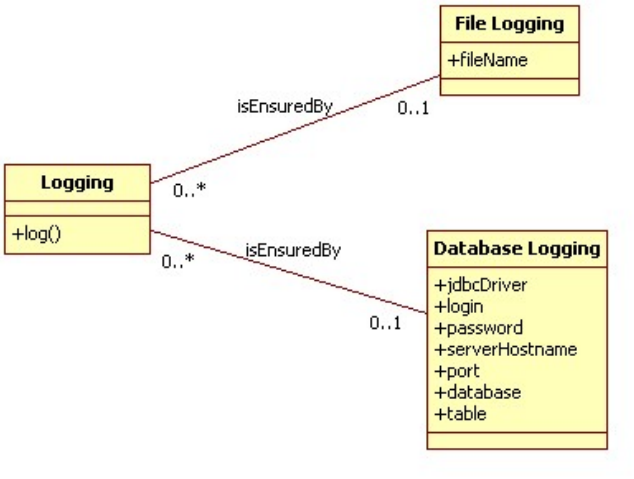


FIGURE 7.7 – Méta-modèle de la journalisation.

Pour la journalisation dans un fichier, seul le nom du fichier est nécessaire. Il n'est pas du ressort de l'utilisateur de choisir le format du fichier. En l'occurrence, nous verrons par la suite que notre système emploie une solution basée sur XML.

En ce qui concerne la journalisation dans une base de données, l'architecte doit donner les informations qui concernent la base qu'il utilise. Il doit donc préciser son type avec le nom du *driver* JDBC [5], par exemple MySQL ou Oracle, son nom d'utilisateur et son mot

5. Pour *Java DataBase Connectivity*, qui est une interface fournie avec Java permettant de se connecter à des bases de données.

de passe, l'adresse et le port du serveur, le nom de la base et la table dans laquelle les informations vont être stockées. De façon identique au stockage des informations dans un fichier, le format de la table n'est pas précisé et les données stockées ne sont pas précisées. Cependant, pour réaliser de la journalisation, il faut avoir au minimum la date et l'événement qui s'est produit.

2.2.2 Qualité de service

La dernière propriété non-fonctionnelle que nous avons commencé à étudier est la qualité de service. Nous pensons que certains des éléments présents dans le modèle d'exécution doivent être exprimés au niveau des spécifications. Le critère de sélection des services doit être définis par l'utilisateur. Nous proposons donc un méta-modèle, Figure 7.8, pour la qualité de service.

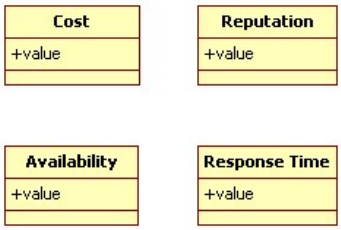


FIGURE 7.8 – Méta-modèle pour la qualité de service.

Notre méta-modèle comporte quatre concepts, qui sont utilisés pour différencier des services ayant les mêmes fonctionalités :

- le **coût** représente ce que le consommateur du service doit payer pour utiliser le service ;
- la **réputation** est mesurée à partir des retours sur les utilisations précédentes du service ;
- la **disponibilité** représente le pourcentage de temps pendant lequel le service fonctionne correctement ;
- le **temps de réponse** correspond au temps que le service met à répondre.

Nous pouvons noter que le coût et la disponibilité sont des informations données par le service alors que la réputation et le temps de réponses sont calculés. Chacun des concepts du méta-modèle ne comprend qu'un attribut. C'est la valeur que le service doit au minimum ou au maximum respecter pour qu'il puisse être sélectionné, puis utilisé. Le méta-modèle, que nous proposons dans cette section, est un début de solution pour gérer au niveau des spécifications la qualité de service. Ce méta-modèle ne contient pas toutes les caractéristiques de la qualité de service mais suffisamment pour tester notre approche.

2.2.3 Composition des différents méta-modèles

Une fois que les méta-modèles ont été spécifiés, il nous reste à définir les liens entre les méta-modèles. Nous présentons successivement la composition de l'orchestration avec la journalisation d'événements, Figure 7.9, puis la composition de l'orchestration avec la qualité de service, Figure 7.10.

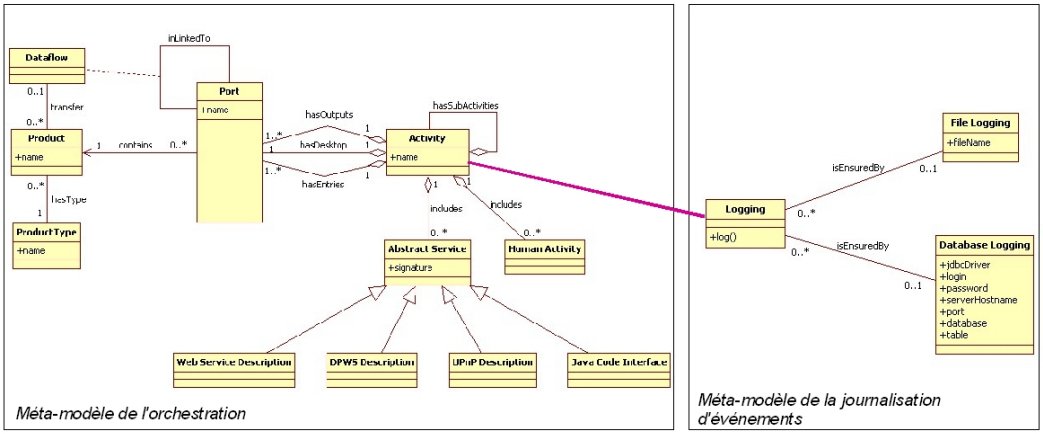


FIGURE 7.9 – Composition des méta-modèles de l'orchestration et de la journalisation d'événements.

Nous avons défini, comme pour l'audit, un lien de cardinalité 0 ou 1 entre le concept *Activité* et le concept *Logging*. La journalisation d'événements peut alors s'appliquer ou non à une activité et par conséquent à un service.

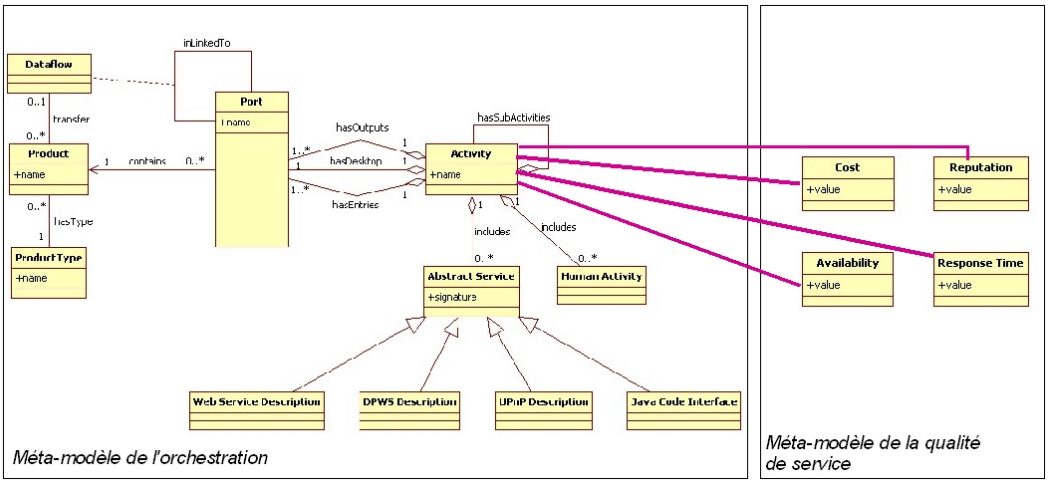


FIGURE 7.10 – Composition des méta-modèles de l'orchestration et de la qualité de service.

Nous avons composé le méta-modèle de l'orchestration avec celui de la qualité de service avec quatre liens de cardinalité 0 ou 1. Pour une activité, on peut choisir ou non son coût, sa réputation, sa disponibilité et son temps de réponse.

Dans les deux paragraphes suivants, nous détaillons la sélection de service et la génération de code.

Sélection de service. La composition du méta-modèle de l'orchestration avec celui de la qualité de service a un impact sur la sélection des services concrets. Le méta-modèle de la qualité de service a pour but d'affiner la sélection. Les services concrets qui sont sélectionnés doivent répondre aux critères de qualité définis par l'utilisateur. Le modèle d'exécution, qui permet d'avoir une connaissance des services concrets disponibles, doit être étendu pour permettre de tenir compte des critères de qualité de service. La Figure 7.11 est le diagramme de classes du modèle d'exécution étendu.

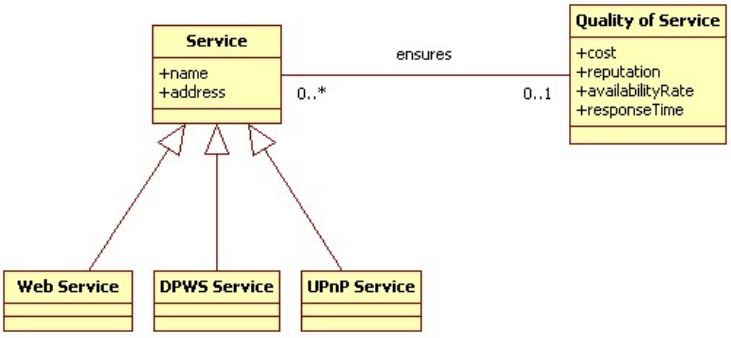


FIGURE 7.11 – Modèle d'exécution avec la qualité de service.

Les données concernant la qualité de service sont renseignées avec les caractéristiques données par les services concrets et aussi avec l'expérience de l'utilisation du service.

Génération du code. Les liens entre l'orchestration de services et la journalisation d'événements ont des conséquences sur la génération du code. Les messages qui sont échangés entre le client et le service sont capturés et stockés dans un fichier XML et/ou dans une base de données. Les informations stockées concernent le contenu du message, sa date d'émission et de réception. La Figure 7.12 présente le principe de la journalisation des messages.

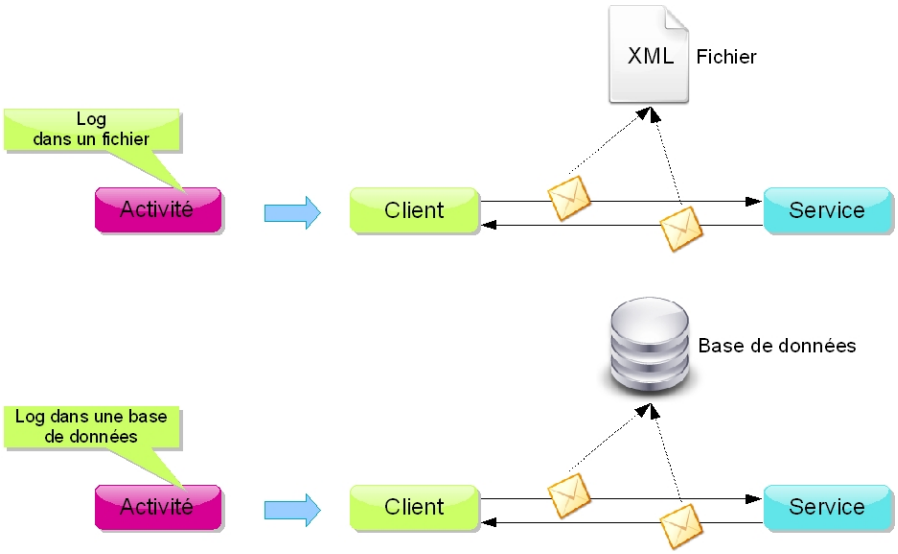


FIGURE 7.12 – Impact de la journalisation d'événements sur la génération de code.

2.3 Ajout dans la plate-forme Secure FOCAS

Nous avons ajouté à notre outil Secure FOCAS la journalisation d'événements en plus de la sécurité. Nous présentons par la suite cette extension que nous avons implantée. Nous détaillons les onglets de l'environnement de conception. Par contre, nous ne décrirons pas les fichiers *templates* pour la génération du code parce qu'ils sont similaires à ceux présentés pour la sécurité.

2.3.1 Onglet général

Nous avons étendu l'onglet général de la sécurité, nommé *Non-Functional Properties*. L'idée est de regrouper tous les concepts de haut niveau de chacune des propriétés non-fonctionnelles dans un même onglet pour avoir une vision globale des caractéristiques appliquées à chacun des services.

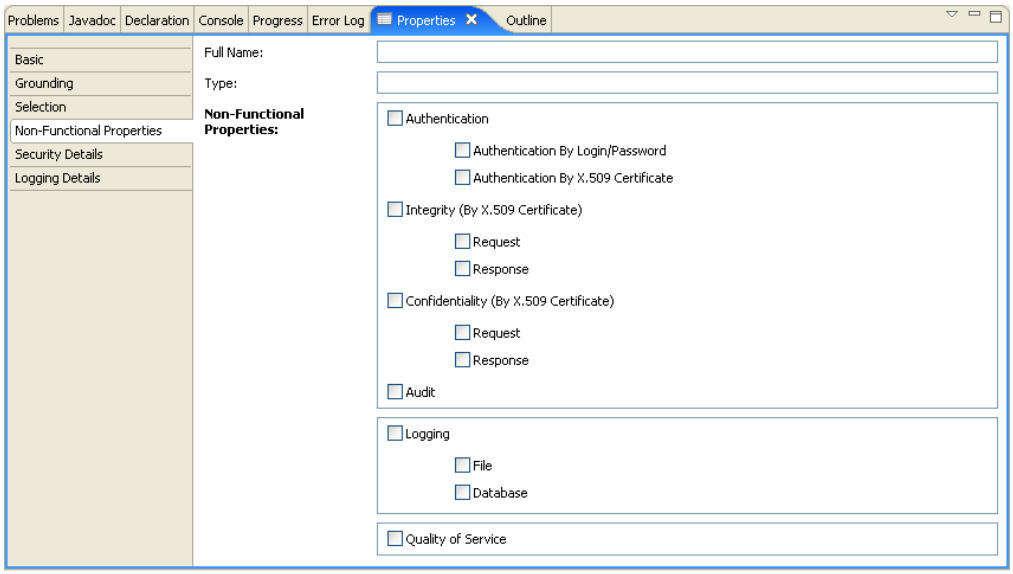


FIGURE 7.13 – Onglet général pour les propriétés non-fonctionnelles.

2.3.2 Onglets pour les caractéristiques concrètes

Nous avons ajouté un onglet pour que l'utilisateur puisse entrer les caractéristiques techniques nécessaires à la journalisation d'événements. La Figure 7.14 est une capture d'écran de cet onglet. Il contient les éléments pour une journalisation dans un fichier et/ou dans une base de données. Cet onglet contient tous les attributs qui ont été définis dans le méta-modèle de la journalisation.

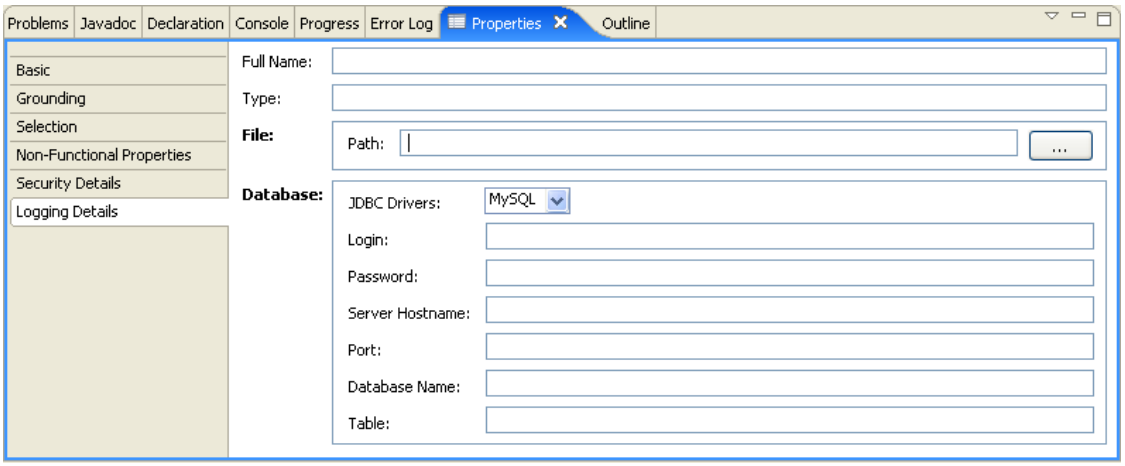


FIGURE 7.14 – Onglet des détails techniques pour la journalisation d'événements.

Pour chaque propriété non-fonctionnelle que l'on souhaite ajouter, nous pensons qu'il faut ajouter un onglet pour que l'utilisateur puisse entrer les informations techniques nécessaires à la génération du code exécutable. Cet onglet permet à l'utilisateur de se concentrer uniquement sur les informations techniques spécifiques au domaine. Seuls les concepts du domaine sont présents dans cet onglet.

2.4 Synthèse

Cette deuxième extension, que nous avons présentée dans cette section, montre que notre approche est généralisable et extensible à d'autres propriétés non-fonctionnelles. Nous l'avons appliqué à deux propriétés non-fonctionnelles : la journalisation des événements et la qualité de service.

La journalisation des événements est une propriété indispensable à mettre en place pour avoir un suivi de ce qui se passe dans un système. Nous avons implanté cette propriété dans Secure FOCAS et nous avons obtenu des résultats intéressants puisque, comme pour la sécurité, l'utilisateur peut entrer, à un certain niveau d'abstraction, les informations concernant le système de stockage. Le code pour réaliser l'enregistrement des événements est alors entièrement généré.

La qualité de service est une propriété qui ne nécessite pas de génération de code particulier. Elle permet uniquement d'affiner les résultats lors de la recherche du service concret approprié aux spécifications.

Cependant, l'ajout de plusieurs propriétés non-fonctionnelles peut poser des problèmes d'interopérabilité. En effet, autant pour la sélection des services que pour la génération du code, il faut déterminer à l'avance quelles sont les propriétés prioritaires. Par exemple, il faut pouvoir déterminer si nous mettons un système de journalisation pour une activité sécurisée ou bien si nous sécurisons une activité dont on suit ses événements. Ce problème ne peut pas être résolu de manière globale, nous sommes contraint de mettre en place une solution ad hoc.

8

Conclusion

L'approche à services est un paradigme qui a été proposé récemment pour faciliter l'intégration d'applications. Un service fournit un ensemble de fonctionnalités grâce à une description de service qui contient les aspects non-fonctionnels qui sont fournis et/ou requis. L'approche à service a pour but de composer les services pour réaliser des applications en intégrant des entités logicielles déjà existantes. L'avantage de cette approche est que seule la description de service est partagée entre les différents acteurs, par conséquent l'architecture de telles applications est faiblement couplée. L'hétérogénéité des plates-formes et des technologies à services sont masquées aux clients des services. Les principes de l'approche à services ont déjà été mis en œuvre par différentes technologies et utilisés dans de nombreux domaines. La réalisation d'application sous la forme de composition de services hétérogènes ou non reste cependant une tâche complexe puisqu'il faut gérer les aspects métier et techniques. En effet, l'application doit répondre aux spécifications mais doit aussi suivre les principes de l'approche à services. De plus, dans la composition de services, il est souvent nécessaire d'ajouter des opérations de médiation pour que les services soient compatibles et/ou assurent certaines propriétés non-fonctionnelles. La sécurité est une propriété non-fonctionnelle particulière qu'il est intéressant d'étudier pour combler les failles de sécurité intrinsèque à l'architecture à services.

Dans cette thèse, nous nous sommes intéressé à la réalisation d'applications qui intègrent des services hétérogènes et sécurisés. En effet, cette problématique soulève aujourd'hui de nombreux défis, tels que :

- la **composition de services hétérogènes** qui nécessite des connaissances approfondies dans chacune des technologies utilisées ;
- la **réalisation d'environnements de conception** pour la composition de services hétérogènes ;
- la **prise en considération de la sécurité** dans les environnements de conception pour éviter les erreurs de développement et pour rendre le code réutilisable.

Comme détaillé dans le chapitre 4, nous proposons une approche pour simplifier la réalisation d'applications à base de services hétérogènes et sécurisés. Notre approche se divise en deux phases (Figure 8.1) :

- une **phase de conception** dans laquelle l'orchestration de services est modélisée en séparant les aspects contrôle et les aspects de sécurité ;
- une **phase d'exécution** qui correspond à l'exécution du code de l'orchestration en fonction du modèle défini lors de la phase de conception et des services disponibles qui répondent aux spécifications.

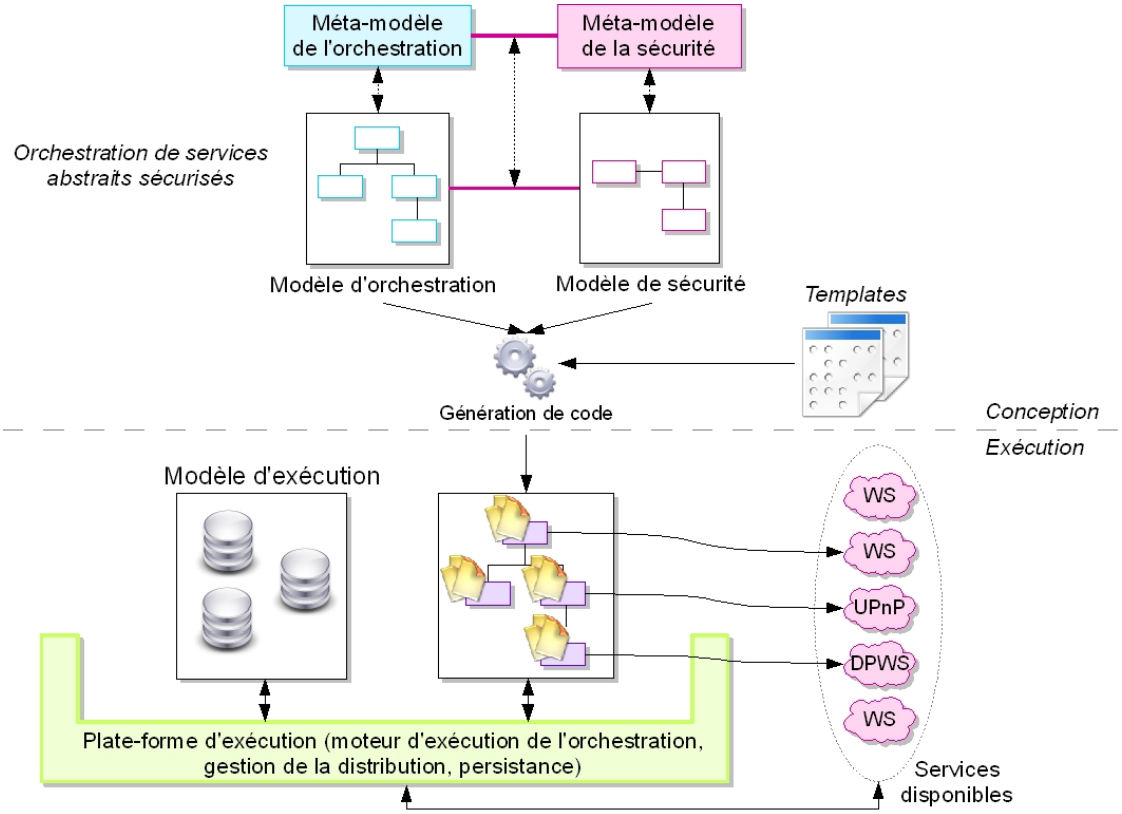


FIGURE 8.1 – Notre approche globale.

Pour la phase de conception, nous avons modélisé les services et leur composition pour **cacher la complexité** technique et technologique de l'approche à services. Pour

gérer la sécurité, nous avons également défini un modèle de la sécurité séparé de celui de l'orchestration. Cette séparation nous permet de réutiliser le modèle de la sécurité pour un autre modèle d'orchestration. Nous avons aussi défini des méta-modèles pour chacun de ces domaines - orchestration de services et sécurité - pour gérer la cohérence et la validité des modèles ainsi que de leur composition.

La phase d'exécution a pour but de générer le code nécessaire à l'exécution et de l'exécuter. La génération se déroule en deux temps :

- une **génération générique** qui permet, à partir d'un ensemble de fichiers *templates* prédéfinis en fonction des technologies à services supportées (Service Web, DPWS, UPnP,...) et choisies lors de la spécification, d'obtenir les fichiers nécessaires à l'appel des services concrets.
- la **spécialisation** des fichiers génériques en fonction des services concrets disponibles et des informations données par l'utilisateur, en particulier les informations de sécurité. L'ensemble des services concrets disponibles sont enregistrés dans le modèle d'exécution.

La phase d'exécution a pour avantage de **gérer le dynamisme** de l'environnement d'exécution, notamment, celui des services grâce au modèle d'exécution.

Dans le chapitre 5, nous avons présenté les deux méta-modèles que nous avons définis pour l'orchestration de services abstraits hétérogènes et pour la sécurité. Le méta-modèle de l'orchestration se base sur le langage de procédés APEL que nous avons étendu au domaine des services et, en particulier, de l'orchestration de services. Le méta-modèle de la sécurité est, quant à lui, défini par niveaux, c'est-à-dire qu'il contient les concepts, puis les techniques de sécurité pour assurer les concepts, et enfin les technologies pour les services qui permettent de mettre en œuvre les techniques. La composition des deux méta-modèles a donné des liens entre les concepts de sécurité et les activités du méta-modèle de l'orchestration. L'expression de ces liens a des impacts au niveau de la sélection des services concrets au moment de l'exécution, mais aussi au niveau de la génération du code fonctionnel et de sécurité.

Notre approche a été validée dans le cadre du projet Européen ITEA SODA en collaboration avec l'entreprise Thales. Nous avons, plus particulièrement, développé un cas d'utilisation, qui nous a permis d'utiliser notre outil pour réaliser une application sous forme d'une composition de services hétérogènes et sécurisés. Le cas d'utilisation nous a également permis de valider notre approche. En effet, les résultats, que nous avons obtenus, ont montré que la productivité des développeurs est améliorée puisque la majorité du code de l'application est généré et en particulier celui de la sécurité ainsi que celui de la sélection des services concrets. De plus, le temps d'apprentissage de notre outil est beaucoup moins long que celui de l'apprentissage des technologies à services et des technologies de sécurité pour les services.

Nous avons proposé, dans le chapitre 7, deux extensions pour notre approche. La première extension est l'intégration d'un gestionnaire d'identités. Nous avons ajouté à l'architecture de notre outil un gestionnaire d'identités qui permet d'avoir pour tous les

utilisateurs possibles l'ensemble de ses informations de sécurité pour les services concrets disponibles. Nous avons implanté notre gestionnaire sous la forme d'un service Web. N'ayant fait aucune hypothèse sur la nature du gestionnaire d'identités, nous pensons qu'il est possible d'intégrer des outils du marché spécialisés dans ce domaine.

La deuxième extension a eu pour but d'élargir notre approche à plusieurs propriétés non-fonctionnelles. Nous avons étendu notre outil Secure FOCAS pour intégrer la qualité de service et la journalisation d'événements. Cette extension a montré qu'il était possible d'intégrer ces deux nouvelles propriétés non-fonctionnelles à condition de gérer explicitement leur composition au niveau du code. La limite de cette extension est que nous ne pouvons pas définir a priori l'ordre de la composition au niveau méta-modèle.

BIBLIOGRAPHIE

[ABPRT04] Daniel Austin, Abbie Barbir, Ed Peters, and Steve Ross-Talbot. Web Services Choreography Requirements 1.0. World Wide Web Consortium, Working Draft WD-ws-chor-reqs-20040311, March 2004.

[AFM05] Marco Aiello, Ganna Frankova, and Daniela Malfatti. What's in an Agreement ? An Analysis and an Extension of WS-Agreement. In *Service-Oriented Computing - ICSOC 2005*, pages 424–436. Springer, 2005.

[Apa06a] Apache. Web Services - Axis, 2006. `http://ws.apache.org/axis/`.

[Apa06b] Apache. Web Services Security for Java, 2006. `http://ws.apache.org/wss4j/`.

[Apa08] Apache Software Foundation. The Apache Tuscany Project, 2008. `http://incubator.apache.org/tuscany/`.

[Ars04] Ali Arsanjani. Service-oriented Modeling and Architecture, November 2004. `http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/`.

[BBB+00] Felix Bachmann, Len Bass, Charles Buhman, Santiago C. Dorda, Fred Long, John Robert, Robert Seacord, and Kurt Wallnau. *Technical Concepts of Component-Based Software Engineering*. Software Engineering Institute, May 2000.

[BBF06] Nelly Bencomo, Gordon S. Blair, and Robert B. France. Summary of the Workshop Models@run.time at MoDELS 2006. In *Lecture Notes in Compu-*

ter Science, Satellite Events at the MoDELS 2006 Conference, pages 226–230. Springer-Verlag, October 2006.

[BEA08] BEA. BEA Aqualogic Service Bus 3.0, March 2008. `http://www.bea.com/content/news_events/white_papers/BEA_AquaLogic_Service_Bus_ds.pdf`.

[BG01] Jean Bézivin and Olivier Gerbé. Towards a Precise Definition of the OMG/MDA Framework. In *ASE '01 : Proceedings of the 16th IEEE international conference on Automated Software Engineering*, pages 273–282, Washington, DC, USA, 2001. IEEE Computer Society.

[BJPW99] Antoine Beugnard, Jean-Marc Jézéquel, Noël Plouzeau, and Damien Watkins. Making Components Contract Aware. *Computer*, 32(7) :38–45, 1999.

[BME+07] Grady Booch, Robert A. Maksimchuk, Michael W. Engel, Bobbi J. Young, Jim Conallen, and Kelli A. Houston. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison-Wesley Professional, April 2007.

[Cer04] Humberto Cervantes. *Vers un modèle à composants orienté services pour supporter la disponibilité dynamique*. PhD thesis, Université Joseph Fourier - Grenoble I, March 2004.

[CH04] Humberto Cervantes and Richard S. Hall. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In *ICSE'04 : Proceedings of the 26th International Conference on Software Engineering*, pages 614–623, Washington, DC, USA, 2004. IEEE Computer Society.

[Con] ContentGuard. XrML The Digital Rights Language for Trusted Content and Services. `http://www.xrml.org/`.

[DEA98] S. Dami, J. Estublier, and M. Amiour. APEL : A Graphical Yet Executable Formalism for Process Modeling. *Automated Software Engg.*, 5(1) :61–96, 1998.

[Des53] René Descartes. *Discours de la méthode (1637)*. Edition Gallimard, 1953. Deuxième Partie, Bibliothèque de la Pléiade.

[Dij74] Edsger W. Dijkstra. On the role of scientific thought. *Selected Writings on Computing : A Personal Perspective*, August 1974. `http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF`.

[Ecl] Eclipse. JET : Java Emitter Templates. `http://www.eclipse.org/modeling/m2t/?project=jet`.

[EH07] Clément Escoffier and Richard S. Hall. Dynamically Adaptable Applications with iPOJO Service Components. In Markus Lumpe and Wim Vanderperren, editors, *6th International Symposium on Software Composition (SC 2007)*, volume 4829 of *Lecture Notes in Computer Science*, pages 113–128. Springer, December 2007.

[EHL07] Clément Escoffier, Richard S. Hall, and Philippe Lalanda. iPOJO : an Extensible Service-Oriented Component Framework. *IEEE International Conference on Services Computing (SCC 2007)*, pages 474–481, July 2007.

[Ei90] Institute O. Electrical and Electronics E. (ieee). IEEE Std 610.12-1990 :IEEE Standard Glossary of Software Engineering Terminology, 1990.

[Esc08] Clément Escoffier. *iPOJO : Un modèle à composant à service flexible pour les systèmes dynamiques*. PhD thesis, Université Joseph Fourier, December 2008.

[Fav04] Jean-Marie Favre. Foundations of Model (Driven) (Reverse) Engineering : Models - Episode I : Stories of the Fidus Papyrus and of the Solarus. In *Post-proceedings of Dagsthul Seminar on Model Driven Reverse Engineering*, 2004.

[FBV06] Marcos Didonet Del Fabro, Jean Bézivin, and Patrick Valduriez. Weaving Models with the Eclipse AMW plugin. In *Eclipse Modeling Symposium, Eclipse Summit Europe 2006*, Esslingen, Germany, 2006.

[FEBF06a] Jean-Marie Favre, Jacky Establier, and Mireille Blay-Fornarino, editors. *L'ingénierie dirigée par les modèles : au-delà du MDA*. Hermes-Lavoisier, Cachan, France, February 2006.

[FEBF06b] Jean-Marie Favre, Jacky Estublier, and Mireille Blay-Fornarino. *L'ingénierie dirigée par les modèles : au-delà du MDA*. Hermes-Lavoisier, Cachan, France, February 2006.

[FK92] David F. Ferraiolo and Richard Kuhn. Role-based access control. In *Proceedings of the 15th NIST-NSA National Computer Security Conference*, pages 554–563, October 1992.

[Fow00] M. Fowler. POJO : An acronym for : Plain Old Java Object, 2000. `http://www.martinfowler.com/bliki/POJO.html`.

[FR07] Robert France and Bernhard Rumpe. Model-driven Development of Complex Software : A Research Roadmap. In *FOSE '07 : 2007 Future of Software Engineering*, pages 37–54, Washington, DC, USA, 2007. IEEE Computer Society.

[FS99] Martin Fowler and Kendall Scott. *UML Distilled : A Brief Guide to the Standard Object Modeling Language (2nd Edition)*. Addison-Wesley Professional, August 1999.

[FT00] Roy T. Fielding and Richard N. Taylor. Principled design of the modern Web architecture. In *ICSE '00 : Proceedings of the 22nd international conference on Software engineering*, pages 407–416, New York, NY, USA, 2000. ACM.

[GHM+05] O. Gruber, B. J. Hargrave, J. McAffer, P. Rapicault, and T. Watson. The Eclipse 3.0 platform : adopting OSGi technology. *IBM Syst. J.*, 44(2) :289–299, 2005.

[HC01] George T. Heineman and William T. Councill. *Component-Based Software Engineering : Putting the Pieces Together (ACM Press)*. Addison-Wesley Professional, June 2001.

[HHKR+07] C. Herrmann, H. Holger Krahn, B. Rumpe, M. Schindler, and S. Völkel. An Algebraic View on the Semantics of Model Composition. In *Model Driven Architecture- Foundations and Applications*, volume 4530 of *Lecture Notes in Computer Science*, pages 99–113. Springer Berlin / Heidelberg, June 2007.

[Hua03] Yan Huang. JISGA : A Jini-Based Service-Oriented Grid Architecture. *Int. J. High Perform. Comput. Appl.*, 17(3) :317–327, 2003.

[IBM04] IBM. Web Services Atomic Transaction (WS-AtomicTransaction), November 2004. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/ws-atomictransaction200411.pdf`.

[IBM05] IBM. WebSphere Process Server, September 2005. `http://www.ibm.com/developerworks/websphere/library/techarticles/0509_kulhanek/0509_kulhanek.html`.

[Jam05] Sonia Jamal. *Environnement de procédé extensible pour l'orchestration - Application aux services Web*. PhD thesis, Université Joseph Fourier, December 2005.

[JMS05] François Jammes, Antoine Mensch, and Harm Smit. Service-Oriented Device Communications Using the Devices Profile for Web Services. In *MPAC'05 : Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, New York, NY, USA, 2005. ACM.

[Kuh06] Thomas Kuhne. Matters of (Meta-) Modeling. *Software and Systems Modeling (SoSyM)*, 5(4) :369–385, December 2006.

[L+06] H. Lockhart et al. Web Services Federation Language (WS-Federation), December 2006. `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP`.

[LCGSG09] Gabriel López, Óscar Cánovas, Antonio F. Gómez-Skarmeta, and Joao Girao. A SWIFT Take on Identity Management. *IEEE Computer*, 42(5) :58–65, 2009.

[LD08] Romain Laborde and Thierry Desprats. Gestion de conditions stables dans XACML : intérêt d'une approche par notification. In Sami Tabbane and Choukair Zied, editors, *Gestion de REseaux et de Services (GRES)*, pages 161–168, `http://www.editions-hermes.fr/`, Décembre 2008. Hermès Science Publications.

[Lud03] Jochen Ludewig. Models in software engineering - an introduction. *Software and Systems Modeling*, 2(1) :5–14, March 2003.

[Mar08] Cristina Marin. *Une approche orientée domaine pour la composition de services*. PhD thesis, Université Joseph Fourier, May 2008.

[Mic05] Microsoft Corporation. Web Services Dynamic Discovery (WS-Discovery), April 2005. `http://schemas.xmlsoap.org/ws/2005/04/discovery/`.

[Mic06] Microsoft Corporation. Devices Profile for Web Services, February 2006. `http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf`.

[MPN98] Manolis Marazakis, Dimitris Papadakis, and Christos Nikolaou. Aurora : An Architecture for Dynamic and Adaptive Work Sessions in Open Environments. In *DEXA '98 : Proceedings of the 9th International Conference on Database and Expert Systems Applications*, pages 480–491, London, UK, 1998. Springer-Verlag.

[MSUW04] Scott Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. *MDA Distilled : Principles of Model-Driven Architecture*. Addison-Wesley Professional, March 2004.

[MTK97] Jeff Magee, Andrew Tseng, and Jeff Kramer. Composing Distributed Objects in CORBA. In *ISADS'97 : Proceedings of the 3rd International Symposium on Autonomous Decentralized Systems*, pages 257–263, Washington, DC, USA, 1997. IEEE Computer Society.

[OAS] OASIS. Web services security x.509 certificate token profile 1.1.

[OAS04a] OASIS. Universal Description Discovery and Integration specification (UDDI) Version 3.0.2, October 2004. `http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm`.

[OAS04b] OASIS. Web Services Security : SOAP Message Security 1.0, March 2004. `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf`.

[OAS05a] OASIS. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf.

[OAS05b] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, February 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[OAS06a] OASIS. Reference Model for Service Oriented Architecture, October 2006. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf.

[OAS06b] OASIS. Web Services Security Kerberos Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf.

[OAS06c] OASIS. Web Services Security Rights Expression Language (REL) Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16687/oasis-wss-rel-token-profile-1.1.pdf.

[OAS06d] OASIS. Web Services Security : SAML Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf.

[OAS06e] OASIS. Web Services Security UsernameToken Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf.

[OAS07a] OASIS. Web Services Business Process Execution Language Version 2.0, April 2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[OAS07b] OASIS. WS-SecureConversation 1.3, March 2007. http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.html.

[OAS07c] OASIS. WS-Trust 1.3, March 2007. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html.

[Obj04] ObjectWeb. The Fractal Project, 2004. http://fractal.objectweb.org/.

[Obj07] Objectweb. JOnAS : Java Open Application Server, 2007. http://wiki.jonas.objectweb.org/xwiki/bin/view/Main/WebHome.

[Obj08] Object Management Group (OMG™). CORBA 3.1, 2008. http://www.omg.org/spec/CORBA/3.1/.

[OMG] OMG. Model Driven Architecture. http://www.omg.org/mda/.

[OMG02] OMG. Meta-Object Facility (MOF™) Specification, version 1.4, April 2002. http://www.omg.org/cgi-bin/doc?formal/2002-04-03.

[OMG09] OMG. Unified Modeling Language™ (UML®), February 2009. http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML.

[OSG07] OSGi™ Alliance. OSGi™ Service Platform Core Specification, Release 4, Version 4.1, April 2007. http://www.osgi.org/download/r4v41/r4.core.pdf.

[OSO07] OSOA. SCA Service Component Architectre, March 2007. http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications.

[Pap03] Michael P. Papazoglou. Service-Oriented Computing : Concepts, Characteristics and Directions. In *WISE'03 : Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12, Washington, DC, USA, 2003. IEEE Computer Society.

[PE08] Gabriel Pedraza and Jacky Estublier. An Extensible Services Orchestration Framework through Concern Composition. In *Proceedings of the 1st International Workshop on Non-functional System Properties in Domain Specific Modeling Languages - NFPinDSML-2008*, Toulouse, September 2008.

[Pel03] Chris Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10) :46–52, 2003.

[PF09] Gabriel Pedraza-Ferreira. *FOCAS : un canevas extensible pour la construction dŠapplications orientées procédé*. PhD thesis, Université Joseph Fourier, October 2009.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[Rea89] C. Reade. *Elements of functional programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2) :38–47, 1996.

[Ser] Service Centric System Engineering (SeCSE). http://www.secse-project.eu/.

[SG96] Mary Shaw and David Garlan. *Software Architecture : Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ, USA, April 1996.

[Suna] Sun Microsystems. Enterprise JavaBeans Technology. http://java.sun.com/products/ejb/index.jsp.

[Sunb] Sun Microsystems. Glassfish : Open Source Application Server. https://glassfish.dev.java.net/.

[Sunc] Sun Microsystems. Java SE Desktop Technologies. http://java.sun.com/javase/technologies/desktop/javabeans/index.jsp.

[Sund] Sun Microsystems. Jini.org. http://www.jini.org/wiki/Main_Page.

[Szy02] Clemens Szyperski. *Component Software : Beyond Object-Oriented Programming, 2nd edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Tay98] David A. Taylor. *Object technology (2nd ed.) : a manager's guide*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[UPn08] UPnP Forum. UPnP™ Device Architecture 1.1, October 2008. http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

[W3C00] W3C. Simple Object Access Protocol (SOAP) 1.1, May 2000. http://www.w3.org/TR/soap/.

[W3C01a] W3C. Canonical XML, March 2001. http://www.w3.org/TR/2001/REC-xml-c14n-20010315.

[W3C01b] W3C. Web Services Description Language (WSDL) 1.1, March 2001. http://www.w3.org/TR/wsdl.

[W3C02a] W3C. Exclusive XML Canonicalization, July 2002. http://www.w3.org/TR/xml-exc-c14n/.

[W3C02b] W3C. XML Encryption Syntax and Processing, December 2002. http://www.w3.org/TR/xmlenc-core/.

[W3C04a] W3C. Web Services Addressing (WS-Addressing), August 2004. http://www.w3.org/Submission/ws-addressing/.

[W3C04b] W3C. Web Services Architecture, February 2004. http://www.w3.org/TR/ws-arch/.

[W3C04c] W3C. Web Services Choreography Description Language Version 1.0, December 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.

[W3C05a] W3C. Web Services Eventing (WS-Eventing), March 2005. http://www.w3.org/Submission/WS-Eventing/.

[W3C05b] W3C. XML Key Management Specification (XKMS 2.0), June 2005. http://www.w3.org/TR/2005/REC-xkms2-20050628/.

[W3C07] W3C. Web Services Policy 1.5 - Framework, September 2007. http://www.w3.org/TR/ws-policy/.

[W3C08a] W3C. Web Services Metadata Exchange (WS-MetadataExchange), August 2008. http://www.w3.org/Submission/2008/SUBM-WS-MetadataExchange-20080813/.

[W3C08b] W3C. XML Signature Syntax and Processing (Second Edition), June 2008. http://www.w3.org/TR/xmldsig-core/.

[WD01] Roel Wuyts and Stéphane Ducasse. Composition languages for black-box components. In *Proceedings of the First OOPSLA Workshop on Language Mechanisms for Programming Software Components*, pages 33–36, October 2001.

[WWJ04] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *FMSE '04 : Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 45–55, New York, NY, USA, 2004. ACM Press.

[YM05] Mamoon Yunus and Rizwan Mallal. An Empirical Study of Security Threats and Countermeasures in Web Services-Based Services Oriented Architectures. In *Web Information Systems Engineering - WISE 2005*, pages 653–659, November 2005.

[ZBBG07] Elmar Zeeb, Andreas Bobek, Hendrik Bohn, and Frank Golatowski. Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services. In *AINAW '07 : Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 956–963, Washington, DC, USA, 2007. IEEE Computer Society.