Une approche déclarative pour la gestion de la qualité de service dans les compositions de service

Fabien Baligand

ED n°82 : Sciences et Technologies de l'Information et de la Communication

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de
Docteur de l'Ecole des Mines de Paris
Spécialité "Informatique temps réel, robotique et automatique"

présentée et soutenue publiquement par
Fabien BALIGAND

le 25 juin 2008

Une Approche Déclarative pour la Gestion de la Qualité de Service dans les Compositions de Services

Directeur de thèse : Pierre Cointe

Jury

M. MALENFANT Jacques	Université Pierre et Marie Curie, Paris	Président
M. SEINTURIER Lionel	Université de Lille	Rapporteur
M. CONSEL Charles	ENSEIRB, Bordeaux	Rapporteur
M. COINTE Pierre	Ecole des Mines de Nantes	Examinateur
M. LEDOUX Thomas	Ecole des Mines de Nantes	Examinateur
M. RIVIERRE Nicolas	France Télécom, Issy Les Moulineaux	Examinateur

UNE APPROCHE DÉCLARATIVE POUR LA GESTION DE LA QUALITÉ DE SERVICE DANS LES COMPOSITIONS DE SERVICES

A Declarative Approach for Quality of Service Management in Service Compositions

Fabien Baligand

Ecole Nationale Supérieure des Mines de Paris

Remerciements

Je souhaite tout d'abord remercier Lionel Seinturier et Charles Consel pour avoir été rapporteurs de cette thèse, ainsi que Jacques Malenfant pour avoir accepté d'être membre du jury.

Je tiens également à remercier mon directeur de thèse, Pierre Cointe, ainsi que l'ensemble des membres de l'équipe OBASCO avec lesquels j'ai pu partager des discussions enrichissantes lors de mes passages à Nantes. Mes remerciements vont aussi vers les membres de l'équipe MAPS-AMS-VIP de France Telecom R&D à laquelle j'ai été rattaché durant ma thèse.

Je souhaite ensuite exprimer ma reconnaissance à mes deux encadrants de thèse, Thomas Ledoux et Nicolas Rivierre, qui m'ont apporté le soutien scientifique et les encouragements sans lesquels ces travaux de thèse n'auraient pas pu s'effectuer. Avec patience et pédagogie, ils m'ont fait découvrir plusieurs facettes de l'activité de chercheur. Je souhaite leur dire que j'ai eu beaucoup de plaisir à travailler avec eux et que cette thèse n'aurait pu être ce qu'elle est sans leurs remarques et leurs conseils judicieux. De plus, je remercie Nicolas grâce à qui j'ai pu participer au projet RNTL FAROS. J'ai également une pensée pour Didier Le Botlan avec qui j'ai eu le plaisir de travailler en début de thèse.

Enfin, mes sentiments les plus chaleureux sont pour ma famille et mes amis. En particulier, je remercie mes parents qui m'ont toujours soutenu dans mes choix et qui m'ont toujours encouragé depuis que je suis tout petit. Avec une infinie patience et tendresse, ils m'ont toujours aidé à me surpasser. Parmi mes amis, j'ai une pensée particulière pour Anneli Lenica et Aurélien Moreau avec qui j'ai partagé mes trois années de thèse. Sans leur amitié, la thèse n'aurait jamais été l'expérience qu'elle a été. J'ai également une pensée spéciale pour Alexis Arragon, Benoit Doumas, Rebecca Interrante, Dhia Daoued, Juan Navas, Marion Didier, Mickael Meulle, Marie Moreau, François Horn, Claire Perdigou, Yasmine Charif et tous les autres, qui comptent tellement pour moi.

Sommaire

Table des figures

Liste des tableaux

Liste des listings

Chapitre 1

Introduction

1.1 Evolution des architectures logicielles

La prolifération et la complexification des systèmes d'information poussent les entreprises à rationaliser leurs développements logiciels. Pour cela, une des caractéristiques recherchées pour la conception de systèmes logiciels est l'*agilité*, c'est à dire la capacité d'une architecture à évoluer afin d'intégrer certains changements. Par exemple, il existe des changements « organisationnels » (avec la création de filiales ou la restructuration des organisations en place), des changements « métier » (c'est à dire relatifs à l'évolution de l'activité tels que la mise en oeuvre d'une gestion de relation clients), des changements dûs aux évolutions de la réglementation (tels que l'ouverture à la concurrence des marchés publics), des évolutions technologiques (prise en compte du *legacy*) ou encore la maîtrise des coûts (réduction des délais de « time to market »).

Parce que les systèmes informatiques sont amenés à évoluer de manière certaine, leur agilité constitue une exigence majeure. Les architectures logicielles doivent donc promouvoir une réelle flexibilité et réutilisabilité pour s'adapter au changement. De multiples paradigmes et techniques de programmation logicielle ont émergé pour répondre à cette problématique. C'est notamment dans ce contexte que l'approche objet [Coi06] puis le paradigme de composant ont fait leur apparition. L'approche objet a apporté un ensemble de principes de programmation (tel que l'encapsulation), tandis que le paradigme de composant a permis le regroupement cohérent et réutilisable des objets. Se définissant comme une « unité de composition dont les interfaces sont spécifiées, et qui peut être déployée indépendamment du reste de l'application », le composant a pour caractéristiques fondamentales l'encapsulation forte, la contractualisation des interactions et la composition par des tiers [Szy98]. De nombreux modèles de composant ont ainsi émergé aussi bien dans l'industrie que dans le milieu académique (J2EE, CORBA, COM+ ou encore FRACTAL).

Parallèlement à l'évolution des architectures d'applications à base de composants, l'émergence d'Internet a permis de mettre en place un environnement hautement ubiquitaire favorisant l'accès permanent à n'importe quelle ressource, et ce depuis n'importe quel point d'accès. Cependant, Internet n'a pas été conçu à son origine comme une infrastructure pour les applications distribuées riches, ce qui le rend initialement impropre à l'utilisation d'une vaste classe d'applications industrielles qui requièrent certaines propriétés de sécurité, de robustesse ou encore de transaction. Le paradigme de « service »

vise à permettre au concepteur d'architectures réparties sur Internet de manipuler des entités logicielles d'une granularité équivalente à celle des composants et offrant un couplage lâche entre le client et le fournisseur de service. En particulier, un service fournit un point d'accès vers une ressource logicielle se trouvant sur Internet et garantit un haut niveau d'interopérabilité entre plateformes en réalisant l'abstraction de toutes les informations liées au traitement de cette ressource. Avec ce nouveau paradigme, émerge un nouveau type d'architecture, appelées « Architectures Orientées Service » [SN98], qui exploitent la propriété de couplage faible des services et permettent de composer de manière flexible et réutilisable des services. Ces architectures visent à répondre de manière pertinente aux challenges liés à l'agilité des systèmes d'information.

1.2 Motivations liées à la gestion de la Qualité de Service

Avec l'émergence des services dans les architectures logicielles, apparait un nouveau modèle relationnel entre les acteurs du monde informatique puisque les clients deviennent dorénavant de véritables consommateurs de service. En effet, grâce à la propriété de couplage faible, les Architectures Orientées Service simplifient la procédure de découverte et d'utilisation des services, permettant ainsi aux clients de substituer aisément un service par un autre selon leur convenance. D'autre part, la prolifération de services aux fonctionnalités sensiblement équivalentes sur Internet offre un vaste choix de produits auxquels les clients peuvent faire appel ainsi qu'une concurrence accrue pour les fournisseurs de service. En outre, le caractère ubiquitaire d'Internet rend plus opaque l'identité des divers acteurs. Par conséquent les acteurs réclament certaines garanties sans lesquels un client peut être amené à dialoguer avec un service dont les ressources varient significativement, ou bien inversement un fournisseur peut voir son service subitement utilisé par un nombre important de clients et peut ne plus pouvoir faire face à leurs demandes.

Dans ce contexte relationnel hautement dynamique, la notion de QdS [1], qui s'intéresse à la qualité de la relation entre un service et son client, devient un enjeu crucial. Plus précisement, cette notion [ISO, ISO98] fait référence à de multiples propriétés telles que la performance, la disponibilité, la robustesse ou encore la sécurité. Dans le contexte des Architectures Orientées Service où les acteurs se connaissent peu, la garantie de ces propriétés est tout aussi fondamentale que le traitement fonctionnel du service. Par exemple, un service permettant de passer des ordres d'achat/vente d'actions en bourse serait tout aussi inutilisable s'il modifiait les ordres donnés par les clients que si ses performances étaient faibles. Ainsi, à l'instar des traitements métiers, les caractéristiques de QdS des services doivent faire rencontrer l'offre et la demande.

Toutefois, contrairement aux standards bien établis du domaine fonctionnel (tels que SOAP [SOA] ou WSDL [WSD]), il n'existe pas de spécification ni de mécanisme officiellement reconnu par la communauté service en ce qui concerne la spécification et le traitement de la QdS des services. Cependant, poussée par les enjeux liés à la QdS, la communauté service a fait émerger plusieurs travaux permettant la réalisation et la mise en oeuvre de contrats de QdS entre un client et un service. Ces travaux ont abouti à la spécification de divers modèles de contrat, de protocoles de gestion associés et d'infrastructures (tels que le modèle WS-AGREEMENT [ACD+07] avec la

1. Qualité de Service

plateforme CREMONA [LDK04]). Ainsi, un client souhaitant avoir des garanties de performance ou encore de sécurité d'un service peut établir un contrat de service avec son fournisseur.

1.3 Limitations actuelles

La propriété de couplage faible des services facilite leur composition. En particulier, la création aisée et flexible de services de plus haut niveau par composition de services existants est une capacité fondamentale des Architectures Orientées Service. Pour cela, des langages tels que BPEL4WS [2] [Dub02] et des plateformes éprouvées comme les moteurs d'orchestration sont apparus pour la spécification et la mise en oeuvre des compositions de services. Cependant l'expressivité du langage BPEL4WS vise uniquement les aspects fonctionnels des compositions et ne prend pas en compte la gestion de la QdS. Or, il n'existe ni langage ni infrastructure pour la gestion de la QdS dans les compositions de services.

Les enjeux liés à la garantie de QdS des compositions de services sont aussi importants que les enjeux liés à l'assemblage fonctionnel des services. En particulier, la gestion de la QdS des compositions de services montre des limitations lors des étapes statique et dynamique de gestion des compositions de services :

Traitement statique Le traitement statique de la QdS s'effectue à un moment antérieur au déploiement de la composition et a pour enjeu de garantir les propriétés de QdS de l'assemblage (*i.e.* du service composite issu de la composition). Ce traitement suppose d'une part d'être capable de trouver un ensemble de services dont les offres de QdS satisfont aux exigences de QdS potentiellement portées par les divers éléments de la composition, et d'autre part de pouvoir déduire la QdS de l'assemblage à partir des offres provenant des services de la composition. En ce qui concerne la sélection des services entrant dans la composition [ZBN+04], si les services fonctionnellement équivalents restent encore relativement peu nombreux sur Internet, l'émergence de « l'orientation service » devrait permettre à moyen terme la création de vastes écosystèmes de services qui fourniront un choix conséquent d'offres. Dans un tel contexte, la sélection de multiples services entrant dans la composition parmi des ensembles de services aux multiples offres, chacune possédant de multiples contraintes de QdS, donne lieu à une explosion combinatoire. En ce qui concerne le problème de garantie de la QdS de l'assemblage en fonction de la QdS des services sous-jacents, des méthodes appropriées permettant d'agréger efficacement la QdS des services de la composition [CSM+04] sont requises. En outre, certains paramètres inconnus au pré-déploiement *a priori* (tels que le nombre d'itérations de certaines activités ou encore la sélection de certains chemins pour l'exécution de la composition) doivent être pris en compte [CPE+06]. Il se peut également que certains services n'exposent pas explicitement leur QdS, auquel cas des estimations de QdS doivent pouvoir être utilisées au lieu des offres.

Les architectes manquent de solutions pour effectuer leur vérification de QdS et la sélection des services. Les solutions proposées par les travaux récents [ZBN+04, CSM+04] sont également limitées quant à l'expressivité de la QdS qu'elles permettent de spécifier. En effet, les compositions de services sont souvent vues par ces méthodes telles des

2. Business Process Execution Language for Web Services

boîtes noires et il n'est pas possible d'envisager leur application sur certaines parties de la composition uniquement. Ces limitations font qu'il est souvent plus intéressant pour un architecte de sélectionner les services et d'effectuer les calculs de QdS manuellement.

Traitement dynamique Le traitement dynamique de la QdS s'effectue lors de l'exécution de la composition de services. A l'instar du traitement statique, il s'agit de garantir les propriétés de QdS de l'assemblage, la différence étant qu'à cette étape, les services ont déjà été identifiés. Cependant, à l'exécution, certains paramètres sont susceptibles de varier tels que la QdS des services appartenant à la composition. Or, si le service composite ne se montre pas capable de maitriser ses capacités de QdS, alors le client pourra préférer utiliser le service d'un concurrent. Il est donc fondamental de pouvoir observer les paramètres de QdS des diverses parties de la composition et de pouvoir réagir afin de préserver la QdS globale garantie initialement. La détection des variations de QdS fait appel à des mécanismes de type *monitoring* tandis que l'adaptation peut se traduire par une replanification de certains services de la composition. A l'exécution, seules des techniques de planification automatisées et efficaces en termes de performance peuvent être envisagées pour le traitement dynamique de la QdS. Or, les moyens de spécifier et de mettre en oeuvre l'adaptation de la QdS dans les compositions de services sont encore relativement limités et ne proposent souvent que des techniques de replanification de services soit globales soit locales [ZBN+04].

En outre, lors de l'exécution des compositions de services, un certain nombre de mécanismes liés à la QdS (tels la sécurité ou la garantie de livraison) sont spécifiés dans les contrats de QdS et doivent être mis en oeuvre. Or, seules quelques plateformes adressent la gestion de ces mécanismes [CM04, TEM07] avec des limitations comme le fait que ces mécanismes soient figés à l'exécution ou encore que ces plateformes n'adressent pas les propriétés de type performance. En outre, ces plateformes permettant de gérer la QdS des compositions de services à l'exécution sont souvent fortement couplées au moteur d'orchestration ce qui a pour effet de limiter significativement l'évolutivité et la réutilisabilité des diverses plateformes.

1.4 Objectifs de la thèse

Dans cette thèse, notre objectif est de proposer un nouveau pouvoir d'expression afin de répondre à la problématique de la gestion de la QdS dans les compositions de services. Pour cela, nous souhaitons concevoir un langage qui encapsule les diverses informations et mécanismes liés au domaine de la gestion de QdS dans le contexte des compositions de services. Ce langage doit permettre à un nouvel acteur, appelé « Architecte Intégrateur », de rédiger des spécifications liées à la gestion statique et dynamique de la QdS d'une composition de services.

Nous souhaitons que ce langage prenne en compte une large variété de traitements liés à la gestion de la QdS de manière à ce que l'architecte intégrateur puisse adresser à la fois des préoccupations liées à la sélection des services, aux mécanismes de QdS (sécurité, garantie de livraison, etc.), ou encore au *monitoring* de la QdS à l'exécution. En particulier, ce langage doit rendre possible la gestion de la QdS sur des parties plus ou moins restreintes de la composition, permettant ainsi de mieux rationaliser le traitement de la QdS.

Notre objectif est également de proposer une approche non intrusive pour la mise en oeuvre de ce langage. En effet, le langage BPEL et les plateformes d'orchestration ont été élaborés avec un effort global de standardisation qui a permis l'émergence des Architectures Orientées Service. Notre approche ne doit donc pas altérer le langage BPEL ni modifier les plateformes d'orchestration. Cette mise en oeuvre doit être capable de prendre en compte l'ensemble des spécifications statiques et dynamiques liées à la gestion de la QdS que peut autoriser notre langage. Nous souhaitons également proposer par le biais de notre langage et de sa mise en oeuvre une nouvelle méthode de planification des services.

1.5 Contributions

Langage QoSL4BP Notre contribution principale est le langage dédié QoSL4BP [3] qui encapsule le domaine de la gestion de QdS relatif à notre étude. Ce langage fait apparaître dans son modèle de données et ses primitives, les données du domaine ainsi que les traitements mis en évidence pour gérer la QdS des compositions de services. Le langage QoSL4BP permet de structurer des spécifications sous la forme de politiques comportant une cible (région d'application de la politique sur la composition de services), une section regroupant les traitements statiques de QdS mis en oeuvre au pré-déploiement de la composition et une section regroupant les traitements dynamiques de QdS mis en oeuvre à l'exécution de la composition. Ce langage offre des propriétés intéressantes pour le domaine de la gestion de QdS telles que la synchronisation des traitements entre gestion de QdS et exécution de la composition de services, la prise en compte des exceptions ou encore la finitude des traitements.

Plateforme ORQOS Pour la mise en oeuvre du langage QoSL4BP, nous avons fait le choix d'une approche n'affectant pas le moteur d'orchestration. Ainsi, nous proposons une plateforme spécifiquement en charge de QoSL4BP et permettant d'effectuer les traitements statique et dynamique liés à la gestion de la QdS. Dans le but que les deux langages puissent se synchroniser à l'exécution de la composition, la composition BPEL est transformée au pré-déploiement afin d'intégrer des indirections vers la plateforme ORQOS. Cette approche vise à privilégier un mélange des préoccupations (composition de services et gestion de la QdS) de la manière la moins intrusive possible. Il est donc possible d'utiliser la plateforme ORQOS avec les divers moteurs BPEL existants.

1.6 Organisation du document

La première partie du manuscrit est consacrée au contexte de l'étude. Au chapitre 2, nous effectuons une étude des concepts et des technologies sous jacents à l'orientation service et au génie logiciel. Au chapitre 3, nous analysons les travaux de l'état de l'art sur la gestion de la QdS dans les compositions de services afin d'identifier leurs points forts ainsi que leurs limitations.

Dans la seconde partie du manuscrit, nous présentons la contribution de cette thèse. Dans le chapitre 4, nous présentons les motivations qui nous ont poussé à faire le choix

3. Quality of Service Language for Business Processes

d'un langage et d'une approche non intrusive. Un exemple fil conducteur y est également présenté. Dans le chapitre 5, nous présentons QoSL4BP, notre langage dédié à la gestion de la QdS dans les compositions de services. Ce chapitre expose la conception de ce langage avec son modèle de données, ses traitements, sa structure et ses propriétés. La chapitre 6 présente le modèle d'exécution du langage QoSL4BP, c'est à dire les traitements effectués lors de l'interprétation et de l'exécution du langage. Dans ce chapitre, sont présentées la planification des services, la mise en oeuvre dynamique du langage ainsi que la plateforme ORQOS qui réalise concrètement les diverses opérations de manière non intrusive avec les plateformes d'exécution des compositions BPEL.

Enfin, nous effectuons l'évaluation de notre approche. Le chapitre 7 déroule un scenario de composition de services inspiré du domaine médical. La gestion de la QdS de cet exemple est mise en oeuvre avec notre approche et nous validons notre approche à travers cette étude de cas. Nous concluons et présentons les perspectives de notre étude dans le chapitre 8.

1.7 Liste des publications liées à cette thèse

Conférences internationales avec comité de lecture

[1] Fabien Baligand, Thomas Ledoux, and Pierre Combes. Une approche pour garantir la qualite de service dans les orchestrations de services web. In *7eme Conference Internationale sur les NOuvelles TEchnologies de la REpartition (NOTERE 2007)*, June 2007.

[2] Fabien Baligand and Valerie Monfort. A concrete solution for web services adaptability using policies and aspects. In *ICSOC '04 : Proceedings of the 2nd international conference on Service oriented computing*, pages 134–142, New York, NY, USA, 2004. ACM.

[3] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux. A declarative approach for qos-aware web service compositions. In Bernd J. Kraemer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Fifth International Conference on Service-Oriented Computing (ICSOC)*, volume 4749 of *LNCS*, pages 422–428. Springer, 2007.

Ateliers internationaux avec comité de lecture et publication des actes

[4] Fabien Baligand, Didier Le Botlan, Thomas Ledoux, and Pierre Combes. A language for quality of service requirements specification in web services orchestrations. In Dimitrios Georgakopoulos, Norbert Ritter, Boualem Benatallah, Christian Zirpins, George Feuerlicht, Marten Schonherr, and Hamid R. Motahari Nezhad, editors, *ICSOC Workshops*, volume 4652 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2006.

Projet RNTL FAROS

[5] Fabien Baligand and Nicolas Rivierre. Chapitre ORQOS dans livrable FAROS : Identification des modalites de prise en charge des contrats pour chaque plate-forme cible. Technical Report F.3.1, RNTL FAROS, June 2007.

[6] Fabien Baligand and Nicolas Rivierre. Chapitre ORQOS dans livrable FAROS : Integration de contrats dans les plates-formes. Technical Report F.3.2, RNTL FAROS, December 2007.

[7] Anne-Francoise Le Meur, Fabien Baligand, Mireille Blay-Fornarino, Philippe Collet, and Nicolas Rivierre. Chapitre service dans livrable FAROS : Etat de l'art sur la contractualisation et composition. Technical Report F.1.1, RNTL FAROS, October 2006.

Première partie

Contexte de l'Étude

Chapitre 2

Contexte logiciel spécifique à la thèse

Dans ce chapitre, nous présentons un certain nombre de domaines liés à nos travaux. Tout d'abord la section 2.1 présente les concepts liés aux Architecture Orientée Service et précise les notions de service ainsi que de composition de services. La section 2.2 présente les principes et méthodes liés à la séparation des préoccupations qui constitue un élément fondamental pour notre étude. Enfin, la section 2.3 traite de la QdS liée aux services.

Sommaire

2.1 Architecture Orientée Service

2.1.1 Objet, composant et service

Objet L'ingénierie logicielle s'est fixée pour mission de concevoir des approches permettant de rationaliser la production des systèmes logiciels. Parce qu'ils évoluent, se complexifient et deviennent plus distribués, ces systèmes requièrent des outils de conception et de développement assurant qualité et fiabilité, tout en offrant plus de flexibilité et de réutilisabilité. Depuis une vingtaine d'années, la programmation par objets [Coi06] s'est imposée comme paradigme standard pour le développement des logiciels (Small-

talk, Java, C++). En particulier, les objets ont apporté un ensemble de principes de conception parmi lesquels l'abstraction (un objet possède un type de données et un comportement bien spécifié), l'encapsulation (interface et implémentation sont séparés), le polymorphisme (capacité de prendre des formes différentes à l'exécution), l'héritage (possibilité d'enrichir un objet avec des extensions) et l'identité (capacité de distinguer un objet parmi les autres). Cependant l'approche objet présente certaines limitations parmi lesquelles la difficulté de donner de la structure aux logiciels pour en maîtriser la complexité, la maintenance et la distribution. Par ailleurs, d'un point de vue technique, les objets sont dépendants des langages et des environnements d'exécution.

Composant La prise en compte de ces limitations est à l'origine de l'émergence des composants [Szy98] : « Un composant est une unité de composition avec des interfaces spécifiées à l'aide de contrats et de dépendances au contexte seulement. Un composant logiciel peut être déployé indépendamment et est sujet à être composé par des tiers ». Techniquement, les composants (par exemple EJB, COM et FRACTAL) permettent des regroupements cohérents et réutilisables d'objets. Ils se déploient et s'exécutent dans des contextes qui leur sont spécifiques. Permettant d'isoler de manière plus nette la phase de production et celle de consommation des logiciels, leur apparition marque une étape dans l'industrialisation des systèmes logiciels. Ainsi, si la programmation à petite échelle est bien couverte par les objets, la programmation à grande échelle trouve sa brique fondamentale dans le paradigme de composant. Cette étape fait notamment apparaître la notion d'architecture logicielle visant à définir la manière d'assembler les composants. Conceptuellement, le composant possède une implémentation en boîte noire, ainsi qu'une interface en boîte blanche. Par ailleurs, l'approche composant permet de mettre en oeuvre certains paramètres d'exécution (sécurité, persistance, mécanismes transactionnels, etc). Le cycle de vie classique du composant fait intervenir la composition statique, le déploiement, l'instanciation et l'exécution.

Service Tout comme les objets, les composants présentent un certain nombre de limitations, parmi lesquelles la dépendance au modèle de composant et au contexte d'exécution. Ces dépendances réduisent l'interopérabilité entre les différents types de composant et induisent des dépendances fortes entre les composants d'un même assemblage. De fait, la composition s'effectue majoritairement de façon figée entre contextes homogènes, souvent au sein d'une même organisation. C'est à ces limitations que la programmation orientée service(SOC [1]) [HS05, Pap03] essaie de répondre. Une définition de « service » peut être : « unité logicielle autonome et indépendante de toute plateforme, qui peut être décrite, publiée, découverte, orchestrée et programmée en utilisant des protocoles standards dans le but de créer des réseaux d'applications collaboratives distribuées à très large échelle ». En particulier, les caractéristiques fondamentales des services sont le couplage faible, l'interopérabilité et l'autonomie, ainsi que les mécanismes de publication et de découverte (faisant intervenir de nouveaux rôles de consommateur et de fournisseur de service). Les interfaces et les données exhibées par le service sont exprimées en termes métier, par le biais de contrats et de schémas de données. Les aspects technologiques ne sont plus essentiels car les services sont autonomes, c'est-à dire qu'ils sont indépendants du contexte d'utilisation ainsi que de

1. Service Oriented Computing

l'état des autres services, et interopèrent par échange de messages via des protocoles standardisés. De façon similaire aux approches par objet ou par composant, l'approche service cherche à fournir un niveau d'abstraction encore supérieur, en encapsulant des fonctionnalités et en permettant la réutilisation de services déjà existants. La propriété de couplage faible que cette approche induit est à l'origine des architectures agiles que sont les « Architectures Orientées Services ».

Architectures logicielles Les architectures basées sur composants requièrent un couplage fort entre les différents composants. Par exemple, il est quasiment impossible de faire communiquer des composants EJB et des composants COM. Leurs ressources et leurs traitements sont fortement liés, ce qui donne au système logiciel une grande stabilité et homogénéité. En effet, les composants s'inspirent du modèle objet et se basent sur des librairies de classes interdépendantes (nécessité d'un environnement d'exécution homogène, d'une compatibilité sémantique, d'un format de fichier bien défini, etc.). Un des avantages non négligeable de ce type d'architectures à base de composant est qu'il est capable de mettre en oeuvre certains mécanismes (sécurité, transaction, etc.) difficiles à mettre en place dans des environnements plus faiblement couplés et donc plus interopérables. L'accent est mis sur les performances de l'architecture.

Les Architectures Orientés Services forment un style d'architecture fondée sur la définition de services qui interagissent. Elles s'inspirent des intergiciels orientés messages et des concepts d'objets distribués, en cela que ces deux approches sont également basées sur des principes d'abstraction, d'encapsulation et d'échange de messages. Les services servent à promouvoir l'intégration logicielle via une compatibilité structurelle basée sur un partage de contrat (traitement) et de schéma (données). L'accent est ainsi mis sur la propriété de couplage faible entre services, qui de fait symbolisent à eux seuls de véritables applications. Les services sont autonomes et peuvent évoluer indépendamment les uns des autres, l'interopérabilité étant assurée par les schémas et les métadonnées autodescriptives. En outre, la propriété de couplage faible permet aux compositions de services de s'organiser sous la forme d'orchestrations. Ce procédé autorise une plus grande flexibilité dans le choix des services à composer ainsi que dans le moment de leur composition. L'accent est mis sur la flexibilité et l'interopérabilité du système.

2.1.2 Services Web

Technologie Dans le cadre de la programmation orientée service, un service peut être défini comme une entité fonctionnelle auto-contenue, auto-décrite, indépendante des plateformes, et pouvant être décrite, publiée, découverte, invoquée, composée à l'aide de protocoles standards. La technologie des services Web constitue une approche pour la concrétisation du paradigme de service sur le Web. Par le biais de cette technologie, il devient possible de délivrer et de consommer des services logiciels sur le Web, détournant l'usage premier de celui-ci qui ne consistait initialement qu'à publier et à lire des documents et des informations. Le W3C définit le service Web de la manière suivante :

> « *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems*

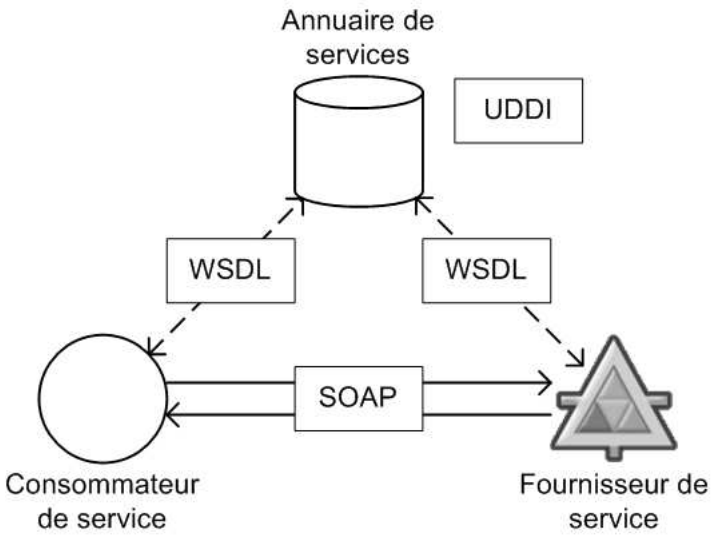

FIGURE 2.1: Modèle des services Web

interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. »

Cette définition est illustrée dans la figure 2.1.

Techniquement, les services Web se présentent comme des composants sans état dont l'enjeu majeur est de promouvoir un couplage faible. Pour cela les normes de communication associées aux services Web sont standardisées et entièrement spécifiées en XML, langage permettant une grande interopérabilité entre les systèmes. En particulier, les documents, messages et données sont modélisés par des schémas XML, ce qui permet leur compréhension par la grande majorité des plateformes.

Web Service Architecture La large implication des industriels dans l'élaboration des normes témoignent de l'ambition collective de faire communiquer plus facilement les systèmes informatiques par le biais du Web. La WSA [2] [WSA] a pour objectif de guider la progression des spécifications liées aux services Web, en les organisant dans divers domaines techniques. Ces domaines, illustrée dans la figure 2.2, sont répartis dans plusieurs couches fondamentales. L'existence de chaque couche est rendue possible par la présence des couches inférieures, l'objectif final étant d'abstraire le plus possible les mécanismes de communication et de rendre possible la création de véritables architectures d'applications distribuées basées sur les services Web.

La couche de transport est fondamentale et permet les échanges basiques de données (HTTP, SMTP, etc.). Elle a suscité de nombreux efforts de recherche lors de l'émergence des réseaux, mais elle n'est plus l'objet d'études spécifiques en ce qui concerne les services Web. La seconde couche décrit la sémantique des spécifications essentielles (SOAP, WSDL et UDDI), ainsi que des spécifications plus avancées (sécurité, garan-

2. Web Service Architecture

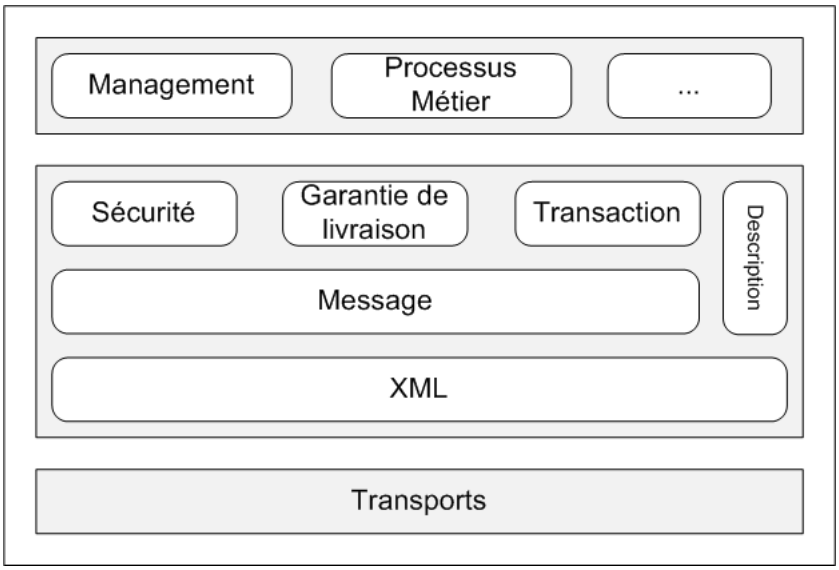

FIGURE 2.2: Web Service Architecture

tie de livraison, transaction et description). La dernière couche, située au sommet de l'organisation WSA, s'adresse à la gestion des services et à leur composition.

Spécifications élémentaires SOAP, WSDL et UDDI constituent les trois spécifications élémentaires liées aux services Web :

- SOAP [3] est le protocole d'échange de messages permettant d'interagir avec les services Web. Ces messages sont délivrés sous la forme de documents XML qui sont structurés par : une *entête* (pouvant être vide) contenant les informations non fonctionnelles liées au message (par exemple la sécurité), ainsi qu'un *corps* contenant les informations fonctionnelles (données et opérations du domaine d'application). SOAP est générique et extensible puisque les autres spécifications sont définies par dessus en définissant par exemple de nouvelles entêtes (sécurité, garantie de livraison). Ce protocole n'est pas lié à un type de protocole de transport de données particulier, mais il est fréquent qu'il soit associé à HTTP ou SMTP.
- WSDL [4] est le langage basé sur XML permettant de décrire les services Web. Il permet de générer des documents structurés en deux parties : une partie abstraite (le Quoi) décrivant l'interface fonctionnelle du service en termes d'opérations et de messages, ainsi qu'une partie concrète (le Comment) qui contient les détails des protocoles à utiliser et de l'adresse physique des opérations. En particulier, un *port type* désigne une collection d'opérations, un *binding* consiste en une association entre un *port type* ainsi qu'un protocole de transport et de format de données, un *port* définit l'adresse physique d'un *binding*, et un *service* constitue une collection de *ports*.
- UDDI [5] est une spécification et un service de registre permettant de publier et de

3. Simple Object Access Protocol
4. Web Service Description Language
5. Universal Description, Discovery and Integration

découvrir des services Web. Un registre UDDI est un registre basé sur XML qui contient des informations à propos d'entités d'affaire fournissant des services Web ainsi que des métadonnées concernant ces services (informations techniques ou légales). En outre, UDDI spécifie plusieurs API [6] pour interagir avec le registre, demander ou publier un service.

Spécifications avancées (WS-*) Les spécifications élémentaires permettent aux services Web de communiquer, de se décrire et d'être publiés. Elles forment une base pour que d'autres spécifications plus avancées puissent les étendre. Il s'agit de spécifications WS-* fournissant notamment des mécanismes de QdS (par exemple WS-Security ou WS-ReliableMessaging), de description avancée, ou bien de composition. Les spécifications WS-* sont composables, faisant ainsi participer plusieurs traitements. Ci-dessous nous analysons les spécifications qui nous intéressent pour les travaux de la thèse :

- WS-Security : Cette spécification traite des multiples mécanismes permettant de sécuriser l'échange de messages SOAP entre clients et services Web. Elle fournit une extension à SOAP implémentant les principes d'intégrité, de confidentialité et d'authentification. Pour cela, WS-Security spécifie des entêtes SOAP de sécurité contenant des constructions de type signature, clé, jetons, etc. Il existe encore d'autres spécifications de sécurité plus spécifiques permettant d'adresser la confiance (WS-Trust), les conversations sécurisées, les fédérations, etc.
- WS-ReliableMessaging : Entre l'envoi et la réception de messages SOAP, il peut se passer des erreurs de livraison, notamment lorsque des canaux de transmission ne sont pas fiables comme c'est parfois le cas avec Internet. Pour prévenir d'éventuelles pertes, la duplication ou bien garantir l'ordre de traitement des messages, la spécification WS-ReliableMessaging apporte des mécanismes de garantie de livraison. Pour cela, cette spécification utilise des éléments additionnels dans les entêtes des messages SOAP afin de mettre en oeuvre des mécanismes usuels tels que les messages d'aquittement.
- WS-Policy fait parti du domaine « description » des spécifications et se scinde en autant de sous spécifications (WS-SecurityPolicy, etc.) qu'il existe de spécifications de QdS (WS-Security, etc.). A l'instar du WSDL, les spécifications WS-Policy fournissent un moyen pour publier des informations concernant l'utilisation des services Web. Cependant, les WS-Policy adressent spécifiquement les exigences, capacités et préférences de QdS du service. Par exemple, un service peut déclarer qu'il supporte la garantie de livraison et qu'il requiert un mécanisme de cryptage 3-DES [7] des messages. Pour cela, WS-Policy fournit une cadre permettant de spécifier des politiques consistant d'une ou plusieurs assertions agrégées à l'aide d'opérateurs. La politique décrite figure 2.1 annonce qu'une politique est définie, puis que la politique requiert l'utilisation d'un jeton de sécurité, et enfin que ce jeton est de type *UsernameToken*.

```
1 <wsp:Policy wsu:Id="PresentAUserNameToken">
    <SecurityToken wsp:Usage="wsp:Required" xmlns="http://schemas.xmlsoap.org..">
      <TokenType>UsernameToken</TokenType>
    </SecurityToken>
  </wsp:Policy>
```

6. Application Programming Interface
7. Data Encryption Standard

Listing 2.1: WS-Policy spécifiant l'utilisation d'un jeton de sécurité

2.1.3 Composition de services

Principe La composition de services peut être définie comme étant « le procédé consistant à combiner des services existants pour former de nouveaux services ». Grâce à la propriété de couplage faible des services, la programmation orientée service permet d'effectuer aisément de tels compositions. Dans le monde des services Web, il existe deux approches pour concevoir des compositions : les orchestrations et les chorégraphies.

Une chorégraphie décrit le flux de messages échangés par un service Web lors de son interaction avec d'autres services. Il s'agit donc d'une manière décentralisée de gérer une composition puisque chaque service Web, acteur, est responsable d'une partie du *workflow*. En particulier, la norme WSCI [8] permet de spécifier le comportement du service Web par rapport au reste de la composition.

Une orchestration comprend un élément central responsable de la composition dans son ensemble. Le processus devient alors la somme de ses sous processus et l'entité responsable de la composition, appelée « orchestrateur », gère seule les échanges de messages. De nombreux travaux se sont intéressés aux langages d'exécution de processus métier, parmi lesquels WSFL d'IBM et XLANG de Microsoft. Ces efforts ont ensuite été fusionnés pour former une spécification commune nommée BPEL4WS [9] (appelé BPEL pour simplifier). Le langage BPEL est aujourd'hui un standard incontournable pour la spécification et la mise en oeuvre de compositions de services. C'est notamment vers ce langage que se sont tournés les efforts de l'industrie en développant de multiples technologies d'implémentation de BPEL (Microsoft BizTalk Server, IBM BPWS4J, ORACLE BPEL Server, etc).

Le langage BPEL4WS BPEL est un langage orienté processus permettant l'implémentation de services Web composites sous la forme de *workflows* exécutables par des moteurs d'orchestration BPEL. Un *workflow* BPEL est lui même constitué d'activités de contrôle, de gestion des données et d'interaction avec des partenaires. Ce langage permet de définir des partenaires (servant de canaux de communication avec les services Web et les clients) ainsi que des variables, et exhibe des activités simples et composites.

- Activités simples : **invoke** permet d'appeler le port d'un service Web partenaire, **receive** permet de se mettre en attente de la réception d'un message, **reply** correspond à la réponse adressée au client de la composition de services, **wait** effectue une temporisation, **assign** permet d'affecter une valeur à une variable, **throw** rejette une exception dans le *workflow*, **terminate** arrête l'instance du *workflow*, et **empty** correspond à une instruction sans effet.
- Activités composites : **flow** exécute un ensemble d'activités de manière concurrente et reliées entre elles par des liens, **sequence** exécute séquentiellement un ensemble d'activités, **switch** permet de sélectionner une branche d'activités parmi plusieurs, en fonction de conditions, **while** effectue des itérations jusqu'à satisfaction d'un critère, **pick** bloque le *workflow* jusqu'à ce qu'un événement spécifique

8. Web Services Choregraphy Interface
9. Business Process Execution Language for Web Services

se produise (réception d'un message, alarme temporelle, etc.), **scope** définit une zone restreinte du *workflow* permettant la définition de variables, de fautes et de gestionnaires d'exception, et **compensate** permet d'invoquer un processus de compensation d'exception.

Le listing 2.2 exhibe un exemple minimaliste de code BPEL déclarant deux partenaires (l'interface du composite et un service Web de météorologie), spécifiant deux variables et déclarant une activité composite de type *sequence* composée de trois sous activités simples (*receive*, *invoke* et *reply*) dont le rôle est de recevoir le message du client de la composition, de le transmettre au service météo, puis de renvoyer la réponse au client.

```
<process>
  <partnerLinks>
    <partnerLink myRole="proxy" name="proxyPLT" partnerLinkType="proxyPLT"/>
    <partnerLink name="Forecast" partnerLinkType="ForecastLinkType"/>
  </partnerLinks>
  <variables>
    <variable messageType="proxyRequest" name="proxyRequest"/>
    <variable messageType="proxyResponse" name="proxyResponse"/>
  </variables>
  <sequence>
    <receive createInstance="yes" operation="proxy" partnerLink="proxyPLT"
      portType="ProxyService" variable="proxyRequest"/>
    <invoke inputVariable="proxyRequest" operation="invokeWF"
      outputVariable="proxyResponse" partnerLink="Forecast"
      portType="invokeWFPT"/>
    <reply operation="proxy" partnerLink="proxyPLT" portType="ProxyService"
      variable="proxyResponse"/>
  </sequence>
</process>
```

Listing 2.2: Exemple de code BPEL

2.2 Séparation des préoccupations

2.2.1 Principes

Une préoccupation correspond à un sous-ensemble d'un système logiciel rattaché à la prise en compte d'un concept, d'un objectif ou bien d'un sujet particulier. Par exemple, dans le cas d'un serveur Web, il existe des préoccupations liées à la gestion des pages HTML ou à la gestion du protocole HTTP, ainsi que des préoccupations liées à la gestion de la sécurité ou du *logging*. Un problème inhérent à la définition de systèmes logiciels est que les multiples préoccupations des systèmes ne peuvent être représentées sans être disséminées dans les différentes composantes de ces systèmes. Autrement dit, les préoccupations ne sont pas encapsulées de façon adéquate tant au niveau des modèles de conception que des langages de programmation. Or, ce sont en particulier les interactions entre préoccupations qui sont responsables de la complexité des systèmes logiciels. Dans le cas du serveur Web Apache, il est montré [KLM+97] que le code implémentant la notion de session de connexion est disséminé dans une cinquantaine de classes différentes et ce malgré les mécanismes usuels de réutilisation tels que l'héritage. Le code obtenu est alors plus difficile à lire, à maintenir et à réutiliser, ce qui a également pour effet d'en réduire sa robustesse.

La séparation des préoccupations (*Separation of Concerns*) [Dij82, HL95] est un principe fondamental du génie logiciel. Son objectif est de permettre la maîtrise de systèmes logiciels complexes en adressant, de façon isolée, les diverses préoccupations qui les composent. Il s'agit pour cela que la structure des logiciels reflète le plus fidèlement

possible les relations intrinsèques entre leurs préoccupations afin de minimiser leurs interactions et, ainsi, de faciliter l'expression et la compréhension de ces interactions.

> « *Separation of concerns is a key guiding principle of software engineering. It refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal or purpose. Concerns are the primary criteria for decomposing software into smaller, more manageable and comprehensible parts that have meaning to software engineers.* » [EFB01]

Bien que, pour être mis en oeuvre l'existence, ce principe nécessite des technologies particulières, la séparation des préoccupations n'est pas une technologie en soi. Il s'agit d'un principe de conception qui a fortement influencé l'émergence de technologies permettant d'isoler les préoccupations à l'aide de stratégies plus ou moins efficaces.

Un premier type de décomposition permet de découper les logiciels de façon hiérarchique, selon une et une seule dimension. C'est ce type de décomposition que permettent d'effectuer la programmation modulaire, par objets, par composants et orientée service à présent. Cependant, ce type de décomposition conduit au problème de la « tyrannie de la décomposition dominante » [OT99] : les concepteurs, ayant choisi de décomposer leur système selon une préoccupation principale (typiquement une préoccupation métier), regroupent dans cette décomposition les autres préoccupations qui n'ont pas pu être modularisée. Ces préoccupations, qualifiées de « transverses » se retrouvent dispersées dans le reste du code et mélangées avec les autres, ce qui rend le logiciel difficile à comprendre, à maintenir et à faire évoluer. Certaines de ces préoccupations transverses peuvent être des préoccupations métier annexes, mais il s'agit pour la plupart de préoccupations dites « non-fonctionnelles » qui traitent par exemple de la sécurité ou encore des performances. Souvent, ces préoccupations ne sont d'ailleurs pas spécifiquement liées à l'application développée. Afin de modulariser ces préoccupations, le génie logiciel a développé de nouvelles techniques de programmation mettant en oeuvre le principe de séparation des préoccupations.

> « *When writing a modular program to solve a problem, one first divides the problem into sub-problems, then solves the sub-problems and combines the solutions. The ways in which one can divide up the original problem depend directly on the ways in which one can glue solutions together. Therefore, to increase ones ability to modularise a problem conceptually, one must provide new kinds of glue in the progamming language.* » [Hug89]

2.2.2 Réflexion et méta-programmation

> « *The most important contribution of the reflection community is to clearly illustrate the potential of separation of concerns.* » [Tho02]

La réflexion réfère à la capacité d'un système à raisonner et à agir sur lui-même [Smi84, Mae87]. En particulier, un système réflexif est un système qui fournit une représentation de son propre comportement qu'il est possible d'inspecter et de modifier. Pour obtenir une telle caractéristique d'un système, il faut que celui-ci dispose de deux capacités :

– La capacité d'*introspection*, qui correspond à la capacité de pouvoir s'observer soi-même. C'est grâce à cette capacité que le logiciel peut raisonner sur lui-même. En particulier, cette capacité se concrétise par la *réification* des structures et des

mécanismes du logiciel. Dans le cas d'un langage de programmation, il s'agit des classes, des méthodes, etc.

– La capacité d'*intercession*, qui correspond à la capacité de pouvoir se modifier soi-même. Dans le cas d'un logiciel, cette capacité correspond à la possibilité de modifier les entités réifiées et au fait que les modifications effectuées sont prises en compte de façon immédiate (principe de *connexion causale* selon lequel la réification détermine le comportement du logiciel).

La réflexion met en relation les deux niveaux fondamentaux que sont le niveau de base et le niveau méta. Le niveau de base capture le sujet principal du logiciel d'une application (par exemple la réservation d'un hôtel ou la gestion de comptes bancaires). Le niveau méta désigne alors le code réflexif, c'est à dire le code agissant sur les réifications du code de base. C'est notamment à ce niveau plus élevé qu'il est alors possible d'introduire des mécanismes d'exécution modularisant des préoccupations transversales (transaction, sécurité, etc.).

Les langages à objet, de part leur structure et leur organisation, sont particulièrement adaptés à la mise en oeuvre du principe de reflexion. Dans ce contexte, les objets appartenant au niveau méta sont alors appelés méta-objets, et leur rôle consiste à contrôler le comportement des objets du niveau de base. La communication entre les objets du niveau de base et du niveau méta est alors gérée par un protocole à méta-objets (MOP) [10] qui constitue l'interface entre les différentes niveaux. L'un des MOP les plus représentatifs est le MOP de CLOS [11] [KdRB91].

En ouvrant l'implémentation des logiciels, la réflexion offre une technologie particulièrement adaptée pour mettre en oeuvre le principe de séparation des préoccupations. En effet, les différents niveaux (de base, méta et multiples méta de plus haut niveau) permettent d'isoler les diverses préoccupations du logiciel, tandis que le MOP prend en charge la mise en oeuvre de la recomposition des préoccupations.

2.2.3 Programmation par aspects

La programmation par aspects (AOP [12]) [KLM+97, Aks96, MC96] représente un paradigme adressant spécifiquement la problématique de modularisation des préoccupations entrelacées dans un système logiciel complexe, et cherchant ainsi à répondre au problème de la tyrannie de la décomposition dominante.

Plusieurs approches pour le developpement de logiciel par aspects et basées sur une même modélisation sont présentées par Masuhara et Kiczales dans [MK03]. L'AOP introduit une nouvelle unité de modularité nommée « aspect » qui encapsule une préoccupation entrelaçante. Un aspect se caractérise par les structures suivantes :

– Un *langage de coupe* (*pointcut language*) qui permet de désigner un ensemble de points d'exécution du programme de base, appelés *points de jonction* (*join points*). Selon la modélisation des points de jonction et des coupes, un langage d'aspect peut cibler de manière plus ou moins précise les points d'exécution du programme de base. Dans le cas de langage à objet, de tels points de jonction peuvent par exemple correspondre à l'appel d'une méthode ou bien à l'accès à une donnée, etc.

10. métaObject Protocol
11. Common Lisp Object System
12. Aspect Oriented Programming

– Un langage d'*action* (*advice*) qui permet de spécifier la logique spécifique à la préoccupation modularisée par l'aspect. Généralement, le langage d'action correspond au langage de programmation de base. C'est cette action qui est mise en oeuvre aux points de jonction du programme de base identifés par la coupe de l'aspect. L'action est alors effectuée avant, après ou autour des points d'exécution, celui-ci étant alors réifié dans le langage d'action afin de pouvoir contrôler son exécution depuis l'aspect.

Un aspect consiste alors en un couple (coupe, action) implémentant une préoccupation et en injectant celle-ci de façon transparente dans le programme de base. Pour l'intégration de l'aspect dans le programme de base, l'AOP fait appel à un mécanisme de mise en oeuvre spécifique appelé « tissage » (*weaving*). Cette étape de mise en oeuvre peut être réalisée soit statiquement (par transformation de code source [KHH+01] au moment de la compilation programme de base ou de son chargement, soit dynamiquement (généralement par le biais de la réflexion [PSDF01] à l'exécution du programme de base.

Le listing 2.3 exhibe un aspect rédigé avec le langage AspectJ [KHH+01] et modularisant une préoccupation lié au *logging* d'une application.

```
1  public aspect Log
   {
     // Coupe
     pointcut logPointCut(Object caller): call(* foo(..)) && this(caller);
     // Placement de l'action
6    before(Object caller): logPointCut(caller)
     {
       // Action
       System.out. println ("Methode␣foo()␣appelee␣depuis␣" + caller.toString ());
     }
11 }
```

Listing 2.3: Aspect effectuant du logging spécifié avec AspectJ

Dans cet exemple, le point de jonction spécifie dans le programme de base l'ensemble des appels à la méthode « foo ». L'objet qui exécute la méthode est identifié dans la variable « caller ». L'action de l'aspect spécifie que, avant l'exécution d'un point de jonction de la coupe, un message doit être affiché précisant quel objet appelle la méthode.

2.2.4 Langages dédiés

> « *Un langage dédié est un langage de programmation ou un langage de spécification exécutable qui offre, grâce à des notations et abstractions appropriées, un pouvoir d'expression concentré sur, et généralement limité à, un domaine d'application particulier.* » [vDKV00]

Les langages de programmation usuels (tels que C, Java, Lisp, etc.) sont qualifiés de langages *généralistes* car, potentiellement, ils permettent d'adresser n'importe quel domaine et ainsi de créer des logiciels de types très variés (interface utilisateur, recherche dans un graphe, gestion d'un portefeuille d'actions). A l'inverse, les langages *dédiés*, ou DSL [13], sont des langages qui ont été spécifiquement conçus pour n'adresser qu'un domaine particulier pour la conception d'un logiciel. Il existe par exemple des DSL pour

13. Domain-Specific Language

l'interrogation de bases de données (SQL [14]), pour le développement de pilote [MRC+00] ou les ordonnanceurs [BM02].

Il s'agit de langages fournissant des constructions appropriées à une classe particulière de problèmes [Con04]. En particulier, ils permettant de simplifier la tâche de programmation étant donné que l'expertise du domaine est capturée dans le langage lui-même plutôt que codée par le développeur du logiciel. Le fait de programmer en utilisant des idiômes du même niveau que le domaine adressé augmente significativement la maîtrise de l'application. Ceci permet aux experts du domaine d'exprimer de manière naturelle leurs spécifications et facilite pour les non experts la compréhension et la réutilisation du code.

Parce que les traitements liés au domaine sont encapsulés dans le langage même, la probabilité pour le développeur de faire des erreurs se retrouve réduite, ce qui permet d'obtenir des applications plus robustes. En outre, par le biais d'analyses statiques, l'utilisation de constructions spécifiques à un domaine autorise des vérifications spécifiques qu'il serait impossible ou trop coûteux de réaliser avec du code écrit avec un langage généraliste. Ces vérifications apportent ainsi des garanties sur le code spécifié avec le langage dédié. Enfin, les langages dédiés sont plus performants car ils tirent profit des algorithmes spécifiquement optimisés pour le domaine qu'ils ciblent.

Pour toutes ces raisons, l'utilisation de langages dédiés s'avère être une technique particulièrement appropriée pour la mise en oeuvre du principe de séparation des préoccupations. Cependant, il est toutefois relativement ardu de concevoir un DSL car il faut d'une part bénéficier de connaissances dans les langages ainsi que dans le domaine capturé par le langage. Par ailleurs, un problème récurrent lors de la conception de DSL est de trouver le juste équilibre dans l'ouverture du langage : un langage dédié trop ouvert perd son intérêt face à un langage généraliste, tandis qu'un langage trop restreint est certes plus efficace mais utilisable sur une classe de problèmes amoindrie. Enfin, un DSL requiert de former les utilisateurs qui ont à l'utiliser et de développer les outils spécifiques à sa mise en oeuvre.

2.3 Qualité de Service

Le concept de QdS recouvre un large domaine de propriétés inhérentes au domaine de la transmission de données. Plusieurs travaux pour définir la QdS dans un système distribué sont à l'origine d'une proposition de standardisation [ISO]. La référence normative est le « Quality of Service Framework » publiée dans [ISO98]. D'après cette référence, les concepts fondamentaux suivants structurent la QdS :

- Caractéristiques de QdS : un aspect quantifiable de QdS, qui est défini indépendamment des moyens par lesquels il est représenté ou contrôlé.
- Etablissement de la QdS : l'utilisation de mécanismes pour créer les conditions pour l'activation d'un système afin que les caractéristiques de QdS désirées soient atteintes.
- Mécanisme de QdS : un mécanisme spécifique, pouvant utiliser des paramètres de QdS ou des informations de contexte, potentiellement en coordination avec d'autres mécanismes de QdS, dans le but de permettre l'établissement, le *monitoring*, la maintenance, le contrôle et la demande de QdS.

14. Structured Query Language

Dans la suite de cette section, nous étudions successivement ces concepts dans le cadre des services Web.

2.3.1 Caractéristiques de Qualité de Service des Services Web

Dans le cadre des services Web, le W3C [15] a identifié un ensemble de caractéristiques de QdS pertinentes pour le domaine des services Web [W3C03] :

Performance représente la vitesse avec laquelle un service Web répond à une requête. Cette qualité peut être mesurée en termes de débit, temps de réponse, latence, temps d'exécution, temps de transaction, etc.

Confiance qualité d'un service à exécuter certaines fonctions sous certaines conditions et dans un intervalle de temps donné. Il s'agit d'une mesure de la capacité d'un service à maintenir sa qualité (généralement le nombre de coupures de service par jour, semaine, mois). La confiance est également associée à la garantie de l'ordre et de la bonne livraison des messages. Dans notre contexte, WS-ReliableMessaging est un protocole qui permet de délivrer des messages selon certaines caractéristiques de livraison.

Passage à l'échelle permet de quantifier le nombre de requêtes auxquelles le service peut faire face dans un intervalle de temps donné.

Capacité nombre de requêtes qu'il est possible de traiter simultanément.

Robustesse capacité à fonctionner alors que les données en entrée provoquent des conflits ou sont incomplètes.

Traitement des exceptions un grand nombre de situations ne peuvent être prédites à l'avance ce qui implique de supporter les exceptions de façon adaptée.

Précision taux d'erreurs générées par le service.

Intégrité propriété garantissant que l'intégrité des données et des transactions est bien respectée, afin de ne pas aboutir à une situation inconsistante.

Accessibilité capacité à répondre à des requêtes.

Disponibilité le service devrait être prêt pour une consommation immédiate lors d'une invocation. La disponibilité est souvent associée au temps de réparation (ou TTR [16]).

15. World Wide Consortium
16. Time to Repair

Interopérabilité capacité à pouvoir communiquer quels que soient la plateforme et les langages utilisés. En ce qui concerne les services Web, c'est la normalisation et l'extensibilité de SOAP et des métadonnées qu'il contient qui permet de garantir l'interopérabilité. Cette tâche incombe aux d'organismes de normalisation et de vérification tels que le WS-I.

Sécurité il existe plusieurs traitements permettant de sécuriser les communications entre services Web (Authentification, Autorisation, Confidentialité, Traçabilité, Cryptage de données, Non répudiation). De multiples normes sont à mettre en relation avec la sécurité, dont principalement WS-Security, WS-Trust, WS-SecureConv, WS-Federation, etc.

2.3.2 Contrats de service

Etablir et s'assurer de la QdS d'un composant tel que le service Web représente un enjeu crucial puisque ceci permet d'établir une relation de confiance entre le fournisseur d'un service et un client en attente d'une certaine fiabilité. Cependant, contrairement aux spécifications bien établies dans le domaine fonctionnel des services Web (telles que WSDL, SOAP ou UDDI), il n'existe pas de standards officiellement reconnus par la communauté en ce qui concerne la description et l'établissement de propriétés de QoS. Pour autant, plusieurs travaux (WS-Agreement [LDK04], IBM WSLA [17] [KL03], WSOL [18] [TPP02], SLAng [LSE03]) visent à apporter des réponses à cette lacune par le biais des contrats de service (SLA) [19] dont l'objectif est de permettre la réalisation d'un accord entre consommateur et fournisseur de service portant sur le niveau de QdS que doit fournir le service.

Plus spécifiquement, un SLA consiste en un document généré à l'issu d'un protocole de négociation (plus ou moins complexe) entre le consommateur et le fournisseur de service, et qui permet de définir les offres et exigences de ces deux rôles. Les spécifications de ce document doivent être vérifiées à l'exécution via un mécanisme de *monitoring*. Le cycle de vie du SLA prévoit ensuite différentes possibilités : soit une spécification du SLA est mise en défaut (violation d'un terme) et un mécanisme correctif (prévu ou non dans le SLA) doit être mis en oeuvre, soit le SLA arrive à son terme (dans le cas où un intervalle temporel a été spécifié), soit une partie (consommateur ou fournisseur de service) décide de mettre fin à la relation client-service.

Pour exprimer un tel document, il faut un langage formel afin de permettre d'exprimer précisément les exigences de QdS. Par exemple, en imaginant qu'un client spécifie que « 95% du temps, la durée d'exécution de telle opération soit inférieure à 20 millisecondes ». Ce message possède des ambiguïtés qui le rendent impropres à réaliser un contrat. En effet, il n'est pas précisé à quels moments cette exigence doit être évaluée, ni quelle partie est en charge d'effectuer l'évaluation (consommateur, fournisseur, acteur externe), ni s'il faut considérer 20 millisecondes comme une moyenne ou bien un temps maximal, etc.

Pour le moment, le langage des SLA ainsi que leur processus de mise en oeuvre sont spécifiques aux travaux faisant la promotion des SLA. Il est toutefois possible de

17. Web Service Level Agreement
18. Web Service Offerings Language
19. Service Level Agreement

faire émerger des caractéristiques communes. Typiquement, la structure d'un SLA est composée des parties suivantes [jJMS02] :

- But : donne les raisons à l'origine de la création du SLA.
- Parties : décrit les parties impliquées dans le SLA et leur rôle respectif.
- Période de validité : définit la période de temps sur laquelle le SLA est actif.
- Service-Level Objectives (SLO) : les niveaux de service sur lesquels l'utilisateur et le service se sont mis d'accord. Chaque aspect du niveau de service (comme la disponibilité, la performance, la sûreté, etc.) doit atteindre une valeur cible.
- Service-Level Indicators (SLI) : ces indicateurs spécifient les sondes grâce auxquelles les mesures de QdS peuvent être effectuées.
- Pénalités : décrit les mesures à prendre quand il se produit une violation.
- Services optionnels : fournissent des services complémentaires, comme par exemple dans le cas d'une exception.
- Exclusions : spécifie ce qui n'est pas couvert par le SLA.
- Administration : décrit le processus pour mesurer les objectifs et définir la responsabilité organisationnelle pour superviser chacun des processus.

2.3.3 Politiques

Bien que n'étant pas uniquement applicable au domaine de la QdS ou des services Web, le concept de politique apporte des outils privilégiés dans le cadre de la mise en oeuvre de mécanismes de QdS dans le contexte des services Web. Conceptuellement, une politique est un moyen offert aux administrateurs pour définir et modifier l'organisation et le comportement d'un système. Elle se compose généralement de règles, dérivant de la description d'une stratégie et spécifiant au système les actions à entreprendre pour répondre à une situation donnée.

> « Policies are rules governing the choices in behavior of a system » [Slo94]

Plus formellement, une politique peut être définie selon deux perspectives [Wes01] :

- une stratégie, des objectifs permettant de guider et déterminer les actions présentes et futures à exécuter au sein du système.
- un ensemble de règles permettant d'administrer, superviser et commander l'accès aux ressources du système.

Ces définitions mettent en évidence les différents niveaux d'abstraction introduits par la spécification de politiques, selon l'utilisateur concerné. Les abstractions au niveau *business* simplifient le travail des administrateurs en définissant leurs besoins en tant qu'objectifs globaux tandis que les règles entrent dans les détails de l'implémentation de la stratégie à utiliser pour atteindre ces objectifs.

> « Objectifs -> Règles -> Commandes de bas niveau »

Ceci rend les systèmes plus flexibles et capables de se reconfigurer, pendant leur exécution si besoin, afin de respecter les règles prédéfinies. Les règles composant les politiques sont généralement décrites par des variantes du paradigme ECA [20] :

> « (on Event(s)) if Condition(s) do Action(s) »

20. Evénement Condition Action

Pour de telles règles, la condition est évaluée lors de l'événement spécifié et l'action est exécutée si cette condition est vérifiée. La spécification d'une politique peut être effectuée à l'aide d'un langage dédié. Les langages utilisent différentes techniques de représentation (objet, logique, graphique, etc.). Une forme canonique [KFY+06] peut être dérivée, se composant des éléments suivants :

- sujet (*subject*) : désigne l'entité à laquelle s'applique la politique.
- cible (*target*) : identifie l'entité affectée par l'application de la politique.
- événement (*event*) : décrit l'événement qui déclenche l'évaluation de la condition.
- action (*action*) : définit les opérations exécutées sur la cible afin d'imposer la politique.
- condition (*condition*) : décrit l'état du système et les contraintes permettant d'activité l'action.

Chapitre 3

Travaux sur la gestion de la Qualité de Service

Ce chapitre donne une description ainsi qu'une évaluation des principaux travaux connexes à ceux de la thèse. Il s'agit de travaux majeurs représentant l'état de l'art dans le domaine de la gestion de la QdS dans les compositions de services. La section 3.1 présente les critères d'évaluation que nous avons utilisés pour réaliser nos analyses. La section 3.2 présente les travaux concernant des plateformes de composition adaptées pour la mise en oeuvre de traitements liés à la QdS, tandis que la section 3.3 cible plus particulièrement les travaux spécifiques à la gestion de la QdS des compositions de services. Un bilan est effectué à la section 3.4.

Sommaire

3.1 Introduction

L'objectif de ce chapitre est d'effectuer une présentation ainsi qu'une évaluation des propositions existantes pour la gestion de la QdS dans les compositions de services. Afin d'analyser ces travaux de manière pertinente, nous avons identifié des critères d'évaluation propres au domaine étudié :

Réutilisation de l'existant Dans le contexte des compositions de services ou plus généralement des SOA, plusieurs standards, langages et plateformes ont été adoptés ou sont en cours de maturation pour devenir des outils incontournables de « l'orienté service ». Il s'agit de standards tels que SOAP, WSDL, BPEL4WS et les moteurs d'orchestration qui se sont stabilisés, ou bien encore les normes WS-* (WS-Policy, WS-Security, etc.), présentés à la section 2.1.2. Les SLA deviennent des outils incontournables malgré qu'ils doivent encore faire l'objet de travaux de normalisation. La réutilisation de ces standards et outils est un critère particulièrement intéressant car il permet de capitaliser sur des approches et des outils déjà développés. Dans le cas où les travaux ne réutilisent pas ces standards, ils doivent redévelopper des outils redondants, dont l'intégration avec les SOA se retrouve réduite. Pour évaluer ce critère, nous considérons deux éléments en particulier : d'une part la réutilisation du langage **BPEL** (qui conditionne la capacité à réutiliser les travaux et les outils concernant les orchestrations) et d'autre part la réutilisation des **SLA** ainsi que des politiques **WS-Policy** qui contiennent la description de la QdS et de mécanismes liés à la QdS.

Séparation des préoccupations La gestion de la QdS peut être considérée comme une préoccupation, comme définie dans la section 2.2. La préoccupation de base (ou métier) est la gestion de la composition de services (capturée dans le langage BPEL et mise en oeuvre par le moteur d'orchestration). Le critère de séparation des préoccupations consiste à évaluer à quel degré les approches proposées isolent ces deux préoccupations. En particulier, nous nous intéressons à l'application de ce principe à deux étapes : l'étape de spécification et l'étape de mise en oeuvre. A l'étape de spécification, les logiques qui adressent les différentes préoccupations sont plus ou moins séparées, réalisant un couplage plus ou moins lâche entre elles. Or, plus les spécifications sont isolées, moins elles sont complexes à comprendre et plus elles sont flexibles et réutilisables. Nous cherchons donc tout d'abord à évaluer l'**isolation** des spécifications liées aux diverses préoccupations. Ces spécifications sont ensuite exécutées par des infrastructures implémentant des mécanismes spécifiques. A l'instar des spécifications, ces infrastructures et ces mécanismes sont plus ou moins couplés entre eux. Or, plus ils sont couplés entre eux, plus il devient difficile de les faire évoluer sans faire évoluer le reste du système. Dans le contexte particulier des compositions de services, nous nous intéressons donc à mesurer l'**intrusivité** des mécanismes de gestion de la QdS dans la plateforme mettant en oeuvre la composition (généralement, le moteur BPEL).

Couverture de gestion de la QdS Comme présentée dans la section 2.3, la gestion de la QdS est un vaste domaine et fait intervenir plusieurs dimensions. Les travaux existants adressent diverses parties de la gestion de la QdS. Notamment, certaines contributions se sont spécialisées et offrent des outils pour une prise en charge exhaustive de certaines parties de la gestion de QdS, tandis que d'autres, plus généralistes mais moins avancées en termes de réalisation, adressent un spectre plus large de préoccupations. Par l'intermédiaire du critère de couverture, nous analysons l'étendu du domaine de gestion de la QdS qu'adressent les différentes contributions. Pour cela, nous évaluons trois éléments dans la prise en compte de la QdS. Tout d'abord, l'étendu des **caractéristiques** de QdS prises en compte par la contribution est évaluée. En particulier, il est intéressant de connaître si la contribution prend en compte les caractéristiques

de type performance (telles que temps de réponse, le débit ou la disponibilité) mais également les caractéristiques plus spécifiques au contexte des services (telles que des informations de sécurité comportant des éléments WS-Security). Dans un deuxième temps, nous analysons l'étendu des **mécanismes** qu'il est possible de mettre en oeuvre à travers la contribution (négociation d'un accord, *monitoring* à l'exécution, réactions aux événements, garantie de livraison, recherche de services). Enfin, certaines approches font apparaître différents niveaux de **granularité** pour les éléments de la composition de services faisant l'objet des traitements de QdS. Ce critère est intéressant en ce sens qu'une approche qui ne permet pas d'adresser tous les niveaux de granularité est limitante. Par exemple, une approche peut permettre d'effectuer une recherche de services sur une zone du *workflow* plutôt que sur son intégralité ou bien sur un service seulement.

Dynamicité La mise en oeuvre des compositions de services constitue un processus composé de plusieurs étapes parmi lesquelles l'étape statique de préparation de la composition, l'étape de déploiement de la composition sur sa plateforme d'exécution et enfin l'étape d'exécution de la composition. Des traitements de QdS peuvent être associés à ces diverses étapes. Tandis que certaines approches ne considèrent que la prise en compte de l'étape statique, d'autres envisagent l'adaptation de la QdS à l'exécution et même certaines contributions laissent aux utilisateurs la possibilité de modifier le comportement de gestion de la QdS à l'exécution. Par conséquent, il existe une variabilité dans les moments de spécification et de mise en oeuvre de la gestion de la QdS. Le critère de **dynamicité** cherche à capturer la capacité des diverses approches à permettre aux utilisateurs d'adresser une plage plus ou moins étendue de moments pour la gestion de la QdS.

Expressivité Les approches étudiées permettent aux utilisateurs de spécifier les traitements de QdS qu'ils souhaitent voir être mis en oeuvre. Pour cela, elles offrent des interfaces aux modèles plus ou moins **riches** permettant d'écrire des spécifications. Un modèle trop pauvre (par exemple sous la forme de simples fichiers de configuration) peut être limitant car l'utilisateur peut ne pas pouvoir spécifier l'ensemble des exigences qu'il désire, tandis qu'un modèle trop riche (par exemple un langage généraliste) peut nécessiter une expertise trop complexe à maîtriser. Il nous semble donc important d'évaluer quel est le degré d'**expressivité** offert par les plateformes quand celles-ci offrent une interface pour les spécifications de l'utilisateur. Dans le cas d'un langage, nous essayons d'identifier sa nature (**déclaratif** ou **impératif**).

Chacune des propositions a été classée parmi deux catégories : **plateformes adaptatives** et **plateformes de traitement de la QdS**. Il existe des recoupements possibles entre ces deux types de plateformes, mais nous avons souhaité isoler ces propositions sur la base ce qui constitue leur contribution fondamentale.

Ainsi, la catégorie « plateformes adaptatives » (section 3.2) concerne les travaux qui visent principalement à ouvrir les compositions de service pour réaliser des adaptations, permettant notamment de rajouter des comportements propres à la gestion de la QdS. Réalisant une meilleure séparation des préoccupations, certaines de ces plateformes proposent des langages pour spécifier avec un haut niveau d'abstraction les comportements que l'utilisateur souhaite mettre en oeuvre.

La catégorie « plateformes de traitement spécifique de la QdS » (section 3.3) concerne les travaux traitant spécifiquement de la gestion de la QdS dans les compositions de services. Ces travaux proposent tous une modélisation des propriétés de QdS et des algorithmes d'agrégation ou de sélection des services dans le but de garantir la QdS globale, et parfois de l'optimiser. Certains de ces travaux ne concernent pas directement les plateformes de composition usuelles (moteurs BPEL) et proposent un degré d'abstraction plus élevé, par l'intermédiaire de leur modélisation, afin de prendre en compte la QdS sur une plus grande variété de *workflows*.

3.2 Plateformes adaptatives pour la composition de services

Dans cette section, nous étudions cinq travaux proposant des plateformes adaptatives pour la composition de services.

3.2.1 AO4BPEL

Introduction Ce travail [CM04, CSHM06] se propose d'introduire le concept de programmation par aspects dans les processus métier afin de pallier d'une part au manque de modularisation des préoccupations transverses, et d'autre part au manque de dynamicité des adaptations. Les auteurs de ces travaux ont ainsi réalisé un modèle et une implémentation de langage d'aspect (AO4BPEL [1]) basé sur le langage BPEL. Un moteur BPEL spécifique a également été conçu pour la gestion des aspects, ainsi que pour l'intégration de services *middleware* non fonctionnels usuels (sécurité, garantie de livraison, transaction) permettant de répondre à la problématique du manque de prise en charge des normes WS-* au niveau des compositions.

Langage Comme tout langage orienté aspect, le langage AO4BPEL possède un modèle de points de jonction, un langage de coupe et un langage d'action. Il a cependant la particularité d'être entièrement rédigé en XML. Selon la modélisation AO4BPEL, chaque activité est un point de jonction potentiel et il existe également des points de jonction liés aux événements d'envoi et de réception de messages SOAP. Etant donné que le langage BPEL est rédigé en XML, chaque balise XML peut être identifiée à l'aide d'une expression XPath. Les auteurs de AO4BPEL ont ainsi basé leur langage de coupe sur le langage XPath [2]. Une coupe consiste en une collection de points de jonction, c'est à dire d'expressions XPath pouvant être composées à l'aide des opérateurs d'union ou d'intersection. Quant au langage d'action, il s'agit du langage BPEL augmenté par des mots clés « <Proceed/> » et un mécanisme de réflexion (donnée « ThisJPInVariable » ou encore « soapmessageout »). Le contenu d'un aspect AO4BPEL consiste ainsi en une coupe et une activité BPEL devant être appelée à l'endroit de la coupe. De plus, il est possible d'assigner des ordres de priorité afin d'ordonner l'exécution d'aspects se déclenchant à un même point de jonction. Finalement, AO4BPEL intègre également un mécanisme d'introduction particulièrement approprié pour ajouter des partenaires ou encore des variables dans les processus BPEL. Le listing 3.1 exhibe un

1. Aspect Oriented for Business Process Execution Language
2. XML Path Language

aspect AO4BPEL dont la coupe spécifie comme point de jonction les activités de type *invoke* placées comme descendants directs de l'activité *process* et dont l'opération porte le nom *getFlight*. L'action correspond à l'appel d'un service de *logging* avant l'exécution des points de jonction.

```
<aspect name="logging">
  <pointcutandadvice>
    <pointcut name="invokeFlight">
      //process[name="travelPackage"]//invoke[operation="getFlight"]
    </pointcut>
    <advice type="before">
      <invoke name="logFlight" ../>
    </advice>
  </pointcutandadvice>
</aspect>
```

Listing 3.1: Exemple d'aspect AO4BPEL

Mise en oeuvre La mise en oeuvre de AO4BPEL est effectuée par un moteur BPEL s'appuyant sur le moteur BPWS4J d'IBM. Cette approche permet aisément de composer dynamiquement des aspects ainsi que des processus métier, et supporte les points de jonction correspondant à l'envoi et à la réception de messages SOAP. Le moteur AO4BPEL étend le moteur BPWS4J avec un composant de déploiement d'aspects et un autre composant d'exécution d'aspects. Le composant de déploiement possède une interface grâce à laquelle l'architecte peut manipuler les aspects et visualiser les interactions de ces derniers avec le processus, tandis que le composant d'exécution construit autour de l'interprète BPEL du moteur original vérifie l'exécution potentielle d'aspects. Si une correspondance existe entre un point d'exécution du processus BPEL et la coupe d'un aspect AO4BPEL, alors cet aspect est exécuté par l'interprète.

Services middleware La gestion des propriétés non fonctionnelles fait parti des services *middleware* usuels. Cette gestion peut avoir lieu à plusieurs niveaux de granularité (message et *workflow*). Pour pallier au manque de prise en compte de tels services *middleware* par les moteurs BPEL, le moteur de AO4BPEL intègre des services *middleware* accessibles sous la forme de services Web qu'il est possible d'appeler via des aspects. Afin de générer aisément les aspects ayant un lien avec les propriétés non fonctionnelles, un mécanisme de descripteur de déploiement a été ajouté. Les services *middleware* offerts concernent la sécurité, la garantie de livraison et les transactions. La figure 3.1 exhibe les interactions entre les différenss éléments faisant parti de la mise en oeuvre d'AO4BPEL.

Evaluation Ce travail démontre que le concept de programmation par aspects est non seulement applicable aux processus métiers, mais qu'il est également approprié pour les rendre facilement adaptables. Le langage AO4BPEL s'inspire judicieusement du langage AspectJ [KHH+01] (notamment pour la spécification du type d'*advice*) et du langage XPath (identification de la coupe). Il étend le modèle de points de jonction afin de pouvoir adresser les événements de type envoi et réception de messages SOAP. Par ailleurs, le déploiement d'aspects est dynamique ce qui permet de faire évoluer les processus métiers à l'exécution. Les services *middleware* mis à disposition par le moteur AO4BPEL sont modifiables et extensibles. Ils permettent aux architectes de mettre en oeuvre les standards WS-* dans des processus métiers, via les descripteurs de déploiement.

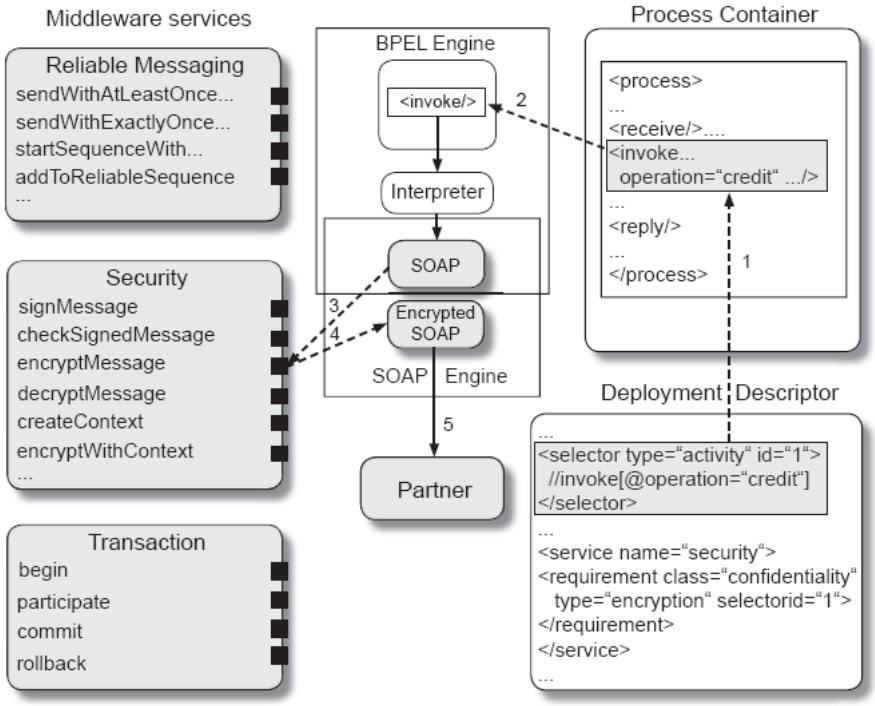

FIGURE 3.1: Modèle de mise en oeuvre des aspects AO4BPEL

Néanmoins ce travail requiert un moteur BPEL spécifique, ce qui en fait une solution inexploitable par les autres plateformes. Par ailleurs l'utilisation d'aspects modularisant du code BPEL est limitée du seul fait que le langage BPEL n'est pas un langage généraliste. Par conséquent, il n'est pas possible de rajouter de nouvelles préoccupations excepté en développant de nouveaux services Web qui intègreront ces préoccupations. Cette façon de procéder n'est pas aisée, n'offre pas de possibilités de garantir certaines propriétés avant l'exécution, et impose un couplage fort entre le moteur d'exécution et les services Web contenant la logique des préoccupations non fonctionnelles. En outre, si les outils BPEL proposent des modélisations sous forme de graphe pour permettre d'écrire aisément du code BPEL, AO4BPEL requiert que l'utilisateur écrive lui même son code. Enfin, AO4BPEL n'intègre pas la gestion des caractéristiques de QdS de type performance (temps de réponse, débit, etc.).

3.2.2 DYNAMO

Introduction Ce travail [BGP05, BGP07, BG07] vise à apporter des solutions face aux lacunes du langage BPEL4WS concernant la spécification et la mise en oeuvre d'exigences non fonctionnelles dans les compositions de services. Pour cela, les auteurs introduisent deux nouveaux langages, « WSCoL » [3] et « WSRel » [4] qui ont respectivement pour but de permettre la définition, dans un processus BPEL, d'assertions liées au *monitoring*, ainsi que de stratégies de réaction. Ces langages ne dépendent

3. Web Service Constraint Language
4. Web Service Recovery Language

pas d'un domaine en particulier et s'expriment à travers WS-Policy qui permet d'attacher des politiques à des services Web ainsi qu'à des documents BPEL grâce à la norme WS-PolicyAttachment. Pour mettre en oeuvre ces langages, les auteurs présentent la plateforme « DYNAMO » [5]. DYNAMO utilise la programmation par aspects pour superviser l'exécution des processus BPEL, effectuer le *monitoring* et mettre en oeuvre les stratégies de récupération.

Langages Les langages WSCoL et WSReL exploitent le standard WS-Policy qui permet de décrire des assertions correspondant au fonctionnement d'un service. Une assertion désigne une préférence, exigence, capacité ou une autre propriété fonctionnelle (contraintes sur des informations) ou non fonctionnelle (sécurité, garantie de livraison). Les assertions sont rédigées à l'aide de langages spécifiques (tels que WS-SecurityPolicy) et un document WS-Policy permet d'agrèger ces assertions à l'aide d'opérateurs. Des documents WS-Policy peuvent être définis à la fois du côté service et du côté client. En réalisant l'intersection des politiques client et service, les politiques effectives (requises et offertes) apparaissent. Dans ce contexte, les langages d'assertion WSCoL et WSRel sont conçus afin de définir des propriétés de *monitoring* et de réaction. Ils offrent une interface nouvelle pour spécifier des comportements indépendants de l'exécution du processus BPEL. Ces informations sont ensuite liées au document BPEL grâce au standard WS-PolicyAttachment qui précise une expression XPATH faisant le lien entre processus et politiques.

Le langage WSCoL Ce langage permet de définir des contraintes sur l'exécution des services et s'inspire de langages d'assertion existants (tels que Anna et JML). Ces contraintes sont vérifiées soient avant l'appel au service (pre-condition), soit au retour du message (post-condition). Ses principales fonctionnalités consistent d'une part à récupérer des données (variables internes liées au processus BPEL, variables externes pour récupérer des informations venant de sondes, variables historiques pour partager des données sur plusieurs instances de processus), et d'autre part à analyser les données recueillies par le biais de formules mathématiques et de structures (« forall », « exists »). Le listing 3.2 donne une utilisation possible du langage WSCoL permettant de vérifier certaines hypothèses sur un message (« parkingLotResponse »).

```
<wsp:Policy xml:base="http://www.microsoft.com/policies" wsu:Id="MapPointPolicy"
  xmlns:wsp="..." xmlns:wscol="...">
  <wsp:All xmlns:wscol="...">
    <wscol:MonitoredItem type="postcondition" path="//definitions/message[..]">
      <wscol:Expression>
        let parkings=message[name='parkingLotResponse']/part[name='parking'];
        let radius=message[name='parkingLotRequest']/part[name='radius'];
        (forall parking in parkings;(parking/UTMEasting-easting)^2 +
          (parking/UTMNorthing-northing)^2 <= radius^2)
      </wscol:Expression>
    </wscol:MonitoredItem>
  </wsp:All>
</wsp:Policy>
```

Listing 3.2: Exemple de code WSCoL

Le langage WSReL A l'exécution, lorsqu'une expression liée au *monitoring* est violée, il faut pouvoir réagir. Pour cela, le langage WSReL a été développé et permet

5. DYNAmic MOnitoring

de spécifier des stratégies de recouvrement. En particulier, ce langage permet d'écrire des séquences d'actions atomiques à mettre en oeuvre lors d'une violation. Certaines actions se terminent avec succès, tandis que d'autres peuvent échouer à nouveau lors du *monitoring*. Les actions sont donc mises en oeuvre successivement jusqu'à ce qu'une action se déroule avec succès. Les actions qu'il est possible d'effectuer sont du type : ignorer, notifier par mail, stopper le processus en exécution, appeler un service extérieur, remplacer un service par un autre, rejetter une exception dans le processus BPEL, réessayer, etc. Le listing 3.2 donne une utilisation possible du langage WSReL : lorsque l'expression liée au module de *monitoring* « Urgent request » est violée, le code met successivement en oeuvre trois stratégies (tant qu'une stratégie termine sur un échec). Il s'agit de renvoyer à nouveau le message, utiliser un autre service, et finalement de notifier par mail un administrateur.

```
<wssup:SupervisionRule>
  <wssup:postcondition>
    <wscol:Expression id="postcond_1">...</wscol:Expression>
  </wssup:postcondition>
  <wssup:strategy>
    <wssup:strategycondition id="strategycond_1">
      <wscol:Expression>‘‘Urgent request’’</wscol:Expression>
    </wssup:strategycondition>
    <wssup:step number="1"> <wssup:retry times="1"/> </wssup:step>
    <wssup:step number="2"> <wssup:rebind url="http://..."/> </wssup:step>
    <wssup:step number="3">
      <wssup:notify>
      <wssup:message>...</wssup:message>
      <wssup:address>...</wssup:address>
      </wssup:notify> <wssup:halt/>
    </wssup:step>
  </wssup:strategy>
</wssup:SupervisionRule>
```

Listing 3.3: Exemple de code WSReL

Mise en oeuvre Afin de superviser l'exécution des processus BPEL, une plateforme (« DYNAMO ») est proposée par les auteurs. Cette plateforme est composée d'un moteur BPEL étendu et d'un système de *monitoring* et de recouvrement. En ce qui concerne le moteur BPEL, DYNAMO réutilise le moteur ActiveBPEL sur lequel sont greffés des aspects responsables de la mise en oeuvre du *monitoring* et du recouvrement. Les coupes de ces aspects sont positionnées sur les composants d'ActiveBPEL en charge de l'exécution de certaines activités (*invoke*, *receive* et *pick*). Ainsi, après tissage, le composant en charge du *monitoring* a un accès direct vers les données du processus et à son état, ce qui facilite la tâche de récupération des données lors de l'interprétation du langage WSCoL. Quant aux règles de recouvrement, elles sont implémentées sur le moteur de règle JBoss Rule Engine. Ce moteur a la charge d'évaluer les expressions de *monitoring* et d'activer les stratégies de recouvrement correspondantes. Ces règles sont de type ECA et les événements (violation) sont signalés par le composant de *monitoring*. Les réactions ont lieu soit au départ d'un message SOAP vers un service Web, soit à la réception. Elle sont effectuées selon un ordre de priorité défini par l'utilisateur dans les politiques.

Evaluation En choisissant de proposer deux nouveaux langages pour le *monitoring* et la réaction aux violations, les auteurs de ces travaux cherchent à introduire une meilleure séparation des préoccupations pour la gestion de la supervision des processus. Ces langages ciblent directement le BPEL en exploitant les standards WS-Policy et

WS-PolicyAttachment afin d'attacher les spécifications de la manière la moins intrusive possible. Les langages WSCoL et WSReL offrent un ensemble de primitives permettant d'agir sur les services et les données échangées. Il est également dans le périmètre de ce travail de traiter des normes WS-*. La plateforme d'exécution DYNAMO vient étendre le moteur ActiveBPEL en utilisant la programmation par aspects afin de récupérer et agir sur le contexte d'exécution de façon découplée. Ces travaux mettent en valeur le net intérêt que constitue la prise en compte de propriétés fonctionnelles et non fonctionnelles, que ce soit lors du déploiement ou bien à l'exécution.

Cependant, en choisissant de travailler uniquement avec les documents WS-Policy, les langages WSCoL et WSReL ne prennent pas en compte les caractéristiques de type performance. En particulier, cette solution s'est spécialisée dans le *monitoring* des données du processus et des données récupérables par des services tiers. Le traitement statique (au déploiement) ne peut pas mettre en oeuvre un grand nombre de vérifications concernant la QdS. Ensuite, la cible du *monitoring* et du recouvrement ne peut dépasser la granularité d'une activité, et donc reste uniquement à un niveau message. En effet, les événements surveillés sont l'arrivée et le départ de messages SOAP (pre et post-conditions), ce qui signifie qu'une simple technique d'interception est suffisante. Enfin, si la plateforme DYNAMO utilise la programmation par aspects pour se lier avec le moteur ActiveBPEL, le partage des variables de contexte et les spécificités du moteur BPEL pour l'exécution du processus impliquent un couplage très fort entre DYNAMO et ActiveBPEL, ce qui diminue l'intérêt de l'utilisation des aspects.

3.2.3 MASC

Introduction MASC[6] [TEM07, ETM07, EMT06] consiste en un *middleware* dont la préoccupation est le *monitoring* des compositions de services Web ainsi que leur adaptation à l'exécution. Les exigences liés au *monitoring* ainsi que les actions d'adaptation sont spécifiées dans un langage appelé WS-Policy4MASC. Ce langage, basé sur XML, définit de nouveaux types d'assertions pour le langage WS-Policy, permettant ainsi à l'utilisateur de préciser des exigences de QdS ainsi que des informations liées à l'exécution (événements, conditions, réactions). Le langage WS-Policy4MASC se présente comme indépendant de tout domaine et est conçu comme un langage de politique. A l'instar de Baresi et al (section 3.2.2), ce travail exploite le standard WS-PolicyAttachment afin de pouvoir relier un document de politique à un *workflow* BPEL ou bien à une interface WSDL.

Langage Le langage WS-Policy4MASC introduit de nouveaux types d'assertion dans le langage WS-Policy :

- « Goal policy assertion » : ce type d'assertion permet d'exprimer des exigences (temps de réponse). Ces exigences guident les activités de *monitoring*.
- « Action policy assertion » : il s'agit d'actions à exécuter si certaines conditions sont satisfaites (par exemple, si des exigences sont violées). Ces actions sont divisées en quatre sections relevant de l'adaptation structurelle (opération sur une portion du *workflow*), de l'adaptation de l'instance d'exécution (terminaison, suppression ou reprise du processus), de l'adaptation de l'activité en cours

6. Manageable and Adaptive Service Compositions

d'exécution (annulation, compensation, etc.), de l'adaptation des assertions (activation, désactivation de politiques). Les actions guident le processus d'adaptation et permettent d'interagir avec la plateforme d'exécution.
- « Utility policy assertion » : ces assertions aident à la spécification de diverses valeurs liées au *business* de certaines situations. Par exemple, elles peuvent être utilisées pour la facturation ou pour la sélection entre différentes alternatives d'assertions.
- « Meta-policy assertion » : ce type d'assertion permet de spécifier les alternatives entre les assertions d'action et comment résoudre les conflits de stratégie (par exemple maximisation des profits).

WS-Policy4MASC supporte la résolution de divers types de conflits et permet de renseigner des informations concernant les conditions liées à l'exécution ou l'évaluation des assertions.

Mise en oeuvre La mise en oeuvre du langage WS-Policy4MASC repose sur la plateforme MASC qui est fortement couplée au moteur d'exécution (comme décrit à la figure 3.2). Pour celle-ci, les auteurs ont choisi d'utiliser le composant Windows « *workflow* Foundation » (WF) pour l'exécution des compositions de services. Ce composant fait parti du *framework* .NET 3.0 et permet d'exporter des *workflows* sous la forme d'applications .NET. Les processus WF sont définis à l'aide du formalisme XAML [7] et sont exécutés à l'aide d'un moteur spécifique léger qui peut prendre en compte un ensemble varié de préoccupations liées au *middleware* (persistance, transaction, etc.). Les auteurs de MASC ont ainsi développé un nouveau service WF, MASCAdaptationService, pour la mise en oeuvre de WS-Policy4MASC dans les processus WF. Ce service transcrit les politiques WS-Policy4MASC sous la forme de classes C# à l'aide d'un générateur et a accès aux variables ainsi qu'au code d'exécution entre les appels du *workflow* vers les services Web. Un autre service, MASCMonitoringService, a pour objectif la gestion du *monitoring* des événements (échange de messages) entre le processus et les services Web. Ce service a accès à la file de messages échangés, ce qui permet de récupérer les événements avant l'envoi de message et à la réception de la réponse. Il permet notamment de prendre la mesure de la fiabilité, du temps de réponse et de la disponibilité. Les événements de type dépassement de seuil pour la QdS sont alors récupérés à ce niveau de l'architecture. Ces services mettent alors en oeuvre divers composants responsables de l'adaptation (Adaptation Manager), du stockage des services Web fonctionnellement équivalents (Web services Selection service), des filtres de message (*Message Inspectors*, *Processing Modules*), et de la transformation de messages (*Message Adaptation Service*).

Evaluation Les travaux autour de MASC soulignent l'intérêt d'un langage pour la spécification de la supervision de l'exécution des compositions de service. En isolant la logique d'adaptation du code en charge du processus d'exécution, cette approche permet une meilleure séparation des préoccupations. En outre, ce langage s'oriente vers la définition de politiques où les actions sont déclenchées lorsque certains événements surviennent et sont captés par les activités de *monitoring* de la plateforme.

7. Extensible Applications Markup Language de Microsoft

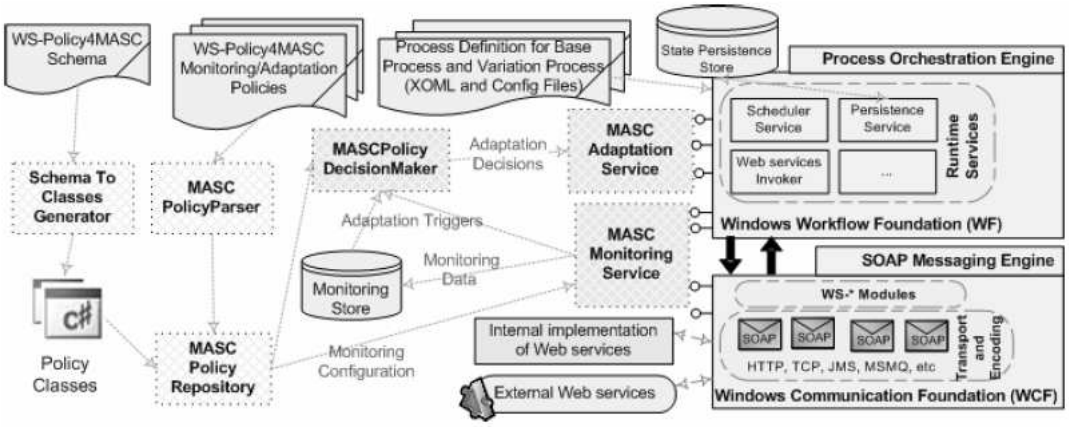

FIGURE 3.2: Plateforme MASC

Bien que très ambitieux, ces travaux restent encore relativement jeunes, d'où le manque de précisions dans les réalisations effectuées. Tout d'abord, le langage ne semble pas posséder de domaine de préoccupation clairement défini, ce qui rend l'écriture de politique relativement complexe. Son expressivité, vaste et parfois non homogène, est difficilement abordable pour un utillisateur non expert. Par ailleurs, les traitements liés au calcul de la QdS sont mis en oeuvre uniquement à l'étape dynamique sans pour autant que les algorithmes de calcul des propriétés de QdS du *workflow* ne soient explicités, et ce malgré les problèmes connus de complexité [ZBN+04]. Il n'existe donc pas de garanties que les exigences données par l'utilisateur soient satisfaites. Enfin la plateforme MASC est fortement liée à la plateforme d'exécution spécifique « *workflow* Foundation » de Microsoft, ce qui lui permet d'être très flexible quant à la variété de traitements qu'il est possible d'effectuer. Cependant ce choix complexifie le portage de ces travaux vers les plateformes usuelles d'exécution de compositions de services comme les moteurs BPEL.

3.2.4 Trap/BPEL

Introduction TRAP/BPEL [ES06, SM05] est un *framework* qui vise à importer de manière automatique et transparente des comportements autonomiques dans les processus BPEL. Les auteurs définissent ce type de comportement comme la capacité d'un service composite à répondre aux variations à l'exécution (en particulier, l'échec d'un service Web partenaire). Ce travail exploite un concept de programmation, appelé *Transparent shaping* [SMCS04] et développé par les auteurs, qui apporte une meilleure séparation des préoccupations ainsi que des mécanismes particuliers de mise en oeuvre. Avec ce procédé, la prise en compte de comportements autonomiques dans les *workflows* ne requiert ni d'extension du langage BPEL ou du moteur BPEL, ni de modification manuelle de l'orchestration originale, ce qui permet une meilleure réutilisation du code lié aux diverses préoccupations. Pour permettre la spécification du code lié aux préoccupations autonomiques, les auteurs prennent en compte des fichiers de configuration spécifiques permettant d'écrire des politiques de haut niveau. En plus de l'aspect robustesse, les auteurs envisagent également la prise en compte de spécifications de QdS

pour leur plateforme.

Mise en oeuvre Selon le *Transparent shaping*, des *hooks* sont insérées dans le code de base afin d'appeler du code adaptatif. Les auteurs définissent une application avec des indirections comme une application *adapt-ready*. Cette adaptation est transparente car elle préserve les fonctionnalités liées au comportement original de l'application et ne mélange pas le code fournissant un nouveau comportement avec le code de base. Dans le cadre de la prise en charge de l'adaptation des processus BPEL, les auteurs ont développé un *framework*, appelé TRAP/BPEL, qui permet de générer des processus BPEL *adapt-ready*. De tels processus sont alors capables d'observer les invocations vers les services Web partenaires et sont augmentés par des invocations vers un proxy générique qui a pour rôle de fournir la logique d'adaptation. Ce proxy redirige alors des messages SOAP vers d'autres services lorsqu'un service Web échoue. Il est utilisé par tous les partenaires du processus et contient la logique de sélection des services. Pour cela, les politiques spécifiées dans le document de configuration sont chargées statiquement (au déploiement) dans le proxy. Les auteurs envisagent de pouvoir modifier ces politiques dynamiquement. Une politique s'applique à une activité d'invocation, ce qui permet au proxy de faire le lien entre une politique et un message SOAP losqu'il reçoit un appel. Ces politiques permettent de spécifier trois types d'action : invoquer un service particulier, rechercher et invoquer un autre service, et réessayer l'invocation en cas d'échec. Le listing 3.4 exhibe une politique effectuant des rappels à un service en cas d'erreur.

```
<Policy>
  <Service>
    <InvokeName value="WS-Invoke"/>
    <WsdlUrl preferred="true" value="http://.../WS-Description.wsdl"/>
    <Timeout seconds="2"/>
    <MaxRetry value="2"/>
    <RetryInterval seconds="5"/>
  </Service>
  <Service> ... </Service>
</Policy>
```

Listing 3.4: Exemple de politique TRAP

Par ailleurs, des indirections sont insérées aux endroits sensibles du processus BPEL original. Lors de leur premier prototype, RobustBPEL, les auteurs avaient introduit des activités *scope* autour des activités d'invocations de manière à pouvoir récupérer les erreurs dans un « fault handler » lorsqu'une invocation échouait, comme décrit dans le listing BPEL 3.5. Un service du proxy était alors appelé pour informer le comportement lié à cet événement.

```
<scope>
  <faultHandlers>
    <catchAll>
      <invoke name="InvokeProxy" ../>
    </catchAll>
  </faultHandlers>
  <eventHandlers>
    <onAlarm for="PT15S">
      <invoke name="InvokeProxy" ../>
    </onAlarm>
  </eventHandlers>
  <invoke name="invokeApprover" .."/>
</scope>
```

Listing 3.5: Modification du BPEL en fonction de la politique

Pour la nouvelle version de leur prototype, les auteurs ont choisi d'intégrer davantage la logique d'adaptation dans le proxy et donc les activités d'invocations sont

remplacées par un appel au proxy qui s'occupe intégralement de la prise en charge de la politique. Le *monitoring* est donc intégralement effectué dans le proxy qui devient une machine virtuelle pour l'exécution des politiques.

Evaluation Bien que le domaine de préoccupations couvertes par les travaux autour de TRAP/BPEL soit encore restreint, la conception de son approche vers une meilleure séparation des préoccupations ainsi que de son processus de composition de préoccupations présentent de nombreux attraits. Tout d'abord, TRAP/BPEL cherche à rester non intrusif vis à vis du langage et de la plateforme BPEL, et vise à séparer clairement la logique d'adaptation de la logique de *workflow* afin de maximiser la réutilisabilité de chaque logique. Cette approche peut ainsi être réutilisée sur tous les moteurs BPEL sans que l'utilisateur ne se préoccupe de la manière de lier les diverses plateformes. Par ailleurs, la composition des préoccupations utilise une approche mixte qui consiste d'une part à tisser des indirections vers la logique d'adaptation aux endroits pertinents dans le code BPEL et d'autre part à intercepter les messages SOAP pour effectuer des traitements. Cette manière de procéder permet de lier les logiques de manière implicite pour l'utilisateur des politiques. L'utilisateur n'a dont pas à développer d'expertise pour apprendre à introduire ses préoccupations non fonctionnelles dans le code BPEL (à l'opposée de travaux comme AO4BPEL [CM04]).

En revanche ce travail ne prend pas en compte ni le traitement statique, ni une gestion avancée de la QdS. Par ailleurs, le langage de politique reste relativement resteint et son ouverture à d'autres préoccupations de QdS n'est pas clairement identifiée.

3.2.5 eFlow

Introduction Développé par HP, eFlow [CS01, CIjJ+00, CIjJ+00] est une plateforme pour spécifier, établir et observer la QdS de services composites, tout en fournissant également un certain nombre de fonctionnalités telles que la gestion des événements, des exceptions, des transactions ou de la sécurité. Etant une approche plus ancienne, eFlow cherche à fournir aux utilisateurs un moyen de spécifier aisément des services composites à partir de services de base, à l'instar de l'approche BPEL. Cette plateforme présente des outils permettant d'adapter les compositions, notamment à la gestion de la QdS. L'objectif de cette plateforme est de rendre les compositions de services Web dynamiques et adaptables, en fonction des services présents sur le Web et des besoins utilisateur. Sa valeur ajoutée se trouve donc en particulier dans sa capacité à créer statiquement, ou modifier à la volée, des compositions qui peuvent ainsi dynamiquement s'adapter aux changements de l'environnement d'exécution.

Modélisation Avec eFlow, une composition de services est modélisée sous la forme d'un graphe qui définit l'ordre d'exécution de noeuds. Les arcs du graphe supportent des prédicats sur les variables du processus pour évaluer les transitions entre noeuds du graphe. Les noeuds peuvent être soit de type service, soit de type décision, soit de type événement :

- Les noeuds service de type simple représentent une invocation vers un service simple ou composite. La spécification d'un noeud simple inclut un nom, un identifiant, une liste de variables que le noeud peut lire ou écrire, ainsi qu'une règle de

sélection de service, afin de sélectionner le service le plus approprié (et son fournisseur). Cette règle est écrite sous la forme d'une requête XQL [8] et est exécutée dans un répertoire de descriptions de service (les services de ce répertoire étant définis en XML). La plateforme eFlow n'impose pas de contraintes concernant la forme des requêtes XQL ni concernant les documents XML de description de service (excepté pour la définition des données d'entrée/sortie et une URI). Les noeuds service de type multiservice permettent l'activation multiple ou parallèle d'un même noeud simple. Son comportement est caractérisé d'une part par le nombre d'instances à démarrer et une condition de terminaison qui détermine quand l'exécution du noeud peut être considérée complétée. Ce type de noeud permet de mettre en relation le nombre d'invocations et l'ensemble des fournisseurs de services qui peuvent délivrer le service requis. Les noeuds service de type générique permettent d'activer l'instanciation de divers noeuds simples. Ils incluent un paramètre de configuration contenant une liste de services définie soit à l'instanciation du processus, soit à l'exécution. Ce type de noeud est utile lorsqu'il n'est pas possible de prédéterminer quel type de service doit être invoqué à l'avance.

- Les noeuds de décision permettent de spécifier des alternatives et des règles controlant le flot d'exécution. Il existe des noeuds *split* permettant de séparer le flot d'exécution pour un traitement parallèle ou bien dans le cas de branches conditionnelles. Des noeuds *join* sont alors responsables de la synchronisation de deux ou plusieurs branches. Enfin des noeuds génériques permettent de spécifier des exigences de routage plus complexes.
- Les noeuds d'événement permettent aux processus de recevoir et d'envoyer des messages. La plateforme eFlow possède un modèle riche d'événements qui permet de capturer des événements dans des variables. Il existe ainsi de nombreux types d'événement : « *workflow* » (traitement du processus), « data » (modification de données), « application-specific » (exceptions provenant d'un service ou du processus), « temporal » (à certains moments). Il existe aussi des noeuds qui permettent d'envoyer (Notify) et de recevoir (Request) ces événements.

Grâce aux capacités d'adaptation de cette modélisation (telles que les noeuds génériques ou multiservice, les règles de sélection de service, ou encore la gestion des événements), il est possible de définir de manière flexible une composition de services, ainsi que de la reconfigurer dynamiquement à l'exécution ou encore de changer les services appelés par les noeuds service. Cependant, il existe toujours des cas pour lesquels le schéma de processus doit être modifié. Pour cela, les auteurs de eFlow proposent deux façons de modifier le processus à l'exécution : soit il s'agit de changements *ad-hoc* et eFlow permet à l'utilisateur de modifier le schéma ou les valeurs à l'exécution, soit il s'agit de changements exceptionnels qui peuvent affecter plusieurs instances en même temps et dans ce cas l'utilisateur peut spécifier une règle pour sélectionner ces instances de processus et les faire évoluer.

Architecture Décrite à la figure 3.3, la plateforme eFlow est principalement constituée des trois composants suivants :

8. XML Query Language

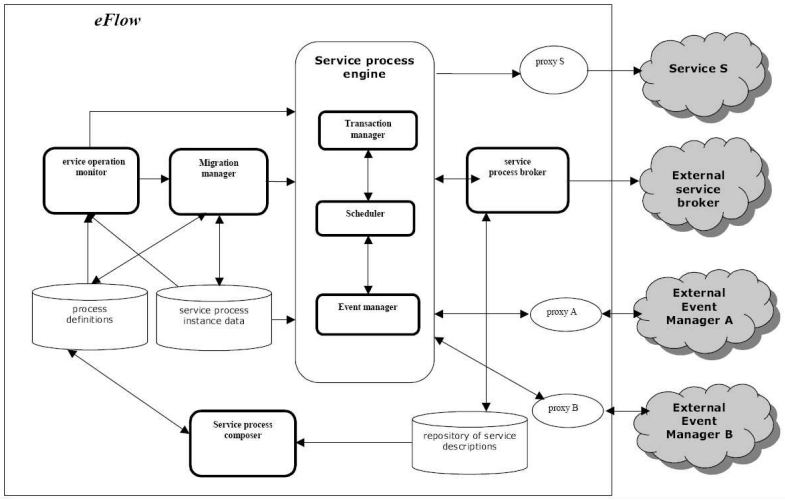

FIGURE 3.3: Plateforme eFlow

– « Service Process Composer » qui permet à l'utilisateur de définir et de modifier des schémas de processus à l'exécution.
– « Service Process Engine » qui a la responsabilité d'exécuter les processus déployés. Il s'agit du composant majeur de la plateforme eFlow et correspond à un moteur BPEL. Les graphes sont exécutés par un sous composant « Scheduler » tandis que les événements sont traités par un « Event Manager » et que les transactions possèdent leur propre sous composant de gestion, « Transaction Manager ».
– « Service Process *broker* » qui fait le lien entre le moteur d'exécution et les services Web existants. C'est notamment ce composant qui exécute la requête XQL et dialogue avec les fournisseurs de service. Il est en lien avec un répertoire de services qu'il peut contacter.

Evaluation La plateforme eFlow possède de nombreux atouts concernant la flexibilité et l'adaptation dynamique des compositions de services. Tout d'abord, sa modélisation permet de définir des règles là où les autres solutions imposent des spécifications rigides. Il devient ainsi possible pour l'utiliseur de spécifier à un niveau plus abstrait les services qui doivent être mis en oeuvre dans sa composition. Une contribution originale de la plateforme eFlow est d'intégrer la préoccupation de négociation des services avec les fournisseurs de service. En particulier, l'accent a été mis sur la capacité à administrer la composition quelle que soit la phase d'exécution puisqu'il est même possible de venir modifier le *workflow* à l'exécution. En outre, bien que le formalisme utilisé pour spécifier et exécuter des processus ne soit pas le BPEL, des techniques peuvent être réutilisées (telles que les règles XQL ou bien l'appel à plusieurs fournisseurs de service).

Néanmoins, les préoccupations liées à la définition du processus et aux règles d'adaptation sont fortement couplées ce qui diminue la réutilisabilité de ces logiques. Par ailleurs, bien que cela semble possible, ce travail n'a pas abouti à une étude plus spécifique sur la prise en charge de paramètres non fonctionnels et en particulier de la QdS. En outre, étant relativement ancien, il ne permet pas de profiter des travaux et des

spécifications développés pour les technologies BPEL et les services Web.

3.3 Plateformes de traitement spécifique de la QdS dans les compositions

Dans cette section, nous étudions cinq travaux réalisant l'étude des traitements spécifiques de la QdS dans les compositions. Il s'agit de travaux s'intéressant en particulier à la gestion des caractéristiques de performance et à la sélection des services des compositions.

3.3.1 AgFlow

Introduction Ce travail [ZBN+04, ZBD+03] cherche à apporter des réponses à plusieurs challenges fondamentaux liés à la QdS des compositions. Tout d'abord, il effectue une étude de plusieurs caractéristiques de QdS chez les services Web. Ensuite, il pose le problème de la sélection des services Web appropriés pour permettre à un utilisateur d'obtenir certaines propriétés de QdS dans sa composition de services (minimiser le prix, maximiser la réputation des services). Finalement, ce travail souligne l'intérêt d'une approche dynamique pour la prise en compte des changements de QdS inhérents à la variabilité du Web et des applications. Les contributions qui en découlent consistent en un modèle multi-dimensionnel capturant les propriétés de QdS et exhibant des méthodes pour attacher des valeurs à ces propriétés dans le cadre de services simples et composites. Deux approches (locale et globale) pour la sélection de services dans les compositions sont ensuite détaillées et un moteur d'exécution,« AgFlow », qui réagit aux changements en replanifiant les services à l'exécution est présenté.

Modélisation de la QdS La modélisation des services composites se fait sous la forme de *statecharts* car d'une part ils permettent d'exprimer les structures de contrôle habituellement offertes par les langages modélisant des processus (branches, fils d'exécution parallèles, boucles) et d'autre part car ils possèdent une sémantique formelle. Les états sont similaires aux activités des diagrammes BPEL4WS, étant soit simples (tâche) soit composées (structure), et permettent de former des chemins d'exécution sous la forme de graphes acycliques orientés. A chaque noeud t_i d'un graphe, est associé un ensemble de services s_j pouvant potentiellement être exécutés pour effectuer la tâche correspondante. Ces graphes enrichis $\{< t_1, s_1 >, < t_2, s_2 >, ..., < t_n, s_n >\}$, appelés « plans d'exécution », sont alors utilisés pour le calcul de la QdS des services composites et le choix du service le plus approprié pour chaque noeud. Pour cela, les auteurs considèrent les caractéristiques de QdS suivantes :

- Prix d'exécution $q_{pr}(s)$: coût pour une exécution du service s. Le prix $q_{pr}(p)$ pour un plan d'exécution p est $\sum_{i=1}^{N} q_{pr}(s_i)$.
- Durée d'exécution $q_{du}(s)$: agrégation de la durée moyenne de transmission (déduite des précédentes exécutions) et du temps de traitement $q_{du}(s) = T_{process}(s) + T_{trans}(s)$. La durée d'exécution pour un plan d'exécution p est calculée en utilisant l'algorithme du chemin critique (recherche du chemin le plus long en durée dans ce cas), soit $q_{du}(p) = CPA(p, q_{du})$.

- Réputation $q_{rep}(s)$: correspond à une mesure de sa confiance. Elle dépend principalement de l'expérience de l'utilisateur. Ainsi, selon le rang R_i attribué à un service pour un utilisateur i, une évaluation de la réputation peut être $\frac{\sum_{i=1}^{n} R_i}{n}$. La réputation d'un plan d'exécution p correspond à la moyenne des réputations des services impliqués dans le graphe : $q_{rep}(p) = \frac{1}{N} \sum_{i=1}^{N} q_{rep}(s_i)$.
- Taux d'exécution avec succès $q_{rat}(s)$: probabilité que le service réponde avec succès. Ce taux est obtenu par division du nombre $N_c(s)$ d'invocations réussies (dans un certain laps de temps) par le nombre total d'invocations effectuées K, soit $q_{rat}(s) = \frac{N_c(s)}{K}$. Le taux d'exécution avec succès d'un plan d'exécution p correspond au produit des taux d'exécution avec succès des services appartenant au chemin critique, soit $q_{rat}(p) = \prod_{i=1}^{N} (q_{rat}(s_i)^{z_i})$ où $z_i$vaut 1 si le service appartient au chemin critique et 0 sinon. Si un service n'appartient pas au chemin critique, son taux d'exécution avec succès n'est donc pas pris en compte en faisant l'hypothèse qu'il est possible d'exécuter à nouveau ce service sans affecter le taux global.
- Disponibilité $q_{av}(s)$: probabilité qu'un service soit disponible, calculée en divisant le temps $T_{av}(s)$ pendant lequel le service est accessible sur une période δ, constante fixée par l'administrateur. La disponibilité d'un plan d'exécution p est le produit des disponibilités des services appartenant au chemin critique, soit $q_{av}(p) = \prod_{i=1}^{N} (q_{av}(s_i)^{z_i})$ où $z_i$vaut 1 si le service appartient au chemin critique et 0 sinon. Si un service n'appartient pas au chemin critique, sa disponibilité n'est donc pas prise en compte en faisant l'hypothèse qu'il est possible de sélectionner à nouveau ce service sans affecter le taux global.

Recherche de services La modélisation de la QdS proposée par ce travail est exploitée pour la recherche des services les plus appropriés pour être utilisés dans une composition de services. En particulier, une approche locale et une approche globale sont proposées pour optimiser la sélection des services.

La recherche **locale** consiste à trouver pour chaque tâche, indépendamment des autres, le service le plus approprié. Pour chaque service candidat, les informations de QdS sont calculées et agrégées sous la forme d'un « vecteur de QdS », puis une prise de décision multi-critères est appliquée (pondération de chaque critère et application de contraintes utilisateurs). Cette méthode permet ainsi de définir des temps de réponse limite ou des prix maximum. Il est ensuite possible de classer les services selon le résultat du produit scalaire des « vecteurs de QdS » de chaque service avec le vecteur de pondération des critères de QdS. En revanche, il n'est pas possible avec cette stratégie de spécifier des contraintes sur des ensembles de tâches. Avec la recherche locale, bien que la sélection soit optimale pour chaque tâche, la QdS globale de la composition peut s'avérer sous-optimale. C'est notamment le cas lorsque deux tâches s'effectuent en parallèle. Le déroulement de l'ensemble ne peut alors pas être plus rapide que la plus longue des deux tâches. En cherchant à optimiser localement la plus rapide des deux tâches, la durée de l'ensemble n'est pas réduite, tandis que d'autres critères peuvent être sous-optimisés (*i.e* le coût).

L'approche **globale** tend à résoudre ces limitations en prenant en compte la composition dans son ensemble, notamment en appliquant, pour chaque chemin d'exécution potentiel, les règles d'agrégation du modèle de QdS. A l'instar de l'approche locale, une

prise de décision multi-critères est appliquée sur le résultat composite pour décider des services permettant d'atteindre une QdS optimale. Dans le cas où plusieurs chemins d'exécution pouvant être parcourus offrent des plans d'exécution optimaux différents, un algorithme de fusion est appliqué selon la fréquence d'utilisation des divers chemins. Par ailleurs, si le chemin d'exécution contient des boucles, les cycles sont « dépliés » sous forme de séquences de tâches dont le nombre est déterminé par l'utilisation faite de la composition.

Optimisation et Dynamicité La recherche globale requiert de réaliser une étude exhaustive sur tous les chemins qu'il est possible de parcourir et avec tous les services qu'il est possible d'utiliser pour chaque tâche. Afin d'optimiser cette étude très couteuse, une résolution par programmation linéaire a été envisagée par les auteurs. Ainsi, pour chaque critère, il est possible de définir un ensemble de variables, une fonction objectif et un ensemble de contraintes où la fonction objectif et les contraintes doivent être linéaires. Le résultat est la valeur maximum (ou minimum) de la fonction objectif ainsi que les valeurs des variables à ce maximum (ou minimum). Les formules d'agrégation sur les propriétés de QdS sont utilisées comme contraintes (avec un passage au logarithme lorsque la formule n'est pas linéaire), tandis que les valeurs de QdS des services servent comme ensemble de valeurs possibles. La fonction objectif consiste à maximiser la prise de décision multi-critère de la QdS de la composition.

Un premier plan d'exécution est construit par recherche globale lors de la première exécution de la composition. Ensuite, lorsqu'il se produit des variations de QdS des services, une nouvelle recherche est effectuée sur la région du plan d'exécution qui n'a pas été encore exécutée. Cette recherche peut également être effectuée via la méthode de programmation linéaire présentée, à ceci près que de nouvelles contraintes sont ajoutées pour fixer les paramètres liés aux services qui ont déjà été mis en oeuvre à ce stade de la composition. Un prototype, nommé AgFlow, contient l'implémentation de ces algorithmes ainsi que le moteur d'exécution de composition lié au formalisme utilisé.

Evaluation Ce travail apporte une importante contribution concernant la modélisation de la QdS dans les compositions de services. En particulier, ce travail fournit une définition de cinq critères de QdS et les formules d'agrégation de ces critères, ainsi que des algorithmes de recherche et d'optimisation. En outre, la plateforme d'exécution AgFlow, développée pour l'implémentation de ces concepts, prend en compte le mécanisme de publication des SLA et s'occupe de la QdS de la composition à la fois statiquement et dynamiquement. Par ailleurs, bien que le formalisme utilisé pour spécifier et exécuter des compositions de service ne soit pas le langage BPEL4WS, les auteurs considèrent la projection de ce langage sur leur formalisme basé sur les *statecharts*.

Cependant la sélection locale et globale des services possèdent toutes les deux un certain nombre de limitations. Tout d'abord, si la recherche locale permet de prendre en compte des exigences fines sur les services de la composition, elle ne permet pas d'optimiser la QdS globale et ne peut pas considérer de contraintes sur la globalité de la composition. En ce qui concerne la recherche globale, bien qu'une optimisation basée sur la programmation linéaire ait été mise en oeuvre, cette stratégie reste très couteuse et est difficilement envisageable dans le cadre d'une replanification dynamique pour de nombreux services. En outre, la recherche globale permet d'optimiser (maximiser ou

minimiser) des critères de QdS sur le résultat (service composite) mais ne permet pas de spécifier des contraintes à une échelle intermédiaire ou locale.

3.3.2 ORBWork

Introduction Lorsque de nouveaux services sont créés par assemblage de services existants, il est souhaitable d'être en mesure d'estimer, d'observer et de controler la QdS délivrée à l'utilisateur final. Ainsi, les auteurs de ce travail [CSM+04, CMSA02] se fixent pour tâche de permettre de déduire et de gérer la QdS des *workflows* de services Web. Pour cela, un modèle prédictif de QdS basé sur la QdS de tâches atomiques a été élaboré, ainsi qu'une implémentation, ORBWork, basée sur un moteur d'exécution de *workflow*. Ce modèle et cette plateforme sont illustrées par la mise en oeuvre de trois critères de QdS. Ce travail met en avant une méthode basée sur la réduction des graphes pour la prédiction de la QdS globale, et prend en compte l'adaptation de prédiction de la QdS à l'exécution.

Modélisation et Estimation de la QdS Dans le modèle de QdS présenté dans ce travail, les tâches du *workflow* se voient attribuées des caractéristiques de QdS. Ainsi, pour une tâche t, il est possible de définir :

- Le temps de la tâche $T(t) = DT(t) + PT(t)$, où DT représente le temps d'attente pour la mise en oeuvre de la tâche (*Delay Time*) et PT le temps nécessaire pour réaliser la tâche (*Process Time*). Le fait de découper le temps de la tâche en plusieurs informations permet de réaliser un modèle plus détaillé pour les analystes et de surcontraindre la QdS du *workflow*.
- Le coût de la tâche $C(t) = EC(t) + RC(t)$, où EC et RC désignent respectivement le coût de mise en oeuvre (*Enactment Cost*) et le coût de réalisation (*Realization Cost*). Ces informations permettent d'effectuer la distinction entre d'une part les coûts de déploiement, de maintenance, de *monitoring*, et de valeur ajoutée, et d'autre part des coûts réels associés avec l'exécution de la tâche.
- La robustesse de la tâche $R(t) = 1 - (SF(t) + PF(t))$ représente la capacité d'une tâche à ne pas échouer. Dans cette formule, SF représente le taux d'erreurs système (*System Failures*) qui correpond aux erreurs causées par la technologie ou les plateformes, tandis que PF caractérise le taux d'erreurs du processus (*Process Failures*) qui surgissent lorsque la tâche n'aboutit pas à un état final.

Une fois les dimensions de QdS définies pour une tâche, une étape d'estimation de la QdS des tâches a lieu afin de pouvoir appliquer les algorithmes de prédiction. La spécification de la QdS est effectuée à la conception et recalculée à l'exécution. Ce sont l'analyste métier et l'expert du domaine qui ont la charge d'effectuer ces estimations pour chaque tâche lors de la construction du *workflow*. Dans les cas simples, l'estimation se base sur les métriques de QdS affichées par le service correspondant à la tâche. Dans d'autres cas, notamment quand la tâche dépend beaucoup de l'environnement ou de l'utilisateur, les estimations sont effectuées sur la base d'expériences réalisées en testant la tâche. A l'exécution, le système garde la trace des métriques de QdS et les valeurs sont réajustées pour plus de précision.

Prédiction de la QdS du *workflow* En se basant sur la modélisation et les estimations de QdS de ce travail, les auteurs développent une méthode de prédiction de la QdS

d'un *workflow* par le biais d'un algorithme de réduction de *workflow* stochastique. Cet algorithme consiste à réduire le graphe du *workflow* en appliquant, par itération, des règles d'agrégation de la QdS des tâches afin d'obtenir ultimement une unique tâche dont la QdS reflète la QdS de l'ensemble du *workflow*. Ces règles d'agrégation sont décrites dans le tableau 3.1 Afin de pouvoir systématiquement effectuer une réduction jusqu'à l'étape finale, la construction du *workflow* analysé est restreinte à l'utilisation d'un ensemble de six patrons de construction (pouvant ainsi être réduits). Les règles de réduction appliquées à la QdS des six patrons sont exposées dans le tableau 3.1.

Patrons	Temps	Coût	Robustesse
Séquence	$\sum_{1\leq i\leq n} T(t_i)$	$\sum_{1\leq i\leq n} C(t_i)$	$\prod_{1\leq i\leq n} R(t_i)$
Parallèle	$\max_{1\epsilon\{1..n\}}\{T(t_i)\}$	$\sum_{1\leq i\leq n} C(t_i)$	$\prod_{1\leq i\leq n} R(t_i)$
Condition	$\sum_{1\leq i\leq n} \omega_i * T(t_i)$	$\sum_{1\leq i\leq n} \omega_i * C(t_i)$	$\sum_{1\leq i\leq n} \omega_i * R(t_i)$
Boucle	$\frac{T(t_i)}{1-p_i}$	$\frac{C(t_i)}{1-p_i}$	$\frac{(1-p_i)*R(t_i)}{1-p_iR(t_i)}$
Tolérance aux fautes	$\min_{1\epsilon\{1..n\}}\{T(t_i)\}$	$\sum_{1\leq i\leq n} C(t_i)$	$\Phi(R_1,..,R_n)$
Réseau (boîte noire)	$T(t_i)$	$C(t_i)$	$TR(t_i)$

TABLE 3.1: Règles d'agrégation

Implémentation Afin de mettre en oeuvre la modélisation, l'estimation et l'algorithme de réduction, les auteurs ont créé une plateforme en modifiant certains composants du moteur ORBWork appartenant au système METEOR [AVMM04]. Dans ce moteur, des gestionnaires de tâche sont responsable des métriques de QdS à l'exécution. Quand une tâche est prête à être exécutée, le gestionnaire active un composant propre à l'évaluation de chacune des caractéristiques de QdS. De cette manière le système peut surveiller le temps d'exécution (agrégation des durées pour la réalisation de la tâche) et la robustesse (surveillance des exceptions pendant le déroulement de la tâche), tandis que le critère de coût ne peut être modifié. Les informations recueillies par le *monitoring* sont stockées dans une base de données, où sont déjà incorporées les informations de QdS spécifiées à la conception du *workflow*. Les informations sont donc mises à jour de façon constante et l'utilisateur du moteur a accès à ces informations pour éventuellement les manipuler. Enfin, l'outil original de conception du moteur a été modifié afin d'intégrer une interface par le biais de laquelle l'utilisateur peut rentrer des informations de QdS en ce qui concerne les probabilités des transitions, le nombre de boucles et les métriques initiales.

Evaluation L'orientation de ce travail vers la prise en compte de la QdS à un niveau *workflow* est particulièrement pertinente dans le cadre des compositions de services telles qu'elles sont décrites avec le langage BPEL4WS. En ciblant directement les tâches du *workflow*, il est possible d'initialiser tous les paramètres de QdS et les informations utiles. En particulier, ce travail fait clairement apparaître deux niveaux de QdS (via la distinction du rôle de l'analyste métier et de l'expert du domaine) : le niveau propre à la mise en oeuvre de la tâche et celui propre à sa réalisation (contraint par les caractéristiques de QdS du service appelé). Ceci permet de surcontraindre le *workflow* et donc de spécifier de nouvelles exigences en plus de celles des services de base. Par

ailleurs, il est appréciable que ce travail offre une interface de spécification statique pour donner des estimations de QdS et que ces informations (éventuellement modifiées lors de l'exécution) puissent à nouveau être accessibles à l'exécution. Par ailleurs, un effort a été effectué pour que l'architecture prenant en charge la QdS dans la plateforme ORBWork soit le plus faiblement couplé au moteur d'exécution.

En revanche, ni la recherche de services ni l'adaptation des services à l'exécution ne sont envisagées (seule la prédiction de la QdS globale est réestimée à l'exécution). L'application de l'algorithme de décomposition pour la recherche de services appropriés pour satisfaire des contraintes globales n'est pas évoqué. Par ailleurs, pour le moment, ce travail ne s'inspire pas des travaux effectués sur les SLA, dont il pourrait bénéficier pour effectuer des estimations ou bien encore pour une potentielle recherche de services.

3.3.3 WS-Binder

Introduction WS-Binder [PEV+06, CPE+06, CPEV05, CPEV05] est un travail qui s'inspire des travaux présentés dans les sections 3.3.1 et 3.3.2. Il s'intéresse aux problèmatiques de recherche statique et dynamique de services dans les compositions. Ce travail réutilise un certain nombre de contributions faites par les travaux précédents (notamment la modélisation de la QdS et les formules d'agrégation), améliore certaines propositions (découpage du *workflow* pour réduire l'espace de recherche à l'ensemble des services n'ayant pas encore été invoqués), et propose des contributions originales dont un algorithme modulable pour la sélection de service, un modèle de déclenchement de la replanification et la gestion de métriques spécifiques à un domaine. Ce travail propose une plateforme (WS-Binder) qui contient l'implémentation de ces propositions et qui travaille en parallèle du moteur d'exécution BPEL.

Modélisation de la QdS Pour modéliser les propriétés de QdS et la spécification des formules d'agrégation, WS-Binder réutilise les travaux de Cardoso [CSM+04] et ajoute sur certaines parties du processus BPEL des annotations permettant d'apporter des informations supplémentaires (taux de probabilité de choix de chemin, nombre de répétitions pour une boucle). Par ailleurs, les autres propriétés de QdS (pouvant être spécifiques au domaine d'application du service) sont également prises en compte à la condition d'être composables sur un *workflow*. Dans [CPE+06], les auteurs présentent un langage pour spécifier de tels attributs de QdS ainsi que les formules d'agrégation correspondantes. Il est ainsi possible de choisir un type particulier parmi un ensemble de types proposés (entier, réel, booléen, ensemble, tableau, etc.) et des domaines de définition (intervalle, taux, nombre absolu) puis de préciser à l'aide d'un éditeur de fonctions d'agrégation comment les attributs ainsi créés et les annotations sur les activités se composent pour déduire la QdS des activités complexes (*Sequence*, *Switch*, *Fork*, *Loop*).

Recherche de services La recherche des services a pour but d'obtenir un service composite dont les propriétés de QdS sont optimales et supportent certaines contraintes. Pour cela, l'algorithme proposé utilise les formules d'agrégation définies par l'utilisateur et se base sur plusieurs critères à résoudre. Il s'agit d'une part de maximiser une fonction d'utilité et d'autre part de satisfaire des contraintes locales et globales. Afin d'optimiser la recherche, les auteurs proposent d'appliquer un algorithme génétique en modélisant

les services abstraits comme un génome où chaque service abstrait peut prendre comme valeur l'un des services concrets qui lui est associé. Dans ce contexte, la fonction d'utilité permet d'associer au génome (*i.e* une configuration particulière de services concrets) un score dont la valeur est fonction des propriétés de QdS. L'algorithme génétique réalise alors des transformations par itération du génome (par enjambement et mutation) de manière à optimiser ce score. Le processus peut aboutir lorsque les contraintes associées au service composite sont satisfaites, ou bien continuer afin d'optimiser davantage la QdS. Par exemple, avec des propriétés de QdS classiques, la fonction d'utilité peut avoir la forme $F(g)= \frac{\omega_1 Disponibilit}{\omega_2 Cot + \omega_3 TempsDeRponse}$ où les paramètres ω_x sont des pondérations estimées par l'utilisateur suite à des expériences. Dans ce cas, l'algorithme cherche à maximaliser la fonction $F(g)$, et chaque nouvelle configuration de services obtenu par modification doit permettre d'augmenter le score de la fonction d'utilité pour constituer « le nouvel individu ». Cette solution permet d'aboutir à des solutions sous-optimales, l'avantage étant que l'utilisateur de l'algorithme peut établir un compromis entre d'une part l'optimalité des propriétés de QdS du service composite et d'autre part le temps de recherche.

Processus de replanification Le processus de replanification fait appel à deux techniques en plus de la mise en oeuvre de l'algorithme génétique. Tout d'abord, les auteurs ont conçu un algorithme de déclenchement de replanification. Cet algorithme correspond à une recherche d'événement susceptible d'impliquer une variation de la QdS globale de la composition et donc une violation possible du SLA du service composite. L'algorithme est exécuté à chaque fois qu'une nouvelle information est disponible. Il varie légèrement pour chaque propriété de QdS car l'impact de la variation de certains paramètres à l'exécution n'est pas toujours le même sur chacune des propriétés. Sa mise en oeuvre consiste à parcourir le *workflow* en visitant récursivement les noeuds les plus complexes jusqu'aux activités les plus imbriquées. La replanification est alors déclenchée si la différence entre les valeurs de QdS obtenues à l'exécution et celles de départ est supérieure à un seuil. Dans ce cas, les auteurs ont développé une stratégie similaire à [ZBN+04] de manière à n'effectuer la replanification que sur une portion particulière du *workflow* et être plus efficace. Ainsi, un algorithme de délimitation de la portion du *workflow* à replanifier a été développé et l'algorithme génétique pour la recherche de services est appliquée à cette portion.

Une plateforme, WS-Binder, décrite dans la figure 3.4, contient l'implémentation de ces algorithmes et agit comme un proxy permettant de rediriger les invocations du moteur BPEL vers les services Web. Les sondes utilisées pour remonter les variations de paramètres dans l'algorithme de déclenchement de la replanification sont introduits dans le *workflow* BPEL.

Evaluation WS-Binder contribue dans des domaines originaux à la gestion de la QdS dans les compositions de services. Tout d'abord, ce travail permet aux utilisateurs de pouvoir prendre en compte des propriétés de QdS spécifiques au domaine de leur application et de préciser comment ces propriétés se composent selon les activités du *workflow*. Ensuite, l'utilisation d'un algorithme génétique laisse une certaine flexibilité dans la gestion du compromis optimalité des propriétés QdS globales obtenues par rapport à la baisse de performance du système lié à cette recherche. La solution peut

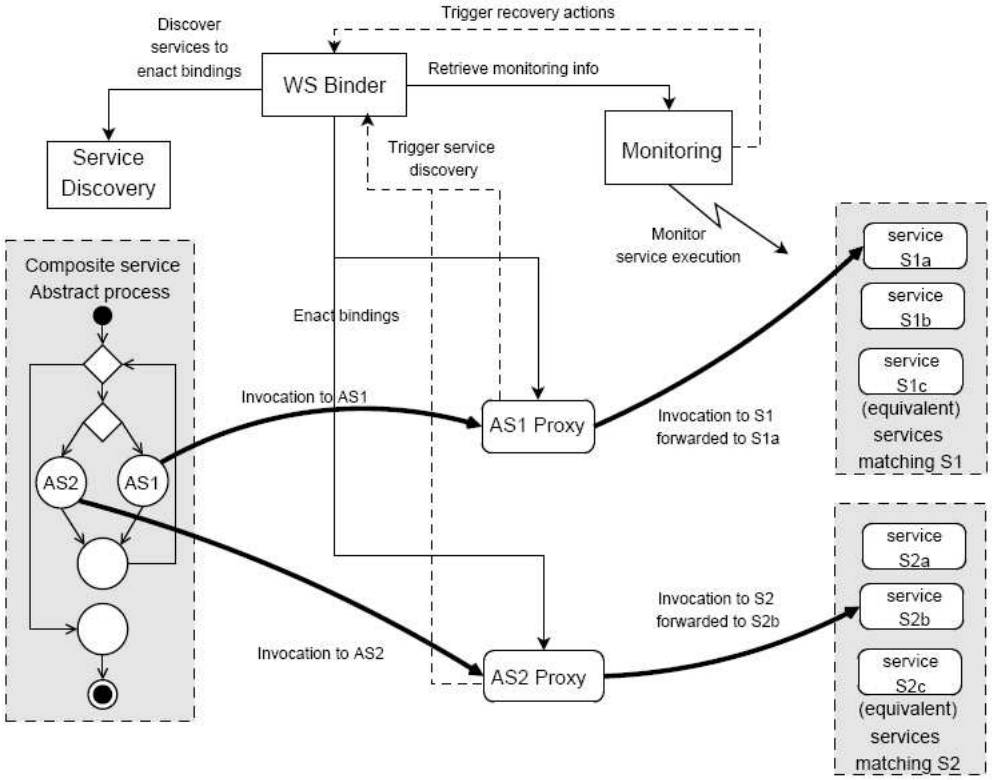

FIGURE 3.4: Processus de mise en oeuvre de la plateforme WS-Binder

ainsi prendre en compte la recherche statique et dynamique des services. En outre, l'algorithme de calcul de la portion du *workflow* à replanifier permet de gagner en performance lors du processus de recherche dynamique, tandis que l'algorithme de déclenchement de la replanification permet de mieux configurer l'adaptation à l'exécution. Enfin la plateforme WS-Binder est non intrusive avec les plateformes déjà existantes, les sondes étant injectées sous la forme d'appel à des services Web depuis le document BPEL et le proxy faisant le lien entre invocations du moteur BPEL et services Web.

En revanche, cette solution n'exploite pas tout le potentiel des SLA puisqu'elle ne permet pas la mise en oeuvre de mécanismes WS-*. Par ailleurs, les algorithmes de replanification et de déclenchement de la replanification étant fixés, WS-Binder ne laisse pas à l'utilisateur l'opportunité de spécifier les événements et les réactions qui lui seraient les plus appropriés pour la prise en compte de l'adaptation dynamique de la QdS de la composition.

3.3.4 QoS-Optimised Web Service Compositions

Introduction Les travaux de Michael Jäger et al. [Jae07, JRGM04] portent sur la prise en compte des propriétés de QdS dans les compositions et sur leur optimisation. En particulier, ces travaux mettent en relief différentes heuristiques qu'il est possible de mettre en oeuvre afin d'obtenir une QdS optimale tout en prenant en compte l'effort

du calcul requis. Ce travail fait apparaître une étude aboutie sur les compositions, la QdS, l'agrégation de la QdS dans les compositions, ainsi que la sélection des services selon la QdS. Parce que ce travail propose un niveau d'abstraction plus élevé, il permet de prendre en compte une plus grande variété de situations. En particulier, il diffère des autres approches car restant à un niveau théorique et ne proposant pas de plateforme pour la mise en oeuvre des divers algorithmes présentés.

Modélisation En se basant sur une analyse des *workflows*, ces travaux font émerger un modèle structurel pour représenter les compositions de services. Ce modèle permet d'exhiber plus d'informations pour une meilleure prise en charge de la QdS, et en particulier de son agrégation sur les structures de la composition. Chacune des structures qu'il fournit sont appelées des « patrons de composition » et des règles d'agrégation sont définies sur chacun de ces patrons pour différentes situations (pire cas, valeur moyenne, valeur maximale). Les auteurs ont ainsi fait émerger neuf patrons de composition selon l'arrangement des activités et leurs conditions d'activation. Le modèle structurel de composition consiste alors en un graphe orienté qui peut être replié en un seul noeud en appliquant récursivement les différents patrons de composition. Par conséquent, une structure de composition peut être définie comme un ensemble K_1 contenant des tâches $t_1, ..., t_i$ ou d'autres sous-ensembles $K_2, ..., K_j$(pouvant à leur tour contenir des tâches ou des sous-ensembles). Chacun des ensembles K_x est associé à un patron de composition particulier parmi les neuf existants $CP_1, ..., CP_9$. Pour chaque patron de composition et chaque propriété de QdS, il existe une règle d'agrégation. Ces règles ont été spécifiées pour les propriétés de QdS suivantes : débit, temps de réponse, coût, disponibilité et fiabilité, réputation et fidélité, niveau de cryptage.

Recherche de services Cette modélisation et les règles d'agrégation forment la base d'une modélisation étendue à la prise en compte de la définition de la QdS dans les compositions de services et du problème de la sélection des services afin d'optimiser la QdS globale. Le problème de la sélection des services est ainsi mis en relation avec des problèmes de combinatoire connus (sac à dos, gestion de projet, planification de requêtes, routage internet). Les résultats de ces travaux montrent que bien souvent le problème de sélection de service ne peut pas en totalité ou en partie être transformé en un problème combinatoire connu. Le problème de la sélection de service est ensuite traité par diverses heuristiques cherchant à se rapprocher de l'optimum global pour un coût de recherche moindre. Ainsi, les auteurs identifient quatre heuristiques permettant de traiter, de manière sous-optimale, le problème de la sélection de services :

- Algorithme gourmand : cet algorithme s'apparente à la recherche locale des travaux de [ZBN+04]. Il consiste à évaluer chaque service de chaque tâche avec une fonction d'évaluation (en pondérant et en équilibrant les différentes propriétés de service) afin d'obtenir un score qui peut être comparé. Une fois que chaque service de chaque tâche a été évalué, le service ayant obtenu le meilleur score pour chaque tâche est le service choisi pour effectuer la tâche. Cet algorithme ne permet pas de considérer des contraintes globales tout en optimisant la QdS, et sa complexité est de l'ordre $\Theta(n \log n)$.
- Algorithme d'élagage : cet algorithme utilise un arbre de recherche dont les noeuds représentent une paire possible formé d'un candidat et d'une tâche. Chaque ni-

veau de l'arbre possède des paires d'une même tâche. Chaque chemin partant de la racine de l'arbre jusqu'à une feuille représente une configuration potentielle pour la sélection des candidats. L'intérêt de cette modélisation est de pouvoir élaguer des branches représentant des combinaisons non favorables à certaines contraintes afin de réduire l'effort de calcul. Ainsi, dès qu'un début de combinaison viole une contrainte, l'algorithme coupe la branche du noeud. Une fonction permet d'évaluer la QdS globale ce qui permet ensuite de classer les configurations potentielles. La complexité de cet algorithme est $\Theta(m^n)$ dans le pire cas et $\Theta(n^2)$ dans le meilleur cas.

- Algorithme *bottom-up* : il s'agit d'effectuer tout d'abord une sélection par rapport à une seule contrainte de QdS. Puis, pour chaque tâche, le candidat original est remplacé par le candidat qui offre la pire valeur suivante pour cette contrainte. La solution proposée est conservée si la contrainte globale n'est pas violée. Cette méthode ne trouve pas nécessairement d'optimum globale et sa complexité dans le pire cas est $\Theta(n^2)$ et $\Theta(n \log n)$ dans le meilleur cas.
- Algorithme de sélection par patron : cet algorithme tire profit de la modélisation par patron proposée par ces travaux pour modéliser les compositions de services. Il consiste tout d'abord à effectuer récursivement une recherche des patrons qui ne contiennent pas de sous-patron. Pour toutes les tâches de ces patrons, chaque ensemble de candidats est évalué jusqu'à trouver une solution optimale. Cette solution est fixée et l'algorithme recommence alors pour la patron englobant jusqu'à finalement réaliser le calcul pour le patron de plus haut niveau. Cet algorithme ne garantit pas de trouver de solution optimale car une fois que la QdS d'un patron est fixée, il n'est plus possible de l'optimiser à un niveau plus élevé de la composition. La complexité de cet algorithme est $\Theta(m^{n_{sub-max}})$ où $n_{sub-max}$ correspond au nombre de tâches dans le plus grand patron.

Evaluation Ce travail sort du cadre des simples compositions de service pour réaliser une étude approfondie et théorique du calcul de la QdS dans les processus métier. La modélisation des compositions proposée par les auteurs permet une prise en compte plus fine de la QdS et des algorithmes d'agrégation. Par exemple, il est plus complexe de traiter la QdS dans un processus modélisé en BPEL car les liens des activités *flow* ne permettent pas d'identifier les structures internes d'où une approximation plus floue dans le calcul de la QdS. Cependant, il reste possible de transcrire un processus modélisé en BPEL dans le formalisme décrit par ces travaux, en particulier en utilisant les neufs patrons de composition identifiés. La problématique de sélection des services est étudiée exhaustivement et apporte beaucoup d'informations sur les transformations possibles en algorithmes de combinatoire connus et sur les heuristiques de recherche. Ce travail fait également apparaître une étude des divers rôles qui entourent la gestion de la QdS dans les compositions de services.

Ce travail n'a en revanche pas pour objectif la conception d'une plateforme ou des traitements de mise en oeuvre de la gestion de la QdS.

3.3.5 Broker-based Framework

Introduction Ce travail [YL05a, YL05b] présente une architecture de *broker* pour l'intégration et l'adaptation dynamique des services Web dans les compositions de ser-

vices. A l'instar d'autres approches, il propose une modélisation de la QdS ainsi que des algorithmes de sélection de services basés sur les caractéristiques globales de la composition. Le *broker* exploite les informations contenues dans le SLA des services et analyse les rapports de QdS du service à l'exécution afin de vérifier si un SLA est violé. Dans ce cas, le *broker* est en charge de trouver une autre service qui satisfait la QdS globale. L'algorithme de sélection ne peut supporter qu'une seule contrainte de QdS et un seul point d'échec.

Architecture La motivation principale pour la conception du *broker* est d'intégrer la logique de sélection des services. Ainsi, la logique de composition et celle de demandeur de service sont clairement isolées. Les *brokers* sont ensuite vus comme des services maintenus de façon autonome et gérés par un acteur indépendant (à la manière d'une agence de voyage). Par la suite les *brokers* peuvent être amenés à travailler ensemble pour répondre à la requête d'un client. Après chaque transaction avec le client, l'utilisateur du service doit rapporter les performances de QdS du service pour que le *broker* puisse mettre à jour les données de QdS de chaque service. L'architecture du *broker*, exhibée dans la figure 3.5, est constituée de quatre composants :

- *Service Information Manager* garde en mémoire un ensemble de services Web candidats dans un registre constitué de deux tables (pour les informations données par le service et les données statistiques de QdS). La table service permet de stocker la description fonctionnelle des services (identifiant, URL, opération, données d'entrée/sortie, description) et les données du SLA (capacité, temps de réponse, disponibilité, fiabilité). La table des données statistiques enregistre les données transmises par les clients ayant utilisé les services (stabilité du service, disponibilité, etc.). En fonction de ces informations statistiques agrégées, les valeurs de QdS des services sont mises à jour dans la table service.
- *Composition Manager* maintient le registre de processus qui contient les plans de processus. Ces plans de processus correspondent aux services et à leurs connections, formant un chemin d'exécution. Ce composant traite également les requêtes de recherche de services pour générer un plan d'exécution. Les plans d'exécution consistent en des processus abstraits. Ils sont formés à partir des plans de processus choisis par un utilisateur. Les plans d'exécution sont utilisés par les algorithmes de sélection de service.
- *Selection Manager* retrouve les services pour le plan d'exécution généré par le *Composition Manager*. La sélection s'effectue sur la base des exigences de QdS de l'utilisateur. L'algorithme utilise une technique de résolution par contraintes sur le graphe de la composition afin de vérifier que la QdS globale (après agrégation) respecte bien les contraintes données par l'utilisateur. Pour optimiser la QdS globale obtenue, une fonction objectif est définie. Cette fonction consiste en une somme pondérée de paramètres de QdS. Plus sa valeur est élevée, plus la QdS globale est élevée. L'algorithme de résolution par contraintes a également été étendu de manière à générer pour chaque chemin optimal un second chemin (pour chaque service) au cas où un service ne soit plus accessible durant l'exécution de la composition.
- *Adaptation Manager* le résultat de la sélection étant renvoyé au moteur d'exécution de la composition, la composition peut être instanciée. Dans ce modèle

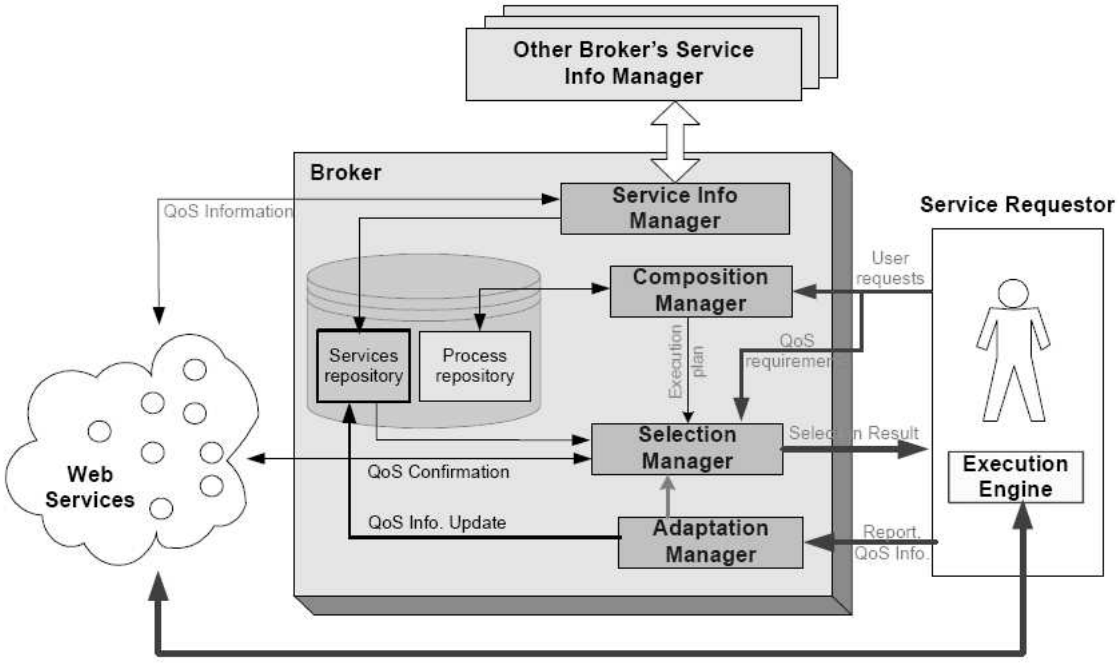

FIGURE 3.5: Architecture du framework

d'architecture, c'est au moteur d'exécution d'observer la QdS des services impliqués dans la composition. Si les SLA ne sont pas violés, alors la QdS des services est renvoyée à l'*Adaptation Manager*. Au cas où un SLA est violé, alors l'Adaptation Manager doit renvoyer un des nouveaux chemins d'exécution pré-calculés au moteur d'exécution et demande au *Selection Manager* de recalculer un ensemble de plans de rechange. La table des statistiques est également mise à jour pour tenir compte de la violation du SLA.

Evaluation Bien que ce travail soit limité sur de nombreux points (une seule propriété peut être contrainte globalement et une seule erreur ne peut être traitée à la fois), il propose une séparation très nette entre les rôles de concepteur de composition et de sélecteur de services. Le *broker* et le moteur BPEL sont clairement découplés et aucune donnée confidentielle ne transite par le *broker* puisque celui-ci n'agit pas comme un proxy, contrairement à la plupart des architectures des plateformes concurrentes. Ceci permet au gestionnaire du *broker* de servir de multiples clients et même de travailler en partenariat avec d'autres gestionnaires de *broker*. Par ailleurs, ce travail réutilise certaines informations des SLA et maintient à jour une base de données sur les statistiques de QdS des services, ce qui apporte plus de fiabilité qu'une simple relation client-fournisseur de service. Enfin, dans des conditions idéales, l'adaptation dynamique ne coûte rien puisque les chemins de rechange sont préparés en avance et donc il n'y a pas de surcharge lorsqu'un service viole son SLA.

En revanche, du fait de l'indépendance des rôles, le *broker* n'a pas de perception de la QdS à l'exécution et c'est donc au moteur BPEL d'implémenter la logique de *monitoring* et de surveillance de la disponibilité. Il est donc de la responsabilité du

client (la composition de services) de s'assurer de la QdS fournie. Cela nécessite en outre de modifier le moteur BPEL et donc de le rendre spécifique à l'utilisation du *broker*.

3.4 Conclusion

Le tableau 3.2 synthétise les évaluations des différents travaux étudiés dans ce chapitre. Pour cela, nous réutilisons les critères définis dans la section 3.1 : Réutilisation (du BPEL, des SLA et des WS-Policy), Séparation des préoccupations (isolation des spécifications, intrusivité des plateformes), Couverture (plage de caractéristiques de QdS, mécanismes de QdS, granularité), Dynamicité et Expressivité (richesse de l'interface, déclarativité). Dans ce tableau, les cases vides représentent des fonctionnalités n'ayant pas été prises en compte par le système. Dans les autres cas, les travaux sont jugés selon une notation comportant trois niveaux croissants : « − », « ± », « + ».

	Réutilisation		SdP		Couverture			Dynam.	Expressivité	
	BPEL	SLA	Isol.	Intrus.	Chara.	Méca.	Gran.		Rich.	Decl.
AO4BPEL	+	−	+	±	−	±	±	±	−	+
DYNAMO	+	±	+	−	±	±	+	±	+	+
MASC	−	±	+	−	±	−	±	±	±	−
TRAP/BPEL	+	−	+	+	±	−	−	±	−	+
eFlow	−	−	−	−	+	±	−	+	−	+
AgFlow	−	+		−	+	±	+	+	±	
ORBWork	+	−		+	+	−	+	+	±	
WS-Binder	+	+	+	+	+	±	+	+	±	+
Optimizing QoS	±	±					+	±		
broker	+	+		±	±	±	+	±		

TABLE 3.2: Synthèse les évaluations

Cette première évaluation globale permet de mettre en relief certaines ressemblances entre plateformes, correspondant à des orientations différentes dans l'approche de la gestion de la QdS dans les compositions de services.

Réutilisation des langages *vs* Nouveaux formalismes Une grande majorité de plateformes considèrent le langage BPEL comme la brique fondamentale pour la spécification des compositions de services. Ce standard, adopté par les industriels et les universitaires, constitue donc un outil indispensable pour les travaux de recherche dans ce domaine. Seuls les travaux MASC et eFlow utilisent leur propre formalisme pour définir les compositions de services. Par ailleurs, si certaines plateformes modélisant la QdS exploitent des modèles plus abstraits pour affiner leur recherche, il est toujours possible de raffiner ces modèles pour traiter spécifiquement du BPEL. En ce qui concerne l'utilisation de standards reconnus chez les services Web pour spécifier la QdS des services, certains travaux (AgFlow, WS-Binder et *broker*) considèrent les SLA, d'autres (DYNAMO et MASC) réutilisent le formalisme WS-Policy, tandis que les autres ne considèrent tout simplement pas l'utilisation de ces outils. Il faut noter que WS-Binder

propose une utilisation aboutie des SLA en permettant à l'utilisateur de spécifier des propriétés de QdS spécifiques aux domaines métiers des services ainsi que leurs règles de composition.

Logique de gestion de QdS et Intrusivité des plateformes Quelle que soit la plateforme utilisée, la logique de gestion de la QdS et celle de la gestion de composition fonctionnent en étroite collaboration. Il existe plusieurs stratégies permettant de faire cohabiter ces deux logiques. Tout d'abord, les logiques peuvent être clairement isolées l'une de l'autre, mais en revanche les plateformes d'exécution sont fortement liées à l'exécution. C'est notamment l'approche choisie par les plateformes DYNAMO et MASC qui offrent en contrepartie un meilleur accès aux informations liées à la composition. Il existe également une stratégie consistant à isoler les logiques lors d'une phase de spécification, puis, avant l'exécution, d'effectuer une « phase de préparation » qui permet d'ajouter des indirections (ou « hooks ») de la logique principale (celle de la composition) vers la logique de gestion de la QdS. C'est l'approche considérée par les travaux AO4BPEL, TRAP/BPEL et WS-Binder, qui, à l'instar de la programmation par aspects, tissent des indirections vers leur plateforme avant le déploiement du BPEL. Cette approche est particulièrement intéressante car elle tend à minimiser l'intrusivité des plateformes d'exécution tout en maximisant l'isolation des spécifications. Cependant, dans le cas d'AO4BPEL, les services Web d'infrastructure contenant les mécanismes de QdS sont fortement couplés avec la plateforme, ce qui diminue l'indépendance de la plateforme d'exécution des aspects. Enfin, une troisième stratégie consiste à mélanger explicitement les deux logiques, comme c'est le cas pour eFlow qui permet d'écrire des noeuds de composition comportant des spécifications quant à la sélection du service. Le restant des plateformes contiennent directement la logique de gestion de QdS et ne permet donc pas, ou très peu, de prendre en compte des spécifications.

Couverture de la QdS La gestion de la QdS recouvre un ensemble de domaines qui sont adressés de manière inégale par les différentes plateformes. En ce qui concerne les caractéristiques de QdS considérées par les différentes approches, il faut noter que les plateformes de la section 3.3 prennent toutes en compte de façon plus ou moins étendues les propriétés de performance telles que le temps de réponse, la disponibilité, le débit, etc. Pour les plateformes de la section 3.2, DYNAMO et MASC considèrent le temps de réponse tandis que TRAP/BPEL envisage la non disponibilité du service et que eFlow offre une plus grande ouverture grâce à son langage de requête XQL. AO4BPEL considère en particulier les caractéristiques non fonctionnelles des services Web de type sécurité, garantie de livraison et transactions. En ce qui concerne les mécanismes de QdS, alors que AO4BPEL et DYNAMO permettent d'introduire des mécanismes de QdS tels que la sécurité, une vaste majorité de plateformes permet d'effectuer du *monitoring*. Les plateformes WS-Binder, AgFlow et *broker* permettent également de gérer les SLA (négociation, violation) et toutes les plateformes de la section 3.3, sauf ORBWork, permettent de gérer la sélection de services. En ce qui concerne la granularité, il faut noter que les plateformes adaptatives permettent généralement d'effectuer des traitements sur des activités basiques du BPEL tandis que les plateformes de la section 3.3 permettent de prendre en charge la recherche des services soit localement, soit

globalement, mais rarement à un niveau de granularité intermédiaire (seul WS-Binder le permet).

Dynamicité Les plateformes ne font pas intervenir le traitement de la QdS aux mêmes moments et certaines d'entre elles sont plus permissives que d'autres pour l'adaptation de ces traitements. La plupart des plateformes proposent une prise en charge dynamique de la gestion de la QdS à l'exception des travaux ORBWork ainsi que Jaeger et al. [Jae07] qui modélisent la QdS statique pour apporter des garanties ou effectuer la sélection de services au pré-déploiement sans envisager l'adaptation à l'exécution. Les plateformes AgFlow, WS-Binder et *broker* proposent des solutions statiques pour la sélection des services ainsi que des solutions élaborées pour l'adaptation de la QdS lorsqu'il se produit des variations à l'exécution. Enfin les plateformes adaptatives ne considèrent généralement pas les traitements au pré-déploiement. Il est à noter que les plateformes AO4BPEL, eFlow et WS-Binder laissent à l'utilisateur la possibilité de modifier certaines informations à l'exécution, ce qui permet d'adapter (avec certaines contraintes) dynamiquement les traitements.

Expressivité *vs* Automatisation des traitements L'étude des travaux nous a permis d'analyser les différentes interfaces que proposent les plateformes aux utilisateurs. Ainsi, alors que certaines plateformes sont accompagnées d'un langage de spécification (AO4BPEL, DYNAMO et MASC), d'autres proposent plutôt des fichiers de configuration ou des annotations (TRAP/BPEL et eFlow). Les plateformes de traitement spécifique de la QdS ne permettent pas de rédiger des spécifications mais il est tout de même possible de modifier certains paramètres (nombre de boucles, pondération des chemins) et la prise en compte des SLA apportent une expressivité supplémentaire. En particulier, WS-Binder propose un formalisme et une interface pour la spécification des paramètres additionnels dans les compositions et dans la prise en compte des SLA. En ce qui concerne les langages proposés par les plateformes adaptatives, la plateforme DYNAMO et ses langages WSCoL et WSReL permettent de spécifier la supervision et le recouvrement de violation de manière déclarative. MASC et son langage WS-Policy4MASC est trop complexe et cherche à prendre un compte une très grande variété de traitements, ce qui le rend riche mais difficilement exploitable. AO4BPEL s'inspire à la fois de la syntaxe d'AspectJ (avec pour langage d'action le langage BPEL) et de XPath pour le langage de coupe. Ceci facilite sa prise en main, mais le fait de limiter l'expressivité à des actions BPEL réduit la richesse des traitements associés à la QdS. Enfin TRAP/BPEL propose une interface sous la forme d'un fichier de configuration, mais son expressivité reste limitée quant aux actions qu'il est possible de spécifier.

Deuxième partie

Contribution

Chapitre 4

Orientation de la thèse

Ce chapitre fait le lien entre les travaux existants et la contribution de la thèse. Il apporte tout d'abord une mise en perspective des approches déjà existantes ainsi que l'ébauche de pistes d'amélioration (section 4.1). Cette discussion sert ensuite de base pour l'élaboration des points saillants de notre approche (section 4.2). Finalement, nous présentons succinctement le processus global de notre approche, ainsi qu'un scenario de composition de services exhibant des exigences de gestion de QdS adressables par cette approche (section 4.3).

Sommaire

4.1 Mise en perspective

L'étude des approches existantes a permis de mettre en évidence un certain nombre de critères d'analyse pertinents. Ces critères demeurent tout autant pertinents pour l'orientation des travaux de la thèse. Ils servent ainsi de cadre de référence pour des discussions vers une meilleure prise en charge de la QdS dans les compositions de services.

Réutilisation de l'existant Globalement, les plateformes réutilisent peu les SLA. Seul WS-Binder [PEV+06] en fait une utilisation avancée, notamment avec la possibilité pour l'utilisateur de spécifier et de composer des propriétés de QdS de son domaine applicatif. Les plateformes adaptatives (section 3.2) ne l'envisagent pas du tout, tandis que peu de plateformes pour le traitement spécifique de la QdS (section 3.3) s'en servent pour recueillir des informations de QdS. Pourtant, dans un contexte « orienté service »,

où les services sont faiblement couplés et amenés à être remplacés par d'autres, il nous apparaît primordial de gérer convenablement le SLA qui spécifie la relation entre le client et le fournisseur de service. Une piste de recherche pour l'élaboration de notre contribution est de tirer profit des SLA afin de pouvoir sélectionner les services, définir un cadre temporel pour l'utilisation des services, spécifier des mécanismes de sécurité ou de garantie de livraison (notamment via la réutilisation des WS-Policy dans le SLA) et établir une réaction à la violation. En ce qui concerne le BPEL, bien que ce standard soit largement adopté, il est à noter que peu de travaux cherchent explicitement à donner à l'utilisateur des moyens pour adresser des spécifications directement sur les *workflows* BPEL. Or, puisque ce standard est reconnu, les travaux, notamment ceux qui offrent des langages, pourraient réutiliser un formalisme inspiré du langage BPEL afin de spécifier les activités du *workflow* dans la spécification des traitements de QdS.

Séparation des préoccupations Pour ce critère, nous avions identifié deux éléments d'étude significatifs : l'isolation des logiques, pour le niveau spécification de la gestion de la QdS, et l'intrusivité des plateformes, pour le niveau mise en oeuvre de la gestion de la QdS. En ce qui concerne l'isolation des logiques, la majorité des travaux proposant des interfaces les isolent clairement de la logique de composition de services, que ce soit par l'intermédiaire de langages ou bien de fichiers de configuration. Cependant, si l'isolation physique des logiques est bien prise en compte par les travaux, il faut toutefois constater que la séparation des domaines de préoccupation que ces logiques capturent n'est pas souvent clairement identifiée. Ceci a pour effet d'une part de rendre moins visible l'étendue exacte des préoccupations que certains travaux permettent d'adresser et d'autre part d'autoriser d'éventuelles intersections avec la préoccupation de gestion de la composition de services. C'est par exemple le cas de AO4BPEL [CSHM06] qui peut potentiellement servir à rajouter des activités BPEL ayant pour rôle d'étendre fonctionnellement les compositions de service. Un autre effet est qu'il devient difficile de savoir à qui s'adresse le langage. En effet, sur l'ensemble des travaux étudiés, seules trois approches ([CSM+04, Jae07, YL05a]) proposent une étude des rôles. Par ailleurs, en ce qui concerne la mise en oeuvre, à l'exception de WS-Binder et de TRAP/BPEL [ES06], toutes les plateformes sont couplées avec le moteur d'exécution de la composition. Nous pensons donc qu'il est souhaitable qu'une étude des rôles et du domaine de préoccupation précède la conception d'une nouvelle approche et d'une nouvelle plateforme. C'est d'abord en identifiant et en isolant clairement les préoccupations qu'il semble possible de mieux définir des interfaces de spécification en évitant les redondances pour développer des plateformes découplées.

Couverture de la QdS Pour simplifier l'étude du domaine de la gestion de la QdS, nous l'avons divisé en trois sous domaines : caractéristiques, mécanismes et granularité. Globalement, les caractéristiques et les mécanismes pris en charge sont variants d'une plateforme à une autre. En effet, il n'y a par exemple pas de plateforme permettant à la fois de gérer la sécurité et d'effectuer la sélection de service. Pourtant, le service sélectionné peut être porteur d'exigences qui modifient la stratégie de sécurité de la composition. Il est ainsi regrettable qu'aucune plateforme n'envisage à la fois la gestion des caractéristiques de performance et celle des mécanismes liés à la QdS de type sécurité ou garantie de livraison. Dès lors, nous envisageons pour notre approche de

fournir à l'utilisateur une plateforme capable de prendre en compte, de manière homogène, les diverses exigences de QdS qu'il souhaite spécifier pour sa composition. Il faut également noter que, à l'exception de Jaeger et al. [Jae07] qui présente la notion de « patrons de composition », aucune plateforme n'adresse véritablement un niveau de granularité entre le niveau global (composition dans son ensemble) et local (service de base). Une des pistes de recherche pour l'élaboration de notre approche est d'explorer les potentialités de ce niveau de granularité.

Dynamicité Les approches qui considèrent à la fois les traitements statiques et dynamiques ne sont pas majoritaires et la plupart d'entre elles sont en fait des plateformes gérant la sélection des services statiques et mettant en oeuvre une méthode de replanification globale lorsqu'un service ne délivre plus la QdS promise. A l'inverse, les plateformes adaptatives ne prennent pas en considération l'étape statique pour ne gérer la gestion de la QdS qu'à l'exécution. A l'image des contrats SLA et du langage BPEL qui sont traités à la fois statiquement et dynamiquement, il est souhaitable de prendre en compte ces deux moments de mise en oeuvre. Une piste de recherche consiste à identifier les différents traitements à mettre en oeuvre pour les moments statique et dynamique afin de mieux les prendre en compte.

Expressivité Peu de travaux considèrent l'approche langage et la majorité des plateformes tendent à automatiser la gestion de la QdS, ne laissant à leurs utilisateurs qu'une interface très restreinte pour l'expression de leurs besoins. En particulier, les plateformes implémentant la sélection de services n'offrent généralement aucune interface aux utilisateurs. A l'image de la plateforme WS-Binder, l'utilisateur n'a souvent pas la possibilité de spécifier ses algorithmes de décision ou de replanification, et ne peut pas exprimer des exigences de QdS additionnelles. Pour les plateformes proposant des langages, soit leurs langages se limitent dans leur expressivité à un sous ensemble trop restreint (TRAP/BPEL [ES06], AO4BPEL [CSHM06]), soit ils sont trop complexes à utiliser car cherchant à prendre en compte des préoccupations trop variées et hétérogènes (MASC [TEM07]). Ainsi, il ne nous apparait ni judicieux, ni réalisable, que la gestion de la QdS soit entièrement automatisée. A l'image du BPEL, l'interface idéale pour la gestion de la QdS serait à la fois homogène (grâce à une étude précise du domaine), et offerant une expressivité équilibrée permettant d'écrire des spécifications de haut niveau.

4.2 Positionnement de la contribution

Les pistes d'amélioration que nous avons isolées lors de l'étude des différents critères servent de base pour l'élaboration de notre contribution. Nous présentons ici quelques points saillants pour l'orientation de nos travaux. Tout d'abord, nous commençons cette présentation en effectuant l'analyse des différents rôles entourant la gestion des compositions de service (section 4.2.1). Ces rôles adressent des domaines de préoccupation isolés et nous nous intéressons plus particulièrement à analyser le domaine de la gestion de la QdS (section 4.2.2). Cette analyse débouche sur la perspective d'utilisation d'un langage dédié comme outil pour capturer la gestion de la QdS (section 4.2.3). Enfin, nous évoquons les caractéristiques de la mise en oeuvre statique et dynamique de ce

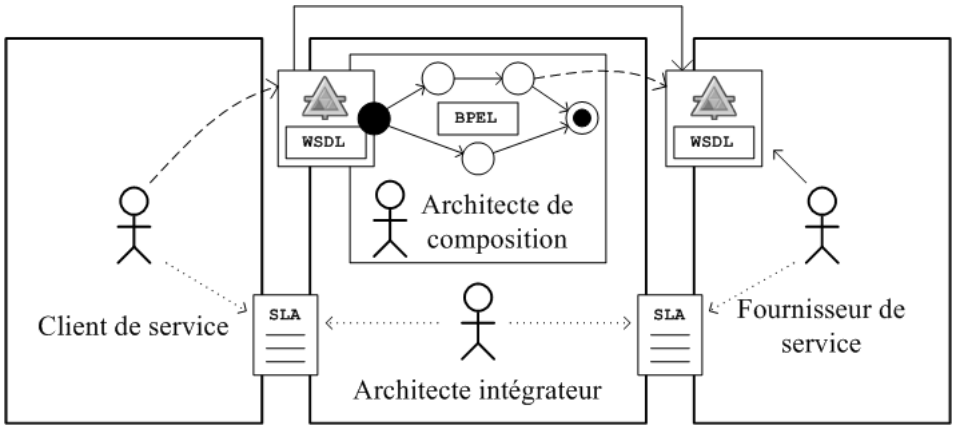

FIGURE 4.1: Rôles intervenant autour des compositions de service

langage avec comme objectif de limiter l'intrusivité entre les plateformes d'exécution des diverses préoccupations (section 4.2.4).

4.2.1 Vers une meilleure séparation des préoccupations

Etude des rôles Présenté à la section 2.2.1, le principe de séparation des préoccupations est un concept fondamental du génie logiciel consistant à décomposer un système en sous parties (chacune de ces sous parties traitant d'une préoccupation particulière) afin d'en simplifier sa maîtrise. Il est envisageable d'avoir un acteur spécifique en charge de chacune des préoccupations du système considéré. L'acteur joue un rôle dans la gestion du système et il en possède une vue propre à ce rôle. Dans notre contexte, le système est la composition de services et nous avons identifié quatre rôles, comme esquissé sur la figure 4.1.

Bien que davantage de rôles puissent être mis en évidence, en fonction du niveau de raffinement désiré, nous avons choisi ces quatre rôles car ils capturent de façon pertinente les préoccupations qui concernent la gestion de la composition et de la QdS. Parmi ces quatre rôles, le Client et le Fournisseur de service n'ont pas accès à la logique de composition (boîte noire). L'Architecte de composition ainsi que l'Architecte intégrateur s'occupent spécifiquement de la composition.

- Le **client du service** est le consommateur du service issu de la composition. Il s'occupe de la logique permettant d'invoquer le service sur le réseau et est le bénéficiaire de la valeur ajoutée fonctionnelle qu'offre la composition de services de base. De plus, il doit veiller à ce que les traitements liés à la QdS soient bien mis en oeuvre lors de ses échanges avec le service. Il peut y avoir plusieurs clients pour une même composition.
- Le **fournisseur de service** déploie un service Web sur le réseau et gère son accès. Il a pour tâche de mettre en oeuvre les traitements de QdS (dont la logique est indépendante de celle du service implémenté). C'est notamment le cas pour la mise en place de préoccupations de QdS liées à la sécurité ou la proposition d'offres de QdS. A l'instar du client, il peut y avoir plusieurs fournisseurs de service par composition.
- L'**architecte de composition** est responsable de la partie fonctionnelle du dé-

veloppement de la composition. Il apporte des informations concernant les interfaces fonctionnelles des partenaires de la composition, les messages échangés et leur logique d'enchainement grâce au langage BPEL4WS. La composition permet de créer un nouveau service accessible sur le réseau et dont l'interface est décrite en WSDL. Cependant, au stade de l'implémentation, les partenaires de la composition ne sont pas encore connectés à des services Web existants.
– L'**architecte intégrateur** est le rôle qui nous intéresse pour notre étude. Il a pour tâche d'effectuer le lien entre la composition et les Fournisseurs de Service. Pour chaque partenaire de la composition, il est possible de connecter plusieurs services Web fonctionnellement équivalents, chacun étant géré par un fournisseur de Service particulier. L'architecte intégrateur a pour rôle de négocier et de gérer les contrats de service avec les fournisseurs de service afin de faire correspondre un service Web approprié pour chaque partenaire de la composition. Par ailleurs, il doit veiller à ce que les traitements liés à la QdS soient bien mis en oeuvre, aussi bien lors de chaque interaction avec les partenaires que lors de l'exécution du *workflow*. Pour cela, l'Architecte Intégrateur doit avoir une vue « boîte grise » de la composition : il connait le *workflow* d'activités et les messages échangés mais il ne peut pas accéder à la logique des services Web. Par ailleurs, l'architecte intégrateur » fait également le lien avec le client de la composition afin de gérer le contrat de service ainsi que les traitements liés à la QdS.

Parmi les quatre rôles, le rôle de l'architecte intégrateur est de spécifier la gestion de la QdS liée à la composition de services. Ce rôle possède une vue boite grise de la composition : il peut voir les activités de la composition ainsi que la QdS des services, mais n'a pas accès au contenu des services. Il a pour tâche de sélectionner les fournisseurs de service appropriés et de communiquer des offres de QdS au client du service composite. Il doit veiller au respect de ces offres et donc pour cela, il doit observer la QdS de la composition, et doit pouvoir mettre en oeuvre des mécanismes liés à la QdS (tels que la sécurité) sur des parties du *workflow*. Enfin, le fait d'avoir une vue boite grise lui permet de gérer finement la QdS de sa composition : il peut ainsi spécifier des exigences plus fines sur certaines parties de sa composition, effectuer une plus-value (par exemple pour dégager un bénéfice), ou bien appliquer des stratégies de replanification à des niveaux intermédiaires (tels que des activités composites).

4.2.2 Etude du domaine de la QdS

Comme nous l'avons déjà constaté dans le chapitre 3, la prise en charge de la QdS dans les compositions comporte un ensemble de traitements divers et variés. Ce sont ces traitements et leurs informations associées qui caractérisent le rôle de l'architecte intégrateur et délimitent un domaine d'étude. L'analyse de ce domaine d'étude est déterminante pour la contribution de la thèse. Pour le faire apparaître, nous analysons successivement les informations puis les traitements pertinents pour la gestion de la QdS.

Informations liées à la gestion de la QdS La figure 4.2 expose les relations entre les différentes informations qui capturent ou entourent la gestion de la QdS. Ces éléments sont regroupés en trois sous domaines.

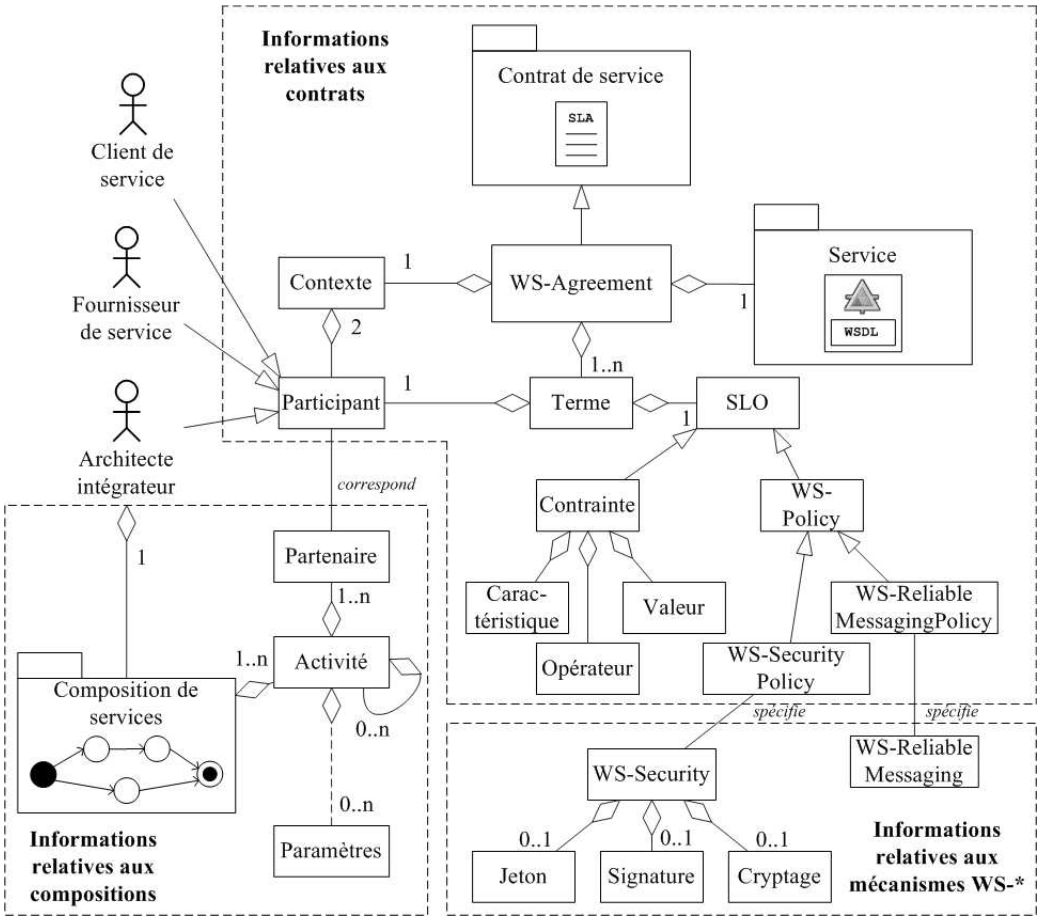

FIGURE 4.2: Informations relatives à la gestion de la QdS dans les compositions

1. Informations liées au contrat de service :
   Il s'agit d'un élément fondamental de la gestion de la QdS. Présenté à la section 2.3.2, le contrat de service (ou SLA) spécifie la QdS attendue entre le client et le fournisseur pour le service délivré. Bien qu'il n'existe pas de standard clairement établi pour la spécification des SLA, nous nous sommes particulièrement intéressés aux travaux autour de WS-Agreement [ACD+07] qui offre un modèle relativement générique. Ce formalisme possède une structure dont les principaux constituants sont : un contexte faisant référence à deux parties (initialisateur de l'accord, pour nous le fournisseur, et celui qui y répond, pour nous le client) et des termes permettant de spécifier des offres (portées par chaque partie) sur la QdS du service. Dans le cas de WS-Agreement, les termes peuvent être soit de plusieurs types (nous nous intéressons en particulier aux termes permettant de définir des SLO), soit des assertions décrites dans le standard WS-Policy. Les SLO peuvent s'écrire sous la forme de contraintes sur des propriétés de performance (temps de réponse inférieure à une valeur, débit supérieur à un seuil), tandis que les assertions WS-Policy permettent de spécifier des mécanismes à mettre en oeuvre via la spécialisation en WS-SecurityPolicy, WS-ReliableMessagingPolicy, etc.

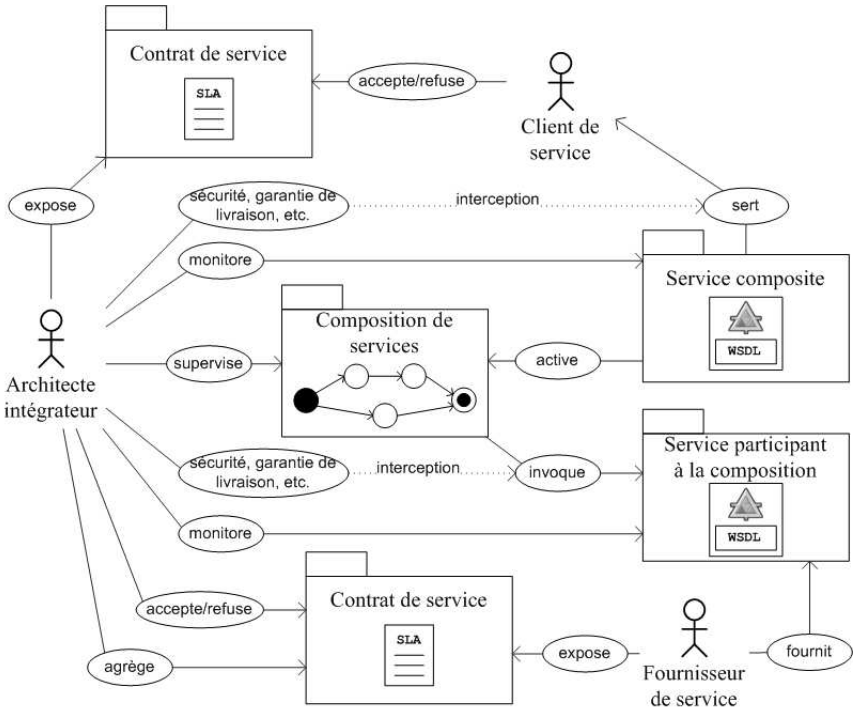

FIGURE 4.3: Traitements relatifs à la gestion de la QdS dans les compositions

2. Informations liées aux mécanismes de QdS
   Ces mécanismes sont capturés dans les standards WS-*. Par exemple, pour les mécanismes de sécurité, il existe différents éléments tels que les jetons de sécurité (UsernameToken ou bien certificats X509, etc.), la signature des messages ou bien encore le cryptage. Ces informations sont réifiées dans la norme WS-Security. La spécification WS-SecurityPolicy qui peut être utilisée à l'intérieur d'un terme de SLA peut servir à spécifier quels mécanismes WS-Security sont attendus par les parties signataires du contrat. La partie concernée peut alors mettre en oeuvre la logique de sécurité correspondante à l'exécution du service.
3. Informations liées à la composition
   L'architecte intégrateur a accès à la composition de services. Décrite par le modèle BPEL [Dub02], la composition de service est en particulier constituée d'activités (simples ou composites) pouvant potentiellement faire intervenir de multiples partenaires. Par ailleurs, ces activités peuvent faire intervenir des paramètres spécifiques tels que le nombre moyen d'itération pour une boucle.

Traitements liés à la gestion de la QdS La figure 4.3 capture les traitements liés à la gestion de la QdS dans les compositions de services. Il s'agit donc des traitements associés au rôle d'architecte intégrateur afin de gérer la QdS de la composition de services.

1. Gestion des accords :
   L'architecte intégrateur dialogue avec le client du service composite ainsi que les fournisseurs de service afin de négocier les contrats de service liés au composite et

aux services participant à la composition. Cette opération est fondamentale car c'est l'établissement de SLA avec les partenaires (client et fournisseurs) qui permet d'établir et de garantir la QdS de la composition. Par ailleurs, elle s'effectue aussi bien à l'étape statique qu'à l'étape dynamique. En effet, les contrats doivent être établis avant le déploiement de la composition, puis, à l'exécution, il se peut que des contrats soient violés ou se terminent, et il devient alors nécessaire d'établir de nouveaux contrats. Le modèle de négociation choisi pour nos travaux est volontairement réduit par souci de simplification, à l'acceptation ou au refus d'un contrat entièrement composé d'offres des fournisseurs de service (ce modèle pouvant alors être complexifié par la suite). L'architecte intégrateur effectue le travail d'agrégation des contrats de ses fournisseurs afin de sélectionner les fournisseurs de service qui lui permettent de satisfaire les exigences de la composition et celles du contrat du composite. Par une gestion fine, il peut réaliser une plus-value ou bien encore adapter les fournisseurs de service selon ses besoins.

2. Observation de la QdS
   Pouvoir recueillir statiquement et dynamiquement des informations spécifiques à la QdS dans la composition est à la base de toute prise de décision. L'architecte intégrateur doit donc se voir offrir la capacité de faire des relevés ou d'accéder aux informations qui lui sont pertinentes. Tout d'abord, il doit pouvoir monitorer la QdS des échanges que la composition de services effectue avec ses partenaires, afin de vérifier que les offres des SLA sont bien respectées à l'exécution. Il doit également pouvoir accéder d'une part aux SLA, afin de lire leurs spécifications (permettant ainsi de mettre en place des mécanismes de QdS appropriés), et aux informations relatives à la composition de services (paramètres permettant d'identifier le choix dans les chemins d'exécution, exigences de QdS sur certaines parties du *workflow*).
3. Mise en oeuvre des mécanismes WS-* de QdS :
   Au delà du *monitoring* et de la gestion des contrats, il existe des traitements plus spécifiques dans la gestion de la QdS. Il s'agit de mécanismes de QdS tels que la sécurité ou la garantie de livraison, comme décrit dans la section 2.1.2, ou bien de mécanismes de gestion des exceptions ou encore de récupération sur erreur. Leur mise en oeuvre requiert une logique spécifique que l'architecte intégrateur doit spécifier. Se situant généralement à un niveau de granularité plus fin que le niveau composition de services, la logique de ces mécanismes requiert que les messages échangés avec les partenaires soient interceptés afin d'y appliquer les traitements correspondants. Ces mécanismes sont fondamentaux pour la gestion de la QdS et de la composition de services.

4.2.3 Approche langage dédié

A présent que nous avons défini un domaine d'étude correspondant au rôle d'architecte intégrateur, nous pouvons proposer un modèle permettant de capturer ce domaine. Comme l'ont déjà fait certains travaux de l'état de l'art, nous avons opté pour une approche langage dédié [MHS05]. Cette approche a pour but de proposer à l'architecte intégrateur un moyen d'expression adéquat afin de spécifier la gestion de la QdS dans des compositions de services. Le choix d'une approche langage dédié nous semble privilégié car elle autorise une définition concise du modèle de gestion de la QdS et elle

permet de disposer de descriptions exécutables. L'intérêt de l'utilisation d'un langage dédié dans notre contexte est multiple :

- Un langage dédié offre des **idiomes de programmation** qui se situent au niveau du domaine capturé. En limitant son expressivité à un domaine particulier (la gestion de la QdS), il permet ainsi de placer le focus sur les concepts de ce domaine. L'analyse de domaine effectuée à la section 4.2.1 a fait apparaître des traitements et des données. Ce sont ces éléments que doit capturer le langage dédié. En utilisant directement les concepts du domaine dans le langage, les spécifications peuvent être rédigées, maintenues et réutilisées de façon plus évidente.
- En encapsulant l'expertise du domaine, les langages dédiés offrent une **automatisation** de certains traitements. Ceci permet de réduire la charge de programmation de l'utilisateur qui a moins de code à écrire et qui doit donc développer une expertise moindre pour la mise en oeuvre des algorithmes. Dans le cas de la gestion de la QdS, il existe des traitements variés à mettre en oeuvre, tels que la gestion des SLA ou de la sécurité, à divers niveaux. L'utilisation d'un langage dédié permet de masquer la complexité de ces traitements en ne laissant apparaître à l'utilisateur que les informations réellement utiles pour la mise en oeuvre de la QdS. Comme mentionné dans la section 2.2.4, un problème inhérent à la conception des langages dédiés est l'ouverture du langage. Lors de l'étude des limitations des approches existantes, nous avons remarqué que certains langages étaient trop ouverts ce qui augmente la complexité du travail de spécification. D'autres, en revanche, sont confinés à un sous domaine limité de la QdS, ce qui diminue significativement leurs possibilités d'application. Grâce à la mise en évidence du domaine que nous cherchons à capturer, la conception de notre langage s'appuie sur des éléments structurants (entre autre, l'utilisation des SLA) et laisse place à des extensions potentielles (par exemple la prise en compte de nouvelles caractéristiques de QdS telles que la robustesse).
- Les langages dédiés permettent également d'offrir des **garanties** sur certaines propriétés du domaine capturé. Dans notre contexte, la gestion de la QdS implique de gérer la complexité de certains mécanismes et algorithmes. Grâce à un langage dédié, les traitements sont encapsulés dans le langage, ce qui réduit le risque d'erreur et qui permet ainsi d'obtenir du code plus fiable. Des vérifications statiques peuvent également être envisagées afin de vérifier certaines propriétés (telles que la prise en compte des exceptions pouvant être rejetées à l'exécution). Par ailleurs, le traitement du langage s'effectue à divers étapes, en parallèle de la gestion de la composition de services. L'utilisation d'un langage dédié pour la gestion de la QdS permet d'apporter des garanties (finitude des traitements, synchronisation) sur la bonne mise en oeuvre de la gestion de la QdS en parallèle des traitements requis pour la mise en oeuvre de la composition de services.

4.2.4 Mise en oeuvre non intrusive

Notre contribution repose sur le principe de la séparation des préoccupations. Si ce principe apporte de nombreux avantages, il soulève néanmoins le problème de la recomposition des préoccupations qui s'intéresse à l'agrégation des préoccupations ayant été préalablement isolées. Par exemple, dans le cas de la programmation par aspects, décrite à la section 2.2.3, la recomposition des préoccupations modularisées par les as-

pects s'effectue via l'opération de tissage : les informations contenues dans la coupe de l'aspect décrivent les endroits du code de base où il faut insérer l'*advice* de l'aspect. Dans notre contexte, il s'agit de recomposer la préoccupation de gestion de la QdS, modularisée dans notre langage dédié, avec la préoccupation de composition de services, modularisée dans le langage BPEL. Comme introduit dans 3.4, il existe différentes stratégies pour recomposer les préoccupations :

1. Tout d'abord, les spécifications peuvent être séparées puis mélangées statiquement par une opération de tissage afin qu'une seule logique soit traitée par la plateforme exécutant la préoccupation de base. Dans notre contexte, cette situation n'est pas envisageable puisque nous faisons le choix d'utiliser le langage BPEL pour spécifier la composition de services et que ce langage dédié est limité à un ensemble restreint d'activités. Il n'est tout simplement pas possible ni de transformer la logique de gestion de la QdS en code BPEL, ni de la tisser directement dans le BPEL.
2. Une seconde stratégie consiste à lier avec un couplage fort les plateformes d'exécution afin que celles-ci intègrent en dur le « cablage » implémentant la recomposition des préoccupations. Afin de pouvoir réutiliser les moteurs BPEL et que ces derniers puissent évoluer librement, nous souhaitons privilégier l'approche la moins intrusive avec les plateformes BPEL. Cette stratégie n'est donc pas envisageable puisque le cablage en dur permettant aux plateformes de synchroniser leurs traitements impose de modifier et donc d'altérer la plateforme BPEL.
3. Enfin une troisième stratégie consiste à établir un couplage lâche entre les plateformes d'exécution. Pour cela, des indirections sont insérées dans le code de la préoccupation principale [DLBS01]. Ces indirections ont pour objectif d'appeler les plateformes en charge d'exécuter les autres préoccupations, lors de l'exécution. C'est cette stratégie que nous avons privilégiée car elle permet une mise en oeuvre de la gestion de la QdS de manière non intrusive avec la plateforme BPEL (*i.e* les plateformes ne sont pas mélangées). Ceci permet aux plateformes d'évoluer indépendamment l'une de l'autre. A l'exécution, la gestion de la QdS doit être synchronisée avec la composition de services. Pour cela, des indirections doivent donc être insérées dans le document BPEL, par transformation de programme, avant l'étape de déploiement. Ces indirections ont pour rôle d'appeler la plateforme en charge de notre langage dédié, lors de l'exécution de la composition. Cette stratégie permet de faire interagir les deux plateformes et donc les deux logiques. Cependant, la logique de gestion de la QdS doit également pouvoir interagir avec les messages échangés entre la composition et les partenaires. Etant donné que ces messages sont transmis sur le réseau, un mécanisme d'interception de type *proxy* permet à la plateforme de gestion de la QdS d'intervenir sur ces messages.

Globalement, la stratégie de mise en oeuvre de notre langage cherche à privilégier une ***approche non intrusive en regard de la plateforme BPEL*** afin de rester neutre vis-à-vis de celle-ci et qu'elle puisse évoluer indépendamment de nos préoccupations. Pour cela, la stratégie consiste d'une part à ***introduire des indirections*** dans le document BPEL, avant l'étape de déploiement, afin que la gestion de la QdS puisse interagir avec la composition à l'exécution, et d'autre part à ***intercepter*** les messages

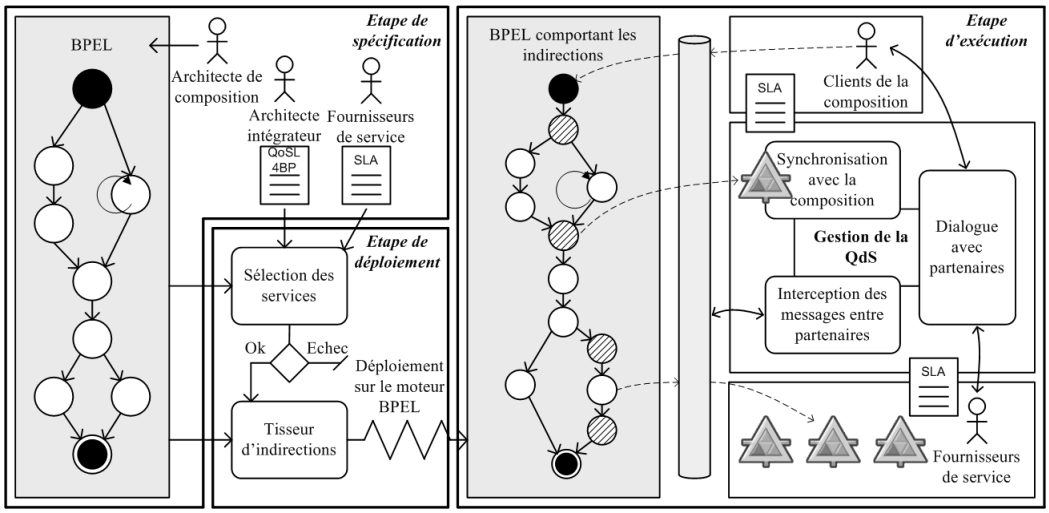

FIGURE 4.4: Processus global de mise en oeuvre de notre approche

échangés entre composition et partenaires afin que la gestion de la QdS puisse intervenir sur ces messages.

4.3 Orientation générale

4.3.1 Vue globale

A présent que nous avons présenté les points saillants du positionnement de nos travaux, nous donnons une vue globale de la mise en oeuvre de notre approche [BRL07]. Cette vue exhibe l'ensemble des points qui ont été exposés ainsi que leurs interactions. La mise en oeuvre de notre approche repose sur la mise en oeuvre du langage BPEL constituée d'une phase de spécification statique, d'une phase de déploiement sur moteur BPEL et d'une phase d'exécution. L'introduction de la préoccupation de gestion de QdS apporte des spécifications additionnelles à prendre en compte dans la mise en oeuvre des compositions BPEL. La figure 4.4 exhibe une vue générale du processus de mise en oeuvre de notre approche dans laquelle il est possible de distinguer les trois étapes originales du BPEL comportant des traitements additionnels.

Etape de spécification L'architecte de composition crée le processus métier qui capture les activités de la composition de services. A ce stade, il ne spécifie pas les partenaires de la composition et se base sur des interfaces **WSDL** génériques pour la définition des messages et des opérations dans le document **BPEL**. Cependant, les fournisseurs de services pouvant être utilisés pour implémenter les partenaires de la composition sont précisés à cette étape. Ces derniers fournissent des propositions de **SLA** pour les services qu'ils offrent. Enfin, l'architecte intégrateur rédige, à l'aide de notre langage dédié (**QoSL4BP**), les spécifications liées à la gestion de la QdS de la composition BPEL produite. Ces spécifications seront mises en oeuvre lors de l'étape de déploiement ainsi que lors de l'étape d'exécution. Le langage dédié est décrit dans le chapitre 5.

Etape de déploiement Avant que le déploiement du document de composition sur le moteur BPEL ne soit effectué, une première exécution du langage dédié à la gestion de QdS intervient afin de réaliser deux traitements. Tout d'abord, les partenaires implémentant les services de la composition sont sélectionnés selon les spécifications de l'architecte intégrateur, les propositions de SLA des fournisseurs et la structure de la composition produite par l'architecte de composition (section 6.2). A l'issue de cette étape, il se peut que certains partenaires n'aient pas trouvé au moins un fournisseur de service permettant de satisfaire aux exigences de l'architecte intégrateur et alors le déploiement échoue. Si chaque partenaire de la composition peut être implémenté par au moins un fournisseur de service avec des offres de QdS satisfaisantes, alors la **planification** des services aboutie. Le deuxième traitement de l'étape de déploiement consiste à transformer la composition BPEL originale afin d'y insérer des **indirections** (section 6.3). Il s'agit des indirections qui appeleront la plateforme en charge de la gestion de QdS à l'exécution de la composition afin de réaliser la synchronisation entre les préoccupations (composition et gestion de QdS).

Etape d'exécution Les clients de la composition invoquent la composition de services. Certains messages échangés entre les partenaires de la composition (clients et fournisseurs de service) et la composition elle-même sont interceptés par la plateforme en charge d'exécuter les spécifications de gestion de QdS. L'**interception** des messages échangés permet de mettre en oeuvre certains traitements de QdS (*monitoring* du service, sécurité, etc.). Lors de l'exécution des activités de la composition par le moteur BPEL, la plateforme de gestion de QdS effectue la **synchronisation** de ses traitements avec la composition grâce aux appels effectués par les indirections insérées dans le document de composition (section 6.4). Ces appels transmettent des informations liées à l'exécution de la composition que la plateforme de gestion de QdS utilise afin d'effectuer les traitements de QdS. Durant l'exécution, la QdS des services ou les exigences de QdS sont susceptibles de varier et il peut devenir souhaitable que les fournisseurs implémentant les partenaires soient remplacés. La plateforme de gestion de QdS a également pour rôle de dialoguer avec les potentiels fournisseurs de service afin de convenir d'un offre de service différente. Les mécanismes liés à la QdS tels que la sécurité sont également mis en oeuvre à cette étape (section 6.5).

4.3.2 Scenario fil conducteur

A présent que nous avons esquissé la manière dont se déroule le processus de mise en oeuvre de notre approche, nous présentons un scénario fil conducteur que nous avons conçu. Il fait intervenir des exigences de QdS que pourrait soumettre un opérateur des télécommunications souhaitant mettre à disposition de ses clients un service pour téléphone mobile. Il s'agit d'un cas commercial dans lequel l'opérateur des télécommunications cherche à assurer la fiabilité et la rentabilité de son service. Ce scénario a notamment été réutilisé dans le cadre du projet RNTL FAROS [PCW07]. Il est décliné dans les chapitres 5 et 6 afin d'exhiber sur un exemple concret de composition de services les différents concepts et mécanismes introduits.

Scenario Illustré dans la figure 4.5, le scenario « Urban Trip Planner » (UTP) fait apparaître une composition de services. Le service que délivre cette composition per-

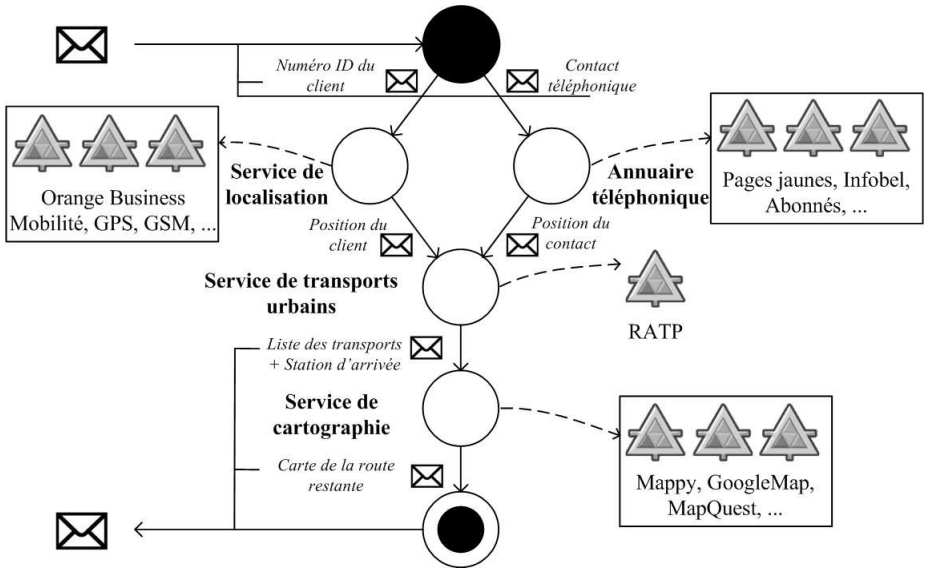

FIGURE 4.5: Scenario Urban Trip Planner

met à un utilisateur de mobile en environnement urbain de connaître les transports à emprunter ainsi que la route restante à parcourir afin de se rendre chez un des contacts de son répertoire. La composition UTP est composée de quatre services et son exécution se déroule de la façon suivante : lorsque la composition reçoit un appel d'un client, elle récupère comme informations le numération d'identification du client ainsi que le numéro du contact téléphonique. Ces informations sont respectivement envoyées en parallèle d'une part à une service de localisation qui retourne la position actuelle du client et d'autre part à un service d'annuaire téléphonique qui retourne la position du contact. Ces deux informations sont ensuite agrégées pour être transmises à un service de transports urbains afin que celui-ci retourne d'une part les transports à emprunter, et d'autre part la position de la station d'arrivée. Cette position ainsi que celle du contact sont alors ensuite transmises à un service de cartographie afin de synthétiser la carte de la route restante jusqu'à destination. Enfin, cette carte ainsi que la liste des transports sont agrégées afin d'être remises au client.

Composition de services Le listing 4.1 exhibe le code BPEL de cette composition. Ce code expose trois sections : les partenaires, les variables et les activités de la composition. Les activités décrivent l'enchainement des tâches implémentant le *workflow*. Les activités sont contenues dans une séquence « UTPComposition » qui spécifie successivement la réception d'un appel au service UTP (activité « ReceiveUTPRequest »), la parallélisation des deux invocations (activités « InvokeLocalization » et « InvokePhoneBook ») avec la création de leurs messages de requête respectifs (activités de type *assign*), puis les deux invocations restantes (activités « InvokeTransportation » et « InvokeGrapher ») et enfin l'activité de réponse au client (activité « ReplyToCustomer »).

```
<process name="utpserviceProcess" ..>
2   <partnerLinks>..</partnerLinks>

    <variables>..</variables>

    <sequence name="UTPComposition">
```

```
 7      <receive name="ReceiveUTPRequest" variable="utpServiceRequest" ../>

        <flow name="LocationScope">
          <sequence>
            <assign name="CreateLocationRequest">
12            <copy><from part="clientID" variable="utpServiceRequest"/>
                <to part="ID" variable="locationRequest"/></copy>
            </assign>
          <invoke name="InvokeLocation" inputVariable="locationRequest"../>
          </sequence>
17        <sequence>
              <assign name="CreatePhoneBookRequest">
              <copy><from part="contactID" variable="utpServiceRequest"/>
                <to part="ID" variable="phonebookRequest"/></copy>
            </assign>
22          <invoke name="InvokePhoneBook" inputVariable="phonebookRequest"../>
          </sequence>
        </flow>

        <assign name="CreateTransportationRequest">
27        <copy><from part="location" variable="locationResponse"/>
            <to part="start" variable="transportationRequest"/></copy>
          <copy><from part="address" variable="phonebookResponse"/>
            <to part="arrival" variable="transportationRequest"/></copy>
        </assign>
32      <invoke name="InvokeTransportation" inputVariable="transportationRequest"../>

        <assign name="CreateGrapherRequest">
          <copy><from part="arrivalStation" variable="transportationResponse"/>
            <to part="start" variable="grapherRequest"/></copy>
37        <copy><from part="address" variable="phonebookResponse"/>
            <to part="arrival" variable="grapherRequest"/></copy>
        </assign>
        <invoke name="InvokeGrapher" inputVariable="grapherRequest"../>

42      <assign name="CreateUTPServiceResponse">
          <copy><from part="transportation" variable="transportationResponse"/>
            <to part="transportation" variable="utpServiceResponse"/></copy>
          <copy><from part="map" variable="grapherResponse"/>
            <to part="map" variable="utpServiceResponse"/></copy>
47      </assign>
        <reply name="ReplyToCustomer" variable="utpServiceResponse"../>

      </sequence>
    </process>
```

Listing 4.1: Code BPEL de la composition UTP

Exigences Plusieurs fournisseurs de service, chacun étant accompagné d'un SLA, peuvent jouer le rôle de partenaires pour les services de la composition. Par ailleurs, le client de la composition UTP souhaite recevoir une proposition de SLA de la part du service UTP afin de pouvoir y accéder. L'architecte intégrateur souhaite également pouvoir spécifier plusieurs exigences relatives à l'expression de la QdS dans la composition UTP :

1. Tout d'abord, l'architecte intégrateur souhaite offrir au client de la composition UTP une QdS ayant pour caractéristiques un temps de réponse inférieur à 25 ms, un débit supérieur à 100 requêtes par minute et un coût de 35 centimes par appel. Il souhaite également proposer au client de mettre en place un mécanisme de garantie de livraison. Pour la sélection des services de sa composition, il veut s'assurer d'effectuer une plus-value d'au moins 3 centimes par appel. Par ailleurs, il exige une QdS plus stricte que celle qu'il offre aux clients afin de prévenir aux éventuelles défaillances des services qu'il faudra alors immédiatement tenter de corriger par replanification à l'exécution. Le cas échéant, l'architecte intégrateur souhaite être prévenu et rejeter les appels de son client.
2. Afin de limiter l'effort de recherche des services, l'architecte souhaite que l'acti-

vité composite « LocationScope » gère elle-même sa QdS. Pour cela, il souhaite garantir que cette activité peut effectuer ses traitements avec un temps de réponse inférieur à 13 ms, un débit supérieur à 110 requêtes par minute et un coût inférieur à 15 centimes par appel. Cette activité gérera elle-même la planification et la replanification de ses services. Par ailleurs, un mécanisme de cryptage devra être mis en place entre la composition et ces services afin de préserver la confidentialité des données échangées.

3. Le service de transports urbains se trouve être un service unique de la RATP qui ne possède pas de SLA. Par expérience, l'architecte intégrateur sait que la QdS de ce service a pour caractéristiques un temps de réponse inférieur à 5 ms, un débit supérieur à 200 requêtes par minute et un coût égal à 10 centimes par appel. Il souhait donc isoler ce service en plaçant lui-même des garanties de QdS sur l'activité « InvokeTransportation ». Si la QdS du service de la RATP ne respectait plus ces caractéristiques à l'exécution, alors le service UTP devrait échouer et l'architecte intégrateur devrait être notifié.

4. Il existe un grand nombre de services potentiels pour l'activité « InvokeGrapher » possédant des caractéristiques de QdS très variantes. Afin de limiter l'effort de recherche et de rationaliser le coût de ce service qui peut être élevé, l'architecte intégrateur souhaite rajouter une exigence de coût sur cette activité, précisant ainsi que le service ne doit pas faire payer plus de 5 centimes par appel. En cas de violation de cette contrainte, il souhaite une replanification du service et si ce n'est pas possible, alors le service UTP devrait échouer et l'architecte intégrateur devrait être notifié.

4.4 Conclusion

Dans ce chapitre qui fait le lien entre la présentation des travaux déjà existants et les contributions de la thèse, nous avons tout d'abord effectuer un bilan général des approches ayant déjà été étudiées. En parcourant les cinq critères d'évaluation, nous avons, pour chacun de ces travaux, analysé les limitations et fait émerger des pistes potentielles pour les surmonter.

Globalement, les approches ne réutilisent pas assez les travaux effectués autour des SLA. Par ailleurs, beaucoup de travaux insistent sur le principe de séparation des préoccupations, mais peu effectuent véritablement une étude des préoccupations et des rôles autour des compositions de services. De ce fait, bien que les spécifications soient physiquement isolées, le domaine des préoccupations capturées n'est pas suffisamment mis en évidence. Par ailleurs, les plateformes de mise en oeuvre requièrent des modifications du moteur d'exécution des compositions dans la majorité des cas, ce qui limite la réutilisation des outils développés autour du langage BPEL. La couverture du domaine de QdS pris en compte par les travaux existants est relativement inégale et aucun travail ne propose véritablement d'adresser toutes les facettes de la QdS. En particulier, très peu de travaux prennent en charge les SLA. Du point de vue de la dynamicité, nous avons pu constater que peu d'approches prenaient en charge à la fois la phase statique et la phase dynamique. Généralement les outils proposés ne s'adressent véritablement qu'à une seule des étapes de mise en oeuvre. Enfin pour ce qui est de l'expressivité des

interfaces proposées aux utilisateurs afin de spécifier des traitements de QdS, celle-ci est souvent trop complexe ou limitée.

L'orientation de la thèse tire profit de ce bilan en réutilisant les éléments qui semblent pertinents pour résoudre la problématique de la thèse et en proposant des pistes d'amélioration lorsque des limitations existaient. En particulier, le principe de séparation des préoccupations, déjà utilisé dans certaines approches, a été réutilisé de manière plus approfondie dans la situation de la gestion des compositions de services. Cette analyse a permis d'aboutir à l'élaboration de quatre rôles avec en particulier le rôle de l'architecte intégrateur qui s'occupe de la gestion de la QdS dans les compositions de services. L'étude de ce rôle et de son domaine de préoccupation, la gestion de la QdS, a permis de mettre en évidence les informations et les traitements qu'il adresse. Nous avons ensuite proposé une approche langage dédié pour capturer ce domaine. Ce langage a pour objectif de mettre à la disposition de l'architecte intégrateur des idiomes de haut niveau encapsulant l'expertise liée à la gestion de la QdS et apportant des garanties. Afin de réutiliser les outils qui ont déjà été développés pour le langage BPEL et leur laisser la possibilité d'évoluer librement, nous avons privilégié une mise en oeuvre non intrusive avec les moteurs BPEL. Pour cela, notre langage dédié est exécuté par une plateforme spécifique. Cette dernière synchronise ses traitements avec ceux du moteur BPEL grâce à des appels réalisés depuis le document BPEL par des indirections insérés lors de l'étape de déploiement.

Chapitre 5

QoSL4BP : Un langage dédié pour la gestion de la Qualité de Service dans les compositions de services

Ce chapitre présente notre langage dédié pour la gestion de la QdS dans les compositions de services. Il s'agit d'un élément fondamental de notre contribution puisque ce langage encapsule le domaine de la gestion de la QdS. C'est à travers ce langage que l'architecte intégrateur peut spécifier ses besoins pour la gestion de la QdS des compositions de services. Nous présentons brièvement ce langage et les caractéristiques de sa conception dans la section 5.1. Nous abordons ensuite le modèle de données et les traitements du langage respectivement dans les sections 5.2 et 5.3, avant d'expliquer plus en détail comment se structurent les traitements dans les politiques à la section 5.4. Finalement, dans 5.5, nous présentons les différentes propriétés et garanties de ce langage avant de conclure.

Sommaire

5.1 Présentation générale du langage

Comme annoncé dans le chapitre 4, nous avons fait le choix d'un langage dédié dans le but de fournir au rôle d'architecte intégrateur un outil aisé et sûr pour la gestion de la QdS dans les compositions de service. Ce langage, nommé « Quality of Service Language for Business Processes » (QoSL4BP), a été élaboré sur la base de l'étude de domaine réalisée à la section 4.2.2. Il vise ainsi à offrir d'une part un modèle de données réifiant les informations qui sont y présentées et d'autre part des instructions permettant de mettre en oeuvre les traitements décrits.

Pour concevoir ce langage, nous avons choisi de privilégier certaines caractéristiques en rapport avec les critères d'évaluation mis en évidence dans la section 3.1.

Langage déclaratif et de haut niveau Dans l'état de l'art, nous avons remarqué que les langages offrant une importante liberté dans l'expression des spécifications sont généralement plus complexes à mettre en oeuvre. En effet, ils requièrent de la part des utilisateurs un certain niveau d'expertise dans le contrôle des algorithmes qu'ils souhaitent mettre en oeuvre, mais en contrepartie ils offrent une plus grande permissivité dans les traitements qu'il est possible de spécifier. En particulier, les langages peuvent être catégorisés selon certains critères (par exemple, langages déclaratifs et impératifs ou bien encore langages de haut niveau et de bas niveau). Alors qu'un langage impératif (tel que AO4BPEL, présenté à la section 3.2.1) s'appuie sur l'évolution de l'état du système lors du déroulement des instructions, un langage déclaratif privilégie plutôt l'expression de l'objectif à atteindre. Par ailleurs, un langage de haut niveau permet au programmeur de manipuler des concepts plus élaborés, s'abstrayant ainsi des détails inhérents à l'implémentation des algorithmes.

Dans le cas du langage QoSL4BP, l'expressivité est limitée à la gestion de la QdS dans les compositions de services. L'ensemble des traitements qu'il est possible d'effectuer dans ce domaine est relativement restreint (sélection des services, mise en place de *monitoring*, etc.). Par ailleurs, l'expression d'objectifs est une manière pertinente pour adresser la gestion de la QdS (par exemple, pour la spécification de contraintes). Pour ces raisons, il est préférable de privilégier la possibilité, pour l'architecte intégrateur, de spécifier aisément des objectifs de QdS ainsi que d'avoir un contrôle plus restreint, mais plus sûr, des traitements liés à la mise en oeuvre de la QdS. Ainsi, le langage QoSL4BP a été élaboré d'une part comme un langage déclaratif, afin de masquer la complexité et permettre la définition aisée d'objectifs, et d'autre part comme un langage de haut niveau, dans le but de fournir à l'architecte intégrateur des concepts de programmation du même niveau d'abstraction que le domaine encapsulé.

Prise en compte des étapes statique et dynamique Le traitement de compositions BPEL impose plusieurs étapes de traitement distinctes dont en particulier le déploiement et l'exécution de la composition. La gestion de la QdS suit également ces étapes. L'étude du critère de dynamicité 3.1 révèle que peu d'approches existantes considèrent à la fois le traitement statique et dynamique dans la gestion de la QdS. Par l'intermédiaire du langage QoSL4BP, nous souhaitons que l'architecte intégrateur puisse adresser ces deux étapes de manière adéquate. Pour répondre à cette problématique, les informations liées à ces deux étapes de traitement sont isolées dans des sections séparées. Cette approche hybride vise à offrir un meilleur contrôle de la QdS

pendant les phases statique et dynamique. Notamment, il devient possible d'effectuer des traitements de sélection de services et d'apporter des garanties sur la QdS de la composition lors son déploiement, puis, lors de l'exécution, de monitorer et de réajuster la QdS, tout en faisant intervenir d'autres mécanismes dynamiques de QdS.

Niveau de granularité variable Lors de l'étude du critère « couverture de domaine », nous avons pu constater d'une part que les plateformes n'adressaient pas un niveau de granularité entre le niveau global (*i.e* la composition dans son ensemble) et le niveau local (*i.e* une activité simple), et d'autre part que les approches ne permettaient généralement pas de mixer les niveaux de granularité lors de leurs traitements. C'est par exemple le cas pour les langages WSCoL et WSReL, présentés à la section 3.2.2, dont la cible des traitements ne peut dépasser la granularité d'une activité, ou bien inversement le cas pour la stratégie de réduction présentée à la section 3.3.2 qui ne s'applique que sur la globalité de la composition de services. Pour l'élaboration du langage QoSL4BP, nous avons souhaité donner à l'architecte intégrateur la capacité de spécifier des exigences et des traitements aux niveaux de granularité qu'il souhaite. Pour cela, QoSL4BP a été conçu de manière à permettre d'adresser tout type d'activités BPEL, simples et composites. Il est également possible de faire intervenir différents niveaux de granularité dans la gestion de la QdS d'une même composition de services. Par exemple, la replanification des services peut être envisagée sur une activité composite de la composition de services.

Regroupement par modules (politiques) Afin de faciliter la réutilisation et le regroupement cohérent de spécifications de gestion de QdS, nous avons introduit une notion de module représentant l'unité de base pour l'écriture de spécifications QoSL4BP. Au regard des choix déjà effectués, ces modules sont conçus comme l'agrégation de trois sections, comportant d'une part une section spécifiant la ou les activités ciblées par le module (quel que soit leur niveau de granularité), et d'autre part deux sections concernant les spécifications statiques et dynamiques. L'architecte intégrateur, qui possède une vue « boite grise » de la composition de services, regroupe ainsi dans un même module l'ensemble des spécifications de gestion de QdS qui concerne une même activité BPEL. Ainsi, c'est la notion d' « activité BPEL ciblée » (*SCOPE*) qui permet de faire le lien entre la composition et la gestion de QdS, tandis que les spécifications statiques (*INIT*) et dynamiques (*RULES*) permettent de modifier le comportement du système. Par ailleurs, du fait que le langage QoSL4BP est conçu comme un langage déclaratif et de haut niveau, les traitements sont assimilables à des objectifs ou des règles permettant de gouverner la QdS de l'activité ciblé. Le listing 5.1 exhibe la structure des politiques QoSL4BP.

```
POLICY "policy_name" = {
  SCOPE = { // Specification de la cible }
  INIT  = { // Specification des traitements statiques }
  RULES = { // Specification des traitements dynamiques }
}
```

Listing 5.1: Structure des politiques QoSL4BP

Comme présenté à la section 2.3.3, une politique est un moyen offert aux administrateurs pour définir et modifier l'organisation et le comportement d'un système. Elle se compose généralement de règles, dérivant d'une description d'une stratégie et spécifiant au système les actions à entreprendre pour répondre à une situation donnée. Les poli-

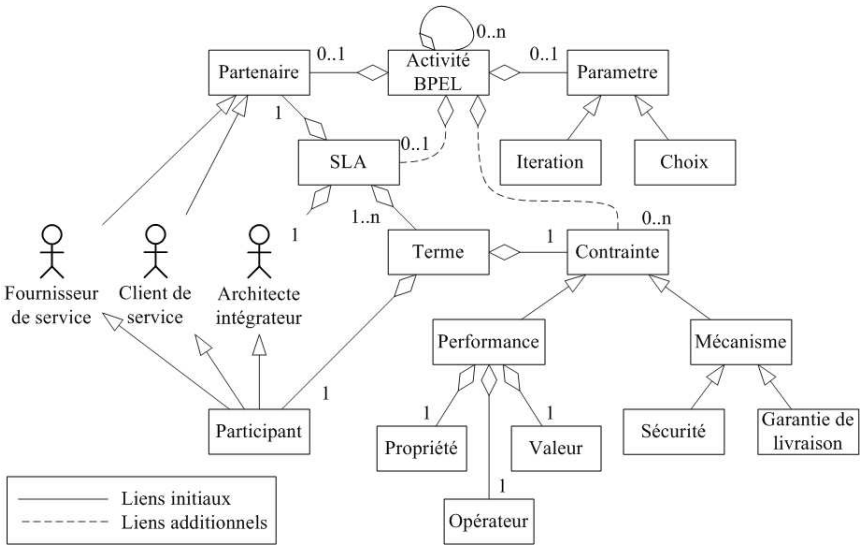

FIGURE 5.1: Données du langage QoSL4BP

tiques consistent généralement en des spécifications de haut niveau et n'intègrent pas les détails d'implémentation. La conception des modules QoSL4BP est très similaire à celle des politiques, notamment par l'utilisation d'une cible pour isoler un système ainsi que de traitements déclaratifs de haut niveau pour spécifier les comportements du système. Pour ces raisons, la dénomination de « politiques » est utilisée pour désigner les modules QoSL4BP.

5.2 Modèle de données

A présent que la structure générale des politiques a été introduite, ainsi que les choix de conception, nous présentons le modèle de données exploité par le langage QoSL4BP ainsi que les traitements qu'il permet de mettre en oeuvre. Le modèle de données exploite l'analyse de domaine effectuée dans la section 4.2.2 en la spécialisant et en établissant des liens plus directs entre les différents concepts du domaine. Nous étudions d'une part ces liens et d'autre part la manière dont les informations sont réifiées dans le langage QoSL4BP.

5.2.1 Vue globale

Les informations fondamentales pour le modèle de données du langage QoSL4BP sont exhibées dans la figure 5.1. Cette figure met en évidence les liens initiaux qui unissent les données, ainsi que des liens additionnels, introduits afin de réunir de façon adéquate le domaine de la gestion de la QdS et le domaine des compositions de services. En particulier, ces liens seront exploités lors des traitements de QdS.

Cette vue globale des données fait apparaître des éléments particulièrement structurant pour la modélisation des informations. Il s'agit tout d'abord de la donnée **Activité BPEL** qui sert de point de départ pour cibler les traitements de QdS. Une activité BPEL peut potentiellement être composée de plusieurs activités BPEL, et per-

process_ decl ::= PROCESS

invoke_ decl ::= INVOKE[*identification*]
flow_ decl ::= FLOW[*identification*]
sequence_ decl ::= SEQUENCE[*identification*]
while_ decl ::= WHILE[*identification*]
switch_ decl ::= SWITCH[*identification*]

identification ::= *string* | *integer* | *

FIGURE 5.2: Réification des activités BPEL en QoSL4BP

met d'accéder d'une part à certains paramètres propres à l'utilisation de la composition et d'autre part aux partenaires potentiels avec qui l'activité interagit. Un second élément structurant est la donnée **SLA** qui relie les fournisseurs de service, les clients de la composition et l'architecte intégrateur. Un premier lien additionnel relie activité BPEL et SLA du fait que les partenaires des activités *invoke* constituent également des participants pour les SLA. Enfin, La donnée **Terme** permet de faire le lien entre la donnée SLA et les contraintes décrivant la QdS (**Performance** et **Mécanisme**). Ici encore, un lien additionnel permet de matérialiser la possibilité d'établir des relations entre les activités BPEL et des contraintes de QdS.

5.2.2 Données Activités BPEL

Les activités BPEL sont fondamentales dans la conception du langage QoSL4BP car l'architecte intégrateur doit être capable de cibler les activités du *workflow* et de naviguer dans une activité composite.

Certaines approches existantes (telles que AO4BPEL présenté dans la section 3.2.1 ou bien WSCoL présenté dans la section 3.2.2) utilisent le formalisme XPATH qui permet de sélectionner des balises XML. Bien que très permissive, l'utilisation d'un formalisme tel que XPATH n'est pas sûr car il peut être utilisé pour spécifier des balises du BPEL qui ne sont pas des activités (telles que les liens) ou bien des activités qui n'ont pas d'intérêt pour la QdS.

Nous avons donc choisi de réifier les activités BPEL dans le langage QoSL4BP. Les activités utiles au langage sont décrites dans la figure 5.2. Il est possible d'identifier de trois manières différentes une activité dans le *workflow* : soit par l'utilisation de son nom (valeur « name » dans le document BPEL), soit en indiquant son index (position dans l'activité composite qui contient l'activité) dans le document BPEL, soit par l'utilisation de la *wildcard* « * » qui a pour sémantique « toutes les activités de ce type ». Par ailleurs, nous avons rajouté le mot clé « PROCESS » permettant de désigner la racine de la composition.

Afin de pouvoir aisément naviguer à travers la hiérarchie des activités et de pouvoir désigner plusieurs activités, nous avons développé un formalisme simple. Ce formalisme introduit deux opérateurs de composition : « OR » afin de sélectionner plusieurs activités et « IN » afin de filtrer des activités selon qu'elles sont inclues dans une activité composite. Ce formalisme est détaillé dans la figure 5.3. Il permet notamment de faire une distinction entre la sélection d'activités simples et composites de manière à garantir

activity_ decl ::= *process_ decl*
| (*basic_ decl*)
| (*composite_ decl*)
| (*basic_ decl* OR *composite_ decl*)
| (*composite_ decl* OR *basic_ decl*)

basic_ decl ::= *invoke_ decl*
| (*basic_ decl* OR *basic_ decl*)
| (*basic_ decl* IN *composite_ decl*)

composite_ decl ::= *flow_ decl* | *sequence_ decl* | *while_ decl* | *switch_ decl*
| (*composite_ decl* OR *composite_ decl*)
| (*composite_ decl* IN *composite_ decl*)

FIGURE 5.3: Formalisme de sélection des activités BPEL en QoSL4BP

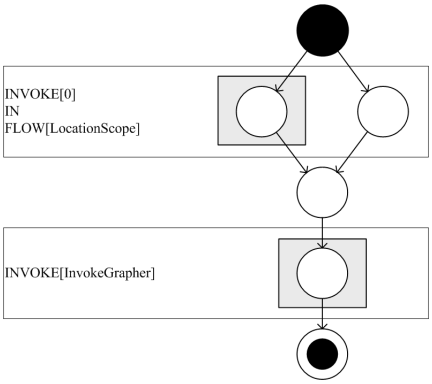

FIGURE 5.4: Sélection d'activités BPEL

la nature de la cible des traitements effectués par le langage QoSL4BP.

Ainsi, par exemple, la spécification du listing 5.2 permet d'isoler l'union de la première activité *invoke* comprise dans l'activité composite *flow* dont le nom est « LocationScope » et de l'activité *invoke* dont le nom est « InvokeGrapher ».

```
((INVOKE[0] IN FLOW["LocationScope"]) OR INVOKE["InvokeGrapher"])
```

Listing 5.2: Exemple de composition d'activités BPEL en QoSL4BP

La figure 5.4 fait apparaître le résultat de cette sélection sur le *workflow* de l'exemple UTP, présenté à la section 4.3.2.

5.2.3 Données SLA

Les données SLA permettent de spécifier la QdS entre la composition et ses partenaires. Un SLA se présente comme une collection de termes, chacun étant constitué d'un participant responsable d'une contrainte de QdS. Ces contraintes de QdS peuvent être de type performance ou bien de type mécanisme de QdS. Par ailleurs, une contrainte de QdS peut être perçue comme une offre ou une exigence selon que cette contrainte est

Caractéristique de QdS	Signification	Unité
RESPONSETIME	Temps de Réponse	millisecondes
THROUGHPUT	Débit	requêtes par minute
AVAILABILITY	Disponibilité	pourcentage de temps
COST	Coût d'un appel	centimes / appel

FIGURE 5.5: Propriétés de performance adressables par QoSL4BP

Domaine de QdS	Mécanisme	Signification
SECURITY	TOKEN, SIGNATURE, ENCRYPTION	Sécurité
RELIABILITY	EXACTLYONCE, INORDER	Garantie de livraison

FIGURE 5.6: Mécanismes WS-* adressables par QoSL4BP

vue comme une information de QdS garantie par le système à un partenaire extérieur ou bien au contraire comme une QdS souhaitée par le système et devant être garantie par un partenaire extérieur.

Avec le langage QoSL4BP, les spécifications rédigées sont toujours perçues du point de vue de l'architecte intégrateur. Ainsi, une contrainte de QdS spécifiée avec le client de la composition correspond à une offre que fait l'architecte intégrateur au client (et après accord, une exigence de la part du client). Inversement, une contrainte de QdS spécifiée avec un fournisseur de service correspond à une exigence de la part de l'architecte intégrateur soumise au fournisseur de service (après accord, il s'agit d'une offre de la part du fournisseur de service). Cette simplification permet de spécifier les acteurs de manière implicite dans le langage QoSL4BP et de ne se préoccuper que des contraintes de QdS.

Ainsi, pour spécifier une contrainte de QdS de type performance, le langage QoSL4BP met à disposition de l'architecte de composition quatre propriétés de QdS décrites dans la figure 5.5. Pouvant être considérées comme des propriétés rudimentaires de la QdS, ces propriétés sont déjà présentées dans les travaux de la section 3.3. Leur nombre et leur métrique ne sont cependant pas figés et le langage pourra exploiter d'autres propriétés de QdS. Par simplification, nous avons fait le choix de considérer des valeurs moyennes sur un intervalle temporel arbitraire pour l'expression des valeurs des propriétés de performance.

A l'instar des contraintes de QdS de type performance, les contraintes de type mécanisme WS-* de QdS utilisent les informations décrites dans la figure 5.6. Ces mécanismes constituent les principaux mécanismes de QdS décrits à la section 2.1.2 et également mis en oeuvre dans certains travaux de la section 3.2.

La figure 5.7 décrit la grammaire liées à ces informations de QdS.

5.3 Traitements

Dans cette section, nous présentons les primitives offertes par le langage QoSL4BP. Ces primitives sont issues de l'étude de domaine réalisée à la section 4.2.2. Elles sont répartis dans les trois domaines identifiés (gestion des contrats, observation de la QdS et gestion des mécanismes WS-*). Ces domaines sont ensuite eux-mêmes subdivisés

QoS_constraint	::=	[*QoS_performance*] | [*QoS_mechanism*]
QoS_performance	::=	*QoS_property operator integer*
QoS_property	::=	RESPONSETIME | THROUGHPUT
	|	AVAILABILITY | COST
operator	::=	< | > | = | >= | =<
QoS_mechanism	::=	*security_mechanism* | *reliability_mechanism*
security_mechanism	::=	SECURITY : *security_extension*
security_extension	::=	TOKEN | SIGNATURE | ENCRYPTION
reliability_mechanism	::=	RELIABILITY : *reliability_extension*
reliability_extension	::=	EXACTLYONCE | INORDER

FIGURE 5.7: Grammaire des contraintes de QdS en QoSL4BP

		Intégrateur-Client	Intégrateur-Activité	Intégrateur-Fournisseur
Gestion des accords	**Contraintes**	SET_OFFER	SET_OFFER SET_REQUIREMENT	SET_REQUIREMENT
	Sélection		PLANNING	PLANNING BIND BOUND
	Lecture	READ_CLIENT		READ_PROVIDER
Observation de la QdS	**Monitoring**	SENSOR	SENSOR MONITOR SET/GET_RATE SET/GET_LOOP	MONITOR
	Violation	VIOLATION_ACTIVITY	VIOLATION_ACTIVITY	VIOLATION_PROVIDER
Mécanismes de QdS	**WS-***	PERFORM_CLIENT	PERFORM_ACTIVITY	PERFORM_ACTIVITY
	Exception	THROW	CATCH (LOG)	

TABLE 5.1: Inventaire des primitives

en divers traitements plus spécifiques. La figure 5.1 fait l'inventaire des primitives du langage QoSL4BP selon ces traitements et selon la relation ciblée. Les différents types de relation que peuvent cibler les primitives sont entre l'architecte intégrateur et soit le client, soit les fournisseurs, soit les activités de la composition. Nous détaillons pour chacun de ces domaines les primitives offertes par le langage QoSL4BP ainsi que leur syntaxe.

5.3.1 Gestion des accords

Ce domaine est fondamental car il permet à l'architecte intégrateur de spécifier des offres et des exigences de QdS sur certaines parties de la composition de services, puis de faire rencontrer ces offres et ces exigences et enfin, après réalisation d'un accord, d'en lire le résultat.

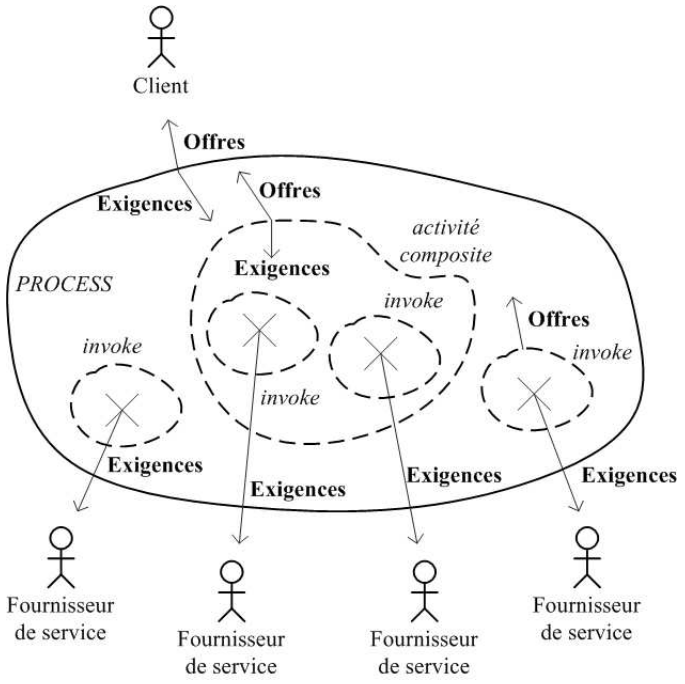

FIGURE 5.8: Placement d'offres et d'exigences de QdS sur une composition de services

Mise en place des contraintes La mise en place des contraintes est un traitement permettant de placer des offres et des exigences de QdS sur certaines parties de la composition. Lorsqu'une offre de QdS est positionnée sur une partie de la composition, alors tous les éléments (activités et acteurs) situés à l'extérieur de cette partie considèrent la QdS de cette partie comme celle garantie par l'offre (sans se préoccuper de la QdS de ses sous-parties). Un cas particulier est le cas où des offres sont placées sur la composition entière. Dans ce cas, les offres de QdS sont vues de l'extérieur par le client qui considère ces offres comme une proposition de SLA. Inversement, lorsque des exigences de QdS sont placées sur une partie de la composition, alors les éléments (activités et acteurs) situés à l'intérieur de cette partie sont soumis à ces exigences et doivent les satisfaire lors de la sélection des fournisseurs. A l'instar des offres, il est possible de placer des exigences sur la composition entière ainsi que sur des activités simples ou composites de la composition. Dans le cas où des exigences de QdS sont placées sur une activité *invoke*, alors ces exigences sont directement mise en confrontation avec les offres des SLA des fournisseurs potentiels de service pour cette activité. La figure 5.8 fait apparaître le placement d'offres et d'exigences de QdS sur une composition de services.

Les primitives « SET_OFFER » et « SET_REQUIREMENT » permettent à l'architecte intégrateur de placer respectivement des offres et des exigences de QdS sur les activités appartenant au *SCOPE* des politiques QoSL4BP. La signature de ces primitives est la suivante :

void **SET_OFFER(** [*QoS_constraint*]+ offres **)** ;
void **SET_REQUIREMENT(** [*QoS_constraint*]+ exigences **)** ;

Le listing 5.3 montre l'exemple d'une définition d'exigences de QdS spécifiant un

temps de réponse inférieur à 40 ms et un mécanisme de sécurité de type cryptage.

```
SET_REQUIREMENT( ([RESPONSE_TIME < 40], [SECURITY:ENCRYPTION]) ) ;
```

Listing 5.3: Exemple d'utilisation de la primitive « SET_REQUIREMENT »

Sélection des services La sélection des services est le traitement permettant de mettre en lien les diverses offres et exigences placées sur la composition avec les fournisseurs de service. Le langage QoSL4BP offre deux mécanismes afin de sélectionner les fournisseurs de service.

Tout d'abord, la primitive « PLANNING » permet d'effectuer la sélection d'un ou plusieurs services concrets. Pour effectuer cette sélection, la primitive « PLANNING » fait appel aux offres et aux exigences de QdS qui ont été placées sur les activités. Les fournisseurs doivent satisfaire aux exigences portées par les activités et les sous-activités portant des offres ne sont pas traitées.

La recherche des fournisseurs de services adéquats s'effectue en appliquant une stratégie spécifiée par l'architecte intégrateur. Celle-ci correspond à la stratégie de sélection des services potentiels successifs jusqu'à l'obtention d'une solution. Il existe de multiples stratégies de recherche, comme présenté dans l'état de l'art à la section 3.3. Nous présentons à la section 6.2.3 une stratégie de recherche par résolution de contraintes (CP [1]) ainsi qu'une stratégie de recherche exhaustive exploitant le *back tracking*. La primitive « PLANNING » est applicable aux activités de type *invoke* et aux activités composites contenant une ou plusieurs activités de type *invoke* (nous faisons l'hypothèse que toute activité composite du *workflow* possède au moins une activité de type *invoke*). La primitive « PLANNING » échoue dans le cas où aucun candidat potentiel n'est trouvé en rejetant une exception « PLANNING_EXCEPTION ». Par ailleurs, à chaque fois que cette primitive est rappelée à l'exécution, les services qui avaient été précédemment sélectionnés sont retirés de la liste des candidats potentiels. La signature de cette primitive est la suivante :

void **PLANNING(** *activity_decl* cible, *strategy_decl* strategie **)** ;
strategy_decl : := CP | BACK_TRACKING

Le listing 5.4 montre l'exemple d'une planification de services effectuée sur l'intégralité de la composition et utilisant comme stratégie de recherche une stratégie de *back tracking*.

```
PLANNING( PROCESS, BACK_TRACKING ) ;
```

Listing 5.4: Exemple d'utilisation de la primitive « PLANNING »

La primitive « BIND » permet de forcer l'utilisation d'un service pour devenir le partenaire d'une activité *invoke* en court-circuitant l'étape de vérification de la satisfaction des exigences de QdS. De ce fait, le service sélectionné peut ne pas avoir de SLA et, dans le cas où ce service vient avec un SLA, alors celui-ci est obligatoirement accepté. La signature de la primitive « BIND » est la suivante :

void **BIND(** *basic_decl* cible, *string* nom_du_service**)** ;

1. Constraint Programming

Le listing 5.5 montre un exemple d'utilisation de la primitive « BIND ». Cette instruction permet d'associer le service en lien avec l'activité « *invoke* grapher » avec le service « mappy ».

```
BIND( INVOKE["grapher"], "mappy" );
```

Listing 5.5: Exemple d'utilisation de la primitive « BIND »

La primitive « BOUND » a pour fonction de récupérer le service du partenaire lié à une activité *invoke*. La sémantique d'utilisation de « BOUND » est la suivante :

string **BOUND(** *basic_ decl* cible **)** ;

Lecture des accords Lorsqu'un accord émerge de la rencontre entre offres et exigences, un SLA est conclu. Dans le cas de la relation architecte intégrateur-client, le SLA satisfait aux exigences du client, tandis que dans le cas d'une relation architecte intégrateur-fournisseur de service, le SLA contient les offres du fournisseur de service. Si l'architecte intégrateur offre un mécanisme de QdS au client (tel que la sécurité), il se peut que celui-ci n'en réclame pas l'usage. Il apparaît donc intéressant pour l'architecte intégrateur de pouvoir connaître au mieux les contraintes spécifiées dans les SLAs conclus avec le client et les fournisseurs de service, notamment afin de tester si des mécanismes de QdS doivent être activés.

Pour cela, les primitives « READ_CLIENT » et « READ_PROVIDER » permettent de tester les contraintes de QdS présentes dans les SLAs signés respectivement par le client ou un fournisseur de service. Dans le cas où le service d'un fournisseur de service n'a pas de SLA associé, alors ces primitives retournent une exception « READ_EXCEPTION ». La signature de ces primitives est la suivante :

boolean **READ_CLIENT(** *QoS_ constraint* contrainte **)** ;
boolean **READ_PROVIDER(** *basic_ decl* cible, *QoS_ constraint* contrainte **)** ;

L'exemple du listing 5.6 montre un premier test validant la présence d'un terme garantissant que le client de la composition exige un mécanisme de garantie de livraison de type « INORDER », puis un second test validant qu'il existe un SLA avec le fournisseur du service lié à l'activité « *invoke* grapher » et que ce SLA spécifie une contrainte de QdS où la politique s'engage à fournir un disponibilité supérieur à 95%.

```
READ_CLIENT( [RELIABILITY : INORDER] ) ;
READ_PROVIDER( INVOKE["grapher"], [AVAILABILITY > 95] ) ;
```

Listing 5.6: Exemple d'utilisation des primitives « READ_CLIENT » et « READ_PROVIDER »

5.3.2 Observation de la QdS

La spécification d'offres et d'exigences, ainsi que l'établissement d'accords permettent de convenir d'une QdS mais il faut également pouvoir observer que ces spécifications sont respectées à l'exécution. Pour cela, des primitives permettent de mesurer la QdS et de vérifier le respect ou, le cas échéant, la violation des diverses contraintes de QdS.

Gestion du *monitoring* Le langage QoSL4BP fournit deux primitives pour le *monitoring* des contraintes de performance. « SENSOR » et « MONITOR » permettent respectivement de tester la QdS du signal en entrée d'une activité et la QdS liée aux traitements du signal par l'activité. Lorsque « SENSOR » est utilisé sur l'activité « PROCESS » alors elle permet d'évaluer le signal en entrée de la composition. En particulier, elle permet de révéler le débit d'entrée des messages du client. En ce qui concerne la primitive « MONITOR », celle-ci permet de vérifier que des contraintes de performance sont bien respectées. Les signatures de « SENSOR » et « MONITOR » correspondent à :

integer **SENSOR(** *activity_ decl* cible, *QdS_ property* propriete_de_QdS) ;
integer **MONITOR(** *activity_ decl* cible, *QdS_ property* propriete_de_QdS) ;

Le listing 5.7 exhibe des instructions effectuant les mesures de débit de requêtes que le client envoie à la composition ainsi que sur la valeur observée de la disponibilité du service lié à l'activité « *invoke* grapher ».

```
SENSOR( PROCESS, THROUGHPUT ) ;
MONITOR( INVOKE[grapher], AVAILABILITY ) ;
```

Listing 5.7: Exemple d'utilisation des primitives « SENSOR » et « MONITOR »

Par ailleurs, certaines activités BPEL possèdent un paramètre contenant des informations sur l'exécution du *workflow*. Ces paramètres sont utilisés dans le but d'effectuer la sélection de services de la manière pertinente. Dans le langage BPEL, les activités « While » et « Switch », décrivant respectivement des activités d'itération et d'embranchement conditionnel, contiennent de tels paramètres utiles au traitement de la QdS. Le langage QoSL4BP met quatre primitives à la disposition de l'architecte intégrateur afin que celui-ci puisse manipuler et connaître ces paramètres

Les primitives « SET_LOOP » et « GET_LOOP » permettent respectivement de saisir et de connaître le nombre d'itérations effectuées par une activité de type « While ». Cette information permet de déterminer combien de fois en moyenne le service sur lequel la boucle itère est appelé. En effet, selon le nombre d'itérations, les exigences de QdS souhaitées pour ce service varient. Les signatures de ces deux primitives sont les suivantes :

void **SET_LOOP(** *composite_ decl* activite_While, *integer* nombre_iterations) ;
integer **GET_LOOP(** *composite_ decl* activite_While) ;

Les primitives « SET_RATE » et « GET_RATE » permettent respectivement de saisir et de connaître, en pourcentage, les taux de sélection des diverses branches d'une activité « Switch ». Cette information permet de déterminer combien de fois en moyenne les services des diverses branches sont appelés. A l'instar des itérations de l'activité « While », les exigences de QdS souhaitées pour les services de ces branches varient avec les taux de sélection des branches. La syntaxe de ces deux primitives est la suivante :

void **SET_RATE(** *composite_ decl* activite_Switch, *string* condition,
integer taux) ;
integer **GET_RATE(** *composite_ decl* activite_Switch, *string* condition) ;

Vérification du respect des accords Les primitives « VIOLATION_ACTIVITY » et « VIOLATION_PROVIDER » ont pour rôle de vérifier si, respectivement, les offres placées sur une activité et les offres faites par le fournisseur ont été violées. Leur syntaxe est la suivante :

boolean **VIOLATION_ACTIVITY(** *activity_decl*) ;
boolean **VIOLATION_PROVIDER(** *activity_decl*) ;

Dans le cas où l'activité ciblée par la primitive « VIOLATION_ACTIVITY » se trouve être l'activité « PROCESS », alors les offres sont celles effectuées au client. De ce fait, la violation de ces offres équivaut à la violation du SLA passé avec le client.

5.3.3 Gestion des mécanismes liés à la QdS

Nous avons vu que, parmi les contraintes de QdS, peuvent se trouver des mécanismes liés à la QdS. Par ailleurs, il peut également se produire des exceptions liées à la QdS ou bien l'architecte intégrateur peut vouloir avoir des traces des exceptions et les rejeter vers le client lorsque la QdS n'est plus maintenue. Pour prendre en compte ces mécanismes, le langage QoSL4BP fournit quatre primitives.

Mécanisme de WS-* La primitive « PERFORM_CLIENT » ainsi que la primitive « PERFORM_ACTIVITY » permettent de mettre en oeuvre les mécanismes de QdS décrits dans les contraintes de type mécanisme de QdS. Ces mécanismes, tels que la sécurité ou la garantie de livraison, se produisent entre le client et la composition ou bien entre la composition et les services. Par choix dans nos travaux, nous limitons l'application des mécanismes de QdS à un seul sens : lorsqu'un mécanisme est appliqué à la relation client-composition, alors le client est l'initiateur du mécanisme (par exemple, pour un mécanisme de jeton, c'est le client qui envoie le jeton et la composition qui l'intercepte), tandis que pour une relation composition-fournisseur de service, l'appel effectué par la composition est l'initiateur du mécanisme de QdS. La primitive « PERFORM_CLIENT » a pour objectif de fixer les mécanismes entre la composition et le client. La primitive « PERFORM_ACTIVITY » permet de fixer les mécanismes entre la composition et un ou plusieurs fournisseurs de service. La signature de ces primitives sont :

void **PERFORM_CLIENT(** *QoS_mechanism* mecanisme
(, [*string*+]) ? parametres) ;
void **PERFORM_ACTIVITY(** *activity_decl* cible,*QoS_mechanism* mecanism
(, [*string*+]) ? parametres) ;

Des paramètres peuvent être requis pour la mise en oeuvre des mécanismes. Le listing 5.8 donne trois exemples d'utilisation des primitives « PERFORM_CLIENT » et « PERFORM_ACTIVITY » faisant apparaître l'utilisation de paramètres qu'offre le langage QoSL4BP pour divers mécanismes de QdS. A la ligne 1, l'instruction précise qu'un mécanisme de jeton de sécurité est mis en place avec le client. Le jeton reçu doit être composé d'un nom d'utilisateur égal à « usrname » ainsi que du mot de passe « mytoken ». A la ligne 2, l'instruction spécifie qu'un mécanisme de garantie de livraison de type « ExactlyOnce » est mis en oeuvre lors de l'envoi d'un message au service « Grapher ». Enfin, l'instruction à la ligne 3 indique qu'un mécanisme de

cryptage de message est utilisé pour appeler les divers services appartenant à l'activité « LocationScope ». L'algorithme 3-DES est utilisé avec pour clé la valeur « key123 ».

```
PERFORM_CLIENT( [SECURITY : TOKEN], ("usrname", "mytoken") ) ;
PERFORM_ACTIVITY( INVOKE["Grapher"], (RELIABILITY : EXACTLYONCE)) ;
PERFORM_ACTIVITY( FLOW["LocationScope"], (SECURITY : ENCRYPTION),
                  ("3DES", "key123") ) ;
```

Listing 5.8: Exemple d'utilisation des primitives « PERFORM_CLIENT » et « PERFORM_ACTIVITY »

Dans le cas où une erreur survient (mauvais jeton, decryptage impossible, mécanisme de garantie de livraison qui échoue), une exception est retournée par les primitives « PERFORM_CLIENT » et « PERFORM_ACTIVITY ». Les exceptions peuvent être de type « SECURITY EXCEPTION » dans le cas d'une exception liée à la sécurité ou bien « RELIABILITY EXCEPTION » dans le cas d'une garantie de livraison.

Gestion des exceptions Les primitives « CATCH » et « THROW » ont respectivement pour rôle de réceptionner une exception commise dans une politique et d'envoyer une exception de QdS au client. Nous avons vu que les traitements effectués par certaines primitives peuvent renvoyer des exceptions. Soit ces exceptions sont rattrapées par la primitive « CATCH », soient elles sont renvoyées dans la composition de services et donc au client. La primitive « THROW » permet également d'envoyer une exception au client. Les exceptions envoyées sont dans un type « QoSException » contenant un message. Leur signature est la suivante :

boolean **CATCH(** *string* message **)** ;
void **THROW(** *string* message **)** ;

La primitive « LOG » permet d'effectuer une action de *logging* dans un fichier. Ce mécanisme permet à l'architecte intégrateur d'avoir une trace de l'exécution de la gestion de la QdS. La signature de cette primitive est :

void **LOG(** *string* **)** ;

5.4 Modèle des politiques QoSL4BP

5.4.1 Structure

Comme introduit à la section 5.1, la conception du langage QoSL4BP repose sur le concept de « politique ». En particulier, une politique QoSL4BP est constituée d'une cible et de spécifications statiques et dynamiques. La structure des politiques fait explicitement apparaître ces éléments dans sa structure, comme exhibée dans le listing 5.9.

```
POLICY "policy_name" = {
  SCOPE = { // Specification de la cible }
  INIT  = { // Specification des traitements statiques }
  RULES = { // Specification des traitements dynamiques }
}
```

Listing 5.9: Structure des politiques QoSL4BP

Dans cette structure, la section *SCOPE* désigne la cible de la politique, tandis que les section *INIT* et *RULES* regroupent les traitements de gestion de QdS mis

QOSL4BP_POLICY ::= POLICY *string* = { *SCOPE_SECTION QOS_SECTION* }
QOS_SECTION ::= *INIT_SECTION*
| *RULES_SECTION*
| *INIT_SECTION RULES_SECTION*

FIGURE 5.9: Grammaire des politiques QoSL4BP

en oeuvre respectivement statiquement et à l'exécution de la composition. Ces trois sections ont chacune leurs spécificités dans l'usage des données et des traitements décrits respectivement dans les sections 5.2 et 5.3.

Notons qu'une politique peut ne pas contenir de section *INIT* ou bien de section *RULES*. La grammaire des politiques QoSL4BP est définie dans la figure 5.9.

5.4.2 Cible des politiques

La section *SCOPE* de la politique désigne la cible d'application de la politique QoSL4BP. Cette cible représente une région de la composition de services et s'exprime sous la forme d'une donnée de type « *activity_decl* ».

SCOPE_SECTION ::= SCOPE = { *activity_select* }

Le *SCOPE* peut ainsi exprimer soit une activité BPEL unique (simple ou composite) soit plusieurs activités (via l'utilisation de l'opérateur « OR » pour composer des activités ou de la *wilcard* « * »). Nous considérons que les traitements statiques et dynamiques de la politique doivent être appliqués de manière identique sur chaque activité individuelle composant le *SCOPE* de la politique. Cette stratégie d'application des traitements peut être rapprochée de la stratégie d'application de l'action des aspects dans la programmation par aspects (présentée à la section 2.2.3). Par extrapolation de cette analogie, le *SCOPE* de la politique peut ainsi être considéré comme une coupe constituée de plusieurs activités isolées. Ces activités sont alors appelées « activités de jonction ». Par exemple, le *SCOPE* défini dans la figure 5.4 est composé de deux activités de jonction. Les traitements de la politique de ce *SCOPE* s'appliquent indépendamment sur l'une et l'autre de ces activités de jonction.

Lorsqu'un traitement est effectué dans une politique et que ce traitement cible une activité, alors cette activité est définie par rapport à l'activité de jonction. Cette stratégie permet de mieux découpler les cibles des traitements. La donnée activité de jonction est d'ailleurs réifiée dans le langage et se nomme « JOIN_ACTIVITY ».

5.4.3 Traitements statiques

La section *INIT* de la politique permet de définir les traitements statiques concernant le *SCOPE*. Ces traitements sont effectués au pré-déploiement de la composition de services. Tous les traitements présentés à la section 5.3 ne peuvent être mis en oeuvre dans cette section.

L'objectif des traitements statiques est la sélection des services servant de partenaires aux activités de la composition de services. Il s'agit donc de positionner des offres

et des exigences de QdS et d'effectuer des recherches de services, ou bien de définir des liens directs avec certains services. Ainsi, les traitements statiques sont soit de type placement de contraintes (primitives « SET_OFFER » et « SET_REQUIREMENT »), soit de type sélection de services (primitives « PLANNING » et « BIND ») ou bien de type gestion des paramètres BPEL (primitives « SET_LOOP » et « SET_RATE »).

INIT_SECTION	::=	INIT = { *static_instr*+ }
static_instr	::=	*set_offer_instr* | *set_requirement_instr*
	|	*planning_instr* | *bind_instr*
	|	*set_loop_instr* | *set_rate_instr*

Le listing 5.10 illustre les traitements statiques correspondant aux spécifications de gestion de QdS de l'exemple UTP, présenté à la section 4.3.2. Ces quatre politiques correspondent respectivement à la spécification des quatre besoins spécifiés par l'architecte intégrateur.

- la politique « sla_composite » (ligne 1 à 14) portant sur la composition entière comporte la définition des offres de la proposition de SLA exhibée au client, ainsi que celle des exigences de QdS portées par l'ensemble de la composition vers les fournisseurs. Les offres et les exigences diffèrent de manière à réaliser une plus-value dans le cout du service d'une part et à maintenir une marge de fiabilité au cas où la QdS des services varient de manière trop importante d'autre part. Il faut également noter que parmi les offres faites au client, le service composite se propose de fournir un mécanisme de garantie de livraison de type « ExactlyOnce ». Une instruction de planification des services est appliquée sur la composition selon une stratégie de « Back Tracking ».
- la politique « localisation_securite » (ligne 16 à 29) a pour effet de garantir des offres de QdS portées par l'activité composite « *flow* LocationScope », gérant ainsi elle même la planification des services de cette activité. Elle place également une contrainte exigeant que des mécanismes de cryptage doivent être mis en oeuvre par les fournisseurs de service. Une instruction de planification des services est appliquée sur cette activité selon une stratégie de programmation par contrainte.
- la politique « selection_RATP » (ligne 31 à 40) a pour objectif de lier l'invocation au service de transport avec le service Web de la RATP appelé « WS_RATP ». Etant donné que ce service ne possède pas de SLA mais que ses capacités de QdS sont approximativement connues, des offres de QdS sont placées de manière à ce que le reste de la composition puisse utiliser ces contraintes lors de leurs traitements de planification (en particulier, la planification effectuée par la politique « sla_composite »).
- la dernière politique, « choix_grapher » (ligne 42 à 48), a pour but de surcontraindre les exigences de QdS lors du choix du service partenaire de l'activité « InvokeGrapher ». Ce service doit coûter moins de 5 centimes à l'appel.

```
POLICY "sla_composite" = {
  SCOPE = { PROCESS }
  INIT = {
    SET_OFFER(([RELIABILITY:EXACTLYONCE],
               [RESPONSETIME < 25],
               [THROUGHPUT > 100],
               [COST = 35])) ;
    SET_REQUIREMENT(([RESPONSETIME < 23],
                     [THROUGHPUT > 110],
                     [COST < 32])) ;
```

```
    PLANNING(JOIN_ACTIVITY, BACK_TRACKING) ;
  }
  RULES = { .. }
}

POLICY "localisation_securite" = {
  SCOPE = { FLOW["LocationScope"] }
  INIT = {
    SET_OFFER(([RESPONSETIME < 13],
               [THROUGHPUT > 110],
               [COST < 15])) ;
    SET_REQUIREMENT(([SECURITY : ENCRYPTION],
                     [RESPONSETIME < 13],
                     [THROUGHPUT > 110],
                     [COST < 15])) ;
    PLANNING(JOIN_ACTIVITY, CP) ;
  }
  RULES = { .. }
}

POLICY "selection_RATP" = {
  SCOPE = { INVOKE["InvokeTransportation"] }
  INIT = {
    SET_OFFER(([RESPONSETIME < 5],
               [THROUGHPUT > 200],
               [COST < 10])) ;
    BIND(JOIN_ACTIVITY, "WS_RATP") ;
  }
  RULES = { .. }
}

POLICY "choix_grapher" = {
  SCOPE = { INVOKE["InvokeGrapher"] }
  INIT = {
    SET_REQUIREMENT(([COST < 5])) ;
  }
  RULES = { .. }
}
```

Listing 5.10: Traitements statiques liés aux spécifications de QdS de l'exemple UTP

5.4.4 Traitements dynamiques

La section *RULES* de la politique permet de définir les traitements dynamiques de QdS liés au *SCOPE* de la politique. Ces traitements sont effectués lors de l'exécution de la composition de services. A l'opposé de la section *INIT*, tous les traitements présentés à la section 2.3.3 peuvent être mis en oeuvre dans cette section.

L'objectif des traitements dynamiques est l'observation de la QdS à l'exécution, la réaction aux variations de QdS et la mise en oeuvre des mécanismes de QdS. Les traitements dynamiques sont modélisés sous la forme de règles s'inspirant des règles de type ECA, présentées à la section 2.3.3. Généralement, les règles ECA spécifient un événement déclencheur (par exemple l'arrivée d'un message ou le début d'un processus métier), des conditions (comme la vérification de l'instance d'un processus), ainsi que des actions à effectuer (telles que la levée d'une exception) lorsque l'événement déclencheur survient et dans le cas où les conditions sont validées. Dans le cas des règles QoSL4BP, les événements d'évaluation ne sont pas à préciser par l'utilisateur. Ces règles sont exécutées implicitement avant et après l'exécution de l'activité de jonction, comme précisé dans la section 6.4. En ce qui concerne la condition des règles, celle-ci est spécifiée grâce aux primitives renvoyant des valeurs. Il est alors possible de tester ces valeurs et de combiner les tests avec les opérateurs booléens usuels (NOT, AND et OR).

RULES_SECTION ::= RULES = { *dynamic_instr*+ }
dynamic_instr ::= *action_instr*
| *rule_decl*
rule_decl ::= *evaluation_decl* − > *action_decl*

evaluation_decl ::= *condition_decl*
| NOT (*evaluation_decl*)
| (*evaluation_decl* AND *evaluation_decl*)
| (*evaluation_decl* OR *evaluation_decl*)
action_decl ::= *action_instr*
| { *action_instr*+ }

condition_decl ::= *boolean_instr*
| *string_instr* == *string*
| *integer_instr operator integer*
boolean_instr ::= *read_client_instr* | *read_provider_instr*
| *violation_activity_instr* | *violation_provider_instr*
| *catch_instr*
string_instr ::= *bound_instr*
integer_instr ::= *sensor_instr* | *monitor_instr*
| *get_loop_instr* | *get_rate_instr*
operator ::= < | > | == | >= | =<

action_instr ::= *planning_instr* | *bind_instr*
| *set_loop_instr* | *set_rate_instr*
| *perform_client_instr* | *perform_activity_instr*
| *throw_instr* | *log_instr*

Le listing 5.11 exhibe les traitements dynamiques correspondant aux spécifications de gestion de QdS de l'exemple UTP, présenté à la section 4.3.2, et étendant les politiques introduites à la section 5.4.3. Ces quatre politiques correspondent respectivement à la spécification des quatre besoins spécifiés par l'architecte intégrateur.

– la politique « sla_composite » (ligne 1 à 16) lit le SLA signé avec le client afin de vérifier si le client a accepté le mécanisme de garantie de livraison proposé. Dans ce cas, le mécanisme est mis en oeuvre. Si ce mécanisme rejette une exception, celle-ci est prise en compte. Par ailleurs, la QdS de la composition est implicitement monitorée et une replanification est effectuée si une offre faite au client n'est plus satisfaite. Si la replanification échoue, alors l'erreur est tracée et une exception est rejetée vers le client.
– la politique « localisation_securité » (ligne 18 à 40) met en oeuvre l'envoi de messages cryptés avec l'algorithme « 3DES » utilisant la clé « utpKeyone » pour le service « InvokeLocation » et l'algorithme « RSA » utilisant la clé « utpKeytwo » pour le service « InvokePhoneBook ». Si le message envoyé génère une exception, alors l'erreur est tracée et le client final reçoit une exception. Par ailleurs, la QdS de cette activité est monitorée et une replanification est effectuée si une exigence portée par cette activité n'est plus satisfaite. Si la replanification échoue, alors

l'erreur est tracée et une exception est rejetée vers le client.
- dans la politique « selection_RATP » (ligne 42 à 51), si la QdS définie pour la politique n'est par respectée, alors cet événement est tracé et une exception est rejetée vers le client.
- dans la dernière politique, « choix_grapher » (ligne 53 à 64), si le service « Grapher » ne peut maintenir sa QdS, une replanification est effectuée pour changer de service. Si aucun autre service n'est trouvé, alors l'erreur est tracée et une exception est envoyée au client.

```
POLICY "sla_composite" = {
  SCOPE = { PROCESS }
  INIT = { .. }
  RULES = {
    READ_CLIENT([RELIABILITY:EXACTLYONCE]) ->
      PERFORM_CLIENT([RELIABILITY:EXACTLYONCE]) ;
    CATCH("RELIABILITY␣EXCEPTION") ->
      THROW("Echec␣du␣mecanisme␣de␣garantie␣de␣livraison");

    VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
      PLANNING(JOIN_ACTIVITY, BACK_TRACKING) ;
    CATCH("PLANNING␣EXCEPTION") -> {
      LOG("Echec␣de␣la␣planification␣globale.");
      THROW("La␣QdS␣du␣service␣UTP␣n'est␣plus␣maintenue.");
    }
  }
}

POLICY "localisation_securite" = {
  SCOPE = { FLOW["LocationScope"] }
  INIT = { .. }
  RULES = {
    PERFORM_ACTIVITY(INVOKE["InvokeLocation"] IN SEQUENCE[0],
                     [SECURITY : ENCRYPTION],
                     ("3DES", "utpKEYone")) ;
    PERFORM_ACTIVITY(INVOKE["InvokePhoneBook"] IN SEQUENCE[1],
                     [SECURITY : ENCRYPTION],
                     ("RSA", "utpKEYtwo")) ;
    CATCH("SECURITY␣EXCEPTION") -> {
      LOG("Echec␣dans␣l'envoi␣du␣message␣crypte.");
      THROW("Echec␣du␣service␣UTP.");
    }

    VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
      PLANNING(JOIN_ACTIVITY, CP) ;
    CATCH("PLANNING␣EXCEPTION") -> {
      LOG("Echec␣de␣la␣planification␣de␣LocationScope.");
      THROW("La␣QdS␣du␣service␣UTP␣n'est␣plus␣maintenue.");
    }
  }
}

POLICY "selection_RATP" = {
  SCOPE = { INVOKE["InvokeTransportation"] }
  INIT = { .. }
  RULES = {
    VIOLATION_ACTIVITY(JOIN_ACTIVITY) -> {
      LOG("La␣QdS␣du␣service␣RATP␣n'est␣plus␣maintenue.") ;
      THROW("La␣QdS␣du␣service␣UTP␣n'est␣plus␣maintenue.") ;
    }
  }
}

POLICY "choix_grapher" = {
  SCOPE = { INVOKE["InvokeGrapher"] }
  INIT = { .. }
  RULES = {
    VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
      PLANNING(JOIN_ACTIVITY, BACK_TRACKING) ;
    CATCH("PLANNING␣EXCEPTION") -> {
      LOG("Echec␣de␣la␣planification␣de␣InvokeGrapher.");
      THROW("La␣QdS␣du␣service␣UTP␣n'est␣plus␣maintenue.");
} } }
```

Listing 5.11: Traitements dynamiques liés aux spécifications de QdS de l'exemple UTP

5.5 Propriétés du langage

L'utilisation d'un langage dédié offre aux utilisateurs certaines propriétés améliorant la robustesse du code qu'il permet d'écrire. Dans notre contexte, l'utilisation du langage QoSL4BP offre un certain nombre de propriétés intéressantes pour la gestion de la QdS dans les compositions de services :

Synchronisation implicite des traitements A l'exécution, la gestion de la QdS s'effectue en parallèle de l'exécution de la composition de services. Or, la synchronisation des traitements liés à ces deux préoccupations constitue un challenge à résoudre. Par exemple, un message devant être crypté doit être traité par le mécanisme de sécurité correspondant juste avant d'être envoyé au service (comme c'est le cas pour la politique « localisation_securité » présentée à la section 5.4.4). Dans le cas de règles ECA usuelles, l'utilisateur doit spécifier des événements pour l'évaluation de la condition ou la mise en oeuvre de l'action. Cependant, si l'événement spécifié est incorrect, alors la règle peut ne pas avoir de sens. Dans notre cas, les événements ne sont pas à la charge de l'architecte intégrateur.
Le langage QoSL4BP est conçu comme un langage déclaratif permettant à l'architecte intégrateur de ne pas se préoccuper de l'exécution de la composition de services et ainsi de ne pas effectuer d'erreur lors de la synchronisation entre les traitements liées à la composition de services et ceux liés à la gestion de la QdS.

Typage des activités BPEL ciblées par les traitements de QdS Les traitements de QdS s'appliquent sur des parties de la composition de services désignées par les paramètres des primitives QoSL4BP. Si certaines primitives peuvent cibler des activités de tout type, certaines ne peuvent que cibler certains types d'activité. Ainsi, la primitive « Bind » a pour cible une activité de type *invoke* tandis que la primitive « Monitor » peut cibler tout type d'activité. C'est grâce au formalisme de sélection et de navigation des activités intégré au langage QoSL4BP que cette vérification peut directement avoir lieu à la compilation.

Composition de politiques Deux politiques rentrent en interférence lorsque l'intersection de leur *SCOPE* n'est pas vide. Etant donné la configuration des activités dans un document BPEL4WS, ainsi que le formalisme utilisé pour spécifier les cibles des politiques, le type d'intersection qu'il est possible de réaliser avec des politiques QoSL4BP est soit de type inclusion, soit de type égalité des domaines. Par ailleurs, la dichotomie statique/dynamique clairement mise en évidence dans la structure des politiques permet une meilleure gestion des interférence aux phases statique et dynamique :

- dans le cas d'une inclusion, la politique, dont le *SCOPE* est inclu dans celui d'une autre politique, est plus locale que cette autre politique. A l'étape statique, si les traitements statiques de la politique la plus locale spécifient des offres de QdS, alors la politique englobante n'a pas le contrôle de la QdS de la zone couverte par le *SCOPE* de la politique locale et voit la zone couverte par le *SCOPE* de la politique locale comme une boîte noire. Sinon, les exigences de QdS de la politique englobante s'ajoutent à celles de la politique locale. A l'étape dynamique, les

traitements des deux politiques sont tous pris en compte, à la seule exception que la planification des services éventuellement effectuée par la politique englobante ne peut pas remplacer les services de la politique locale si celle-ci possède des offres.
- dans le cas d'une égalité des *SCOPES*, les deux politiques sont à un même niveau et aucune n'est privilégiée par rapport à l'autre. Ainsi, à l'étape statique, les ensembles d'offres et d'exigences s'agrègent et, si des contraintes sont incompatibles, alors le processus échoue. A l'étape dynamique, les traitements des deux politiques sont tous pris en compte.

Prise en charge des exceptions Certains traitements du langage QoSL4BP sont susceptibles d'échouer. Il s'agit en particulier des traitements liés à la planification (primitive « PLANNING ») ainsi que des traitements liés à la mise en oeuvre des mécanismes de QdS (primitives « PERFORM_CLIENT » et « PERFORM_ACTIVITY »). L'utilisation de ces primitives est donc susceptible de provoquer un échec à l'exécution. La prise en charge des exceptions est effectuée par la primitive « CATCH » qui permet de récupérer l'échec de ces primitives. A la compilation d'une politique, la présence de ce mécanisme de récupération est testée pour chaque primitive susceptible d'échouer. Cette propriété permet une meilleure robustesse du code écrit par l'architecte intégrateur.

Finitude du traitement lié à la QdS Si la synchronisation des traitements est une propriété importante du langage QoSL4BP, il est également nécessaire de garantir que les traitements de QdS se terminent afin de garantir la bonne exécution de la composition de services. Généralement, un traitement ne termine pas lorsqu'il se produit un interblocage (*deadlock*) ou un cycle infini. La limitation volontaire des structures de contrôle dans le langage QoSL4BP ne permet pas d'effectuer de boucles. Par ailleurs, la stratégie de gestion des règles, présentée à la section 6.4, permet de garantir que le traitement des règles aboutit.

5.6 Conclusion

Ce chapitre a présenté le langage QoSL4BP offert à l'architecte intégrateur afin qu'il puisse spécifier la gestion de QdS liée à une composition de services. Ce langage dédié encapsule le domaine de la gestion de QdS et fait apparaître dans son modèle de données et ses primitives, les données du domaine ainsi que les traitements mis en évidence pour gérer la QdS des compositions. En particulier, le modèle de données réifie les activités BPEL et les contraintes de QdS, tandis que les primitives se décomposent en trois sous domaines que sont la la gestion des accords, l'observation de la QdS et la gestion des mécanismes de QdS (sécurité, garantie de livraison, exception).

Le langage QoSL4BP permet de structurer les instructions sous la forme de politiques comportant une cible (région d'application de la politique sur la composition de services), une section regroupant les traitements statiques mis en oeuvre au prédéploiement de la composition et une section regroupant les traitements dynamiques mis en oeuvre à l'exécution de la composition.

Ces traitements définissent des objectifs et des règles offrant des propriétés intéressantes pour le domaine de la gestion de QdS. Tout d'abord, la synchronisation des traitements entre gestion de QdS et exécution de la composition de services. Ensuite, le langage vérifie le typage des activités BPEL ciblées par les traitements de QdS, ainsi que la bonne prise en compte des exceptions pouvant être rejetées. Enfin, le langage garantie la finitude des traitements de QdS et exploite un ensemble de règles permettant une composition cohérente des politiques.

Chapitre 6

Modèle d'exécution du langage QoSL4BP

Ce chapitre présente les traitements permettant de mettre en oeuvre le langage QoSL4BP présenté au chapitre 5. Compte tenu de la conception du langage QoSL4BP, les traitements liés à la gestion de QdS consistent en des traitements statiques ainsi qu'en des traitements dynamiques prenant respectivement place lors du pré-déploiement et de l'exécution de la composition de services.

Ayant fait le choix d'une approche non intrusive pour la mise en oeuvre des traitements de QdS (voire section 4.3.2), l'exécution des traitements dynamiques du langage QoSL4BP requiert une étape de transformation de la composition de services pour faire le lien avec la gestion de la QdS. Ainsi, le processus global de mise en oeuvre du langage QoSL4BP, présenté à la section 6.1, est composé d'une étape de traitement statique de la QdS (section 6.2), d'une étape de transformation de la composition de services (section 6.3), puis d'une étape de traitement dynamique de la QdS (section 6.4). Un prototype, nommé « ORQOS » (ORchestration Quality Of Service), implémentant la logique de ces divers traitements est présenté dans la section 6.5.

Sommaire

6.1 Introduction

Le langage QoSL4BP encapsule la préoccupation de gestion de la QdS pour les compositions de services. Ainsi, la mise en oeuvre du langage QoSL4BP a comme particularité de prendre place lors des étapes de mise en oeuvre du langage BPEL. La figure 4.4 de la section 4.3.1 exhibe une vue générale des étapes de mise en oeuvre du BPEL ainsi que les traitements additionnels requis par la gestion de la QdS. Les diverses sections des politiques QoSL4BP sont utilisées à chaque étape pour mettre en oeuvre ces traitements :

Etape statique de planification des services Cette étape de traitement prend place au pré-déploiement de la composition. Une fois que la composition de services a été spécifiée à l'aide du langage BPEL et que les partenaires potentiels pour les activités de la composition ont été identifiés, il faut effectuer la sélection des services en fonction des exigences et des offres de QdS placées sur la composition. Cette sélection, dont le processus est détaillé dans la section 6.2, fait usage des spécifications statiques contenues dans la section *INIT* des politiques, ainsi que des informations de leur *SCOPE*. Dans le cas où il n'existe aucune configuration de services répondant aux diverses exigences de QdS des politiques QoSL4BP, alors le processus échoue.

Etape statique de transformation de la composition de services La mise en oeuvre de la partie dynamique des politiques QoSL4BP impose « d'ouvrir le système observé », c'est à dire la composition de services. Un des points saillants de notre approche consiste à rester non intrusif vis à vis de la plateforme d'exécution du BPEL. Comme présenté dans la section 4.2.4, notre stratégie d'ouverture consiste à insérer des indirections dans le document BPEL et à effectuer des interceptions. Pour cela, le document BPEL doit être modifié, ce qui n'est plus possible une fois la composition déployée sur le moteur BPEL. Ainsi, une étape statique de transformation de la composition, détaillée dans la section 6.3 est réalisée au pré-déploiement. Elle requiert comme information, les sections *SCOPE* des politiques QoSL4BP. Une fois cette étape réalisée, le document BPEL4WS transformé peut être déployé sur le moteur d'orchestration.

Etape dynamique de gestion de la QdS Cette étape, décrite dans la section 6.4, a lieu à l'exécution de la composition. A cette étape, les règles réactives de la partie dynamique des politiques QoSL4BP sont mises en oeuvre afin de réagir aux variations de QdS à l'exécution et mettre en place certains mécanismes de QdS. Ce sont donc les informations de la section *RULES* des politiques qui sont utilisées. Les règles sont instanciées à partir des politiques qui ont été chargées au pré-déploiement. Elles reçoivent des informations de la composition de services d'une part via les indirections qui ont été ajoutées dans le document BPEL à l'étape statique et d'autre part via l'interception des messages échangés avec les divers partenaires.

6.2 Etape de recherche des services de la composition

L'étape de recherche des services de la composition utilise les informations de *SCOPE* et met en oeuvre la section *INIT* des politiques QoSL4BP avant le déploiement de la composition. Elle prend en entrée le document BPEL de composition de services, les politiques QoSL4BP et la liste des services potentiels pour les partenaires de la composition ainsi que leur SLA. Cette étape est constituée de trois sous étapes successives prenant chacune en compte l'ensemble des spécifications statiques des politiques.

Tout d'abord, la composition est transcrite sous la forme d'un arbre portant des informations de QdS (obtenues dans la section *INIT* des politiques). Une stratégie de décomposition est ensuite pratiquée sur l'arbre obtenu afin de potentiellement obtenir plusieurs sous-arbres de taille moindre. A ce stade, des mécanismes de planification sont appliqués sur chacun de ces sous-arbres dans le but de sélectionner les services partenaires de la composition. A l'issue de l'étape de recherche, le processus échoue ou bien une liste de services partenaires à invoquer depuis la composition est retournée ainsi que, potentiellement, le SLA du service composite.

6.2.1 Transcription de la composition de services en arbre

Afin de pouvoir mettre en relation les offres et les exigences de QdS des différentes activités de la composition, nous utilisons une modélisation de la composition sous la forme d'un arbre. Le choix d'une modélisation sous la forme d'un arbre se justifie par le fait qu'elle permet d'associer les informations de la structure la composition (contenues dans la structure de l'arbre) et les informations de QdS (contenues dans les noeuds et les feuilles de l'arbre). Par ailleurs, la structure de l'arbre fait apparaître explicitement les liens entre la QdS des activités composites et la QdS des activités qu'elles englobent. Dès lors, il devient possible de déduire aisément la QdS des activités composites (notamment par le biais de méthodes telles que [CSM+04] décrite à la section 3.3.2) ou bien inversement, d'appliquer des exigences de QdS sur les activités englobées.

L'algorithme de création de l'arbre procède de la manière suivante :
- les activités *invoke* sont modélisées par des feuilles,
- les activités composites sont modélisées par des noeuds dont les fils correspondent à la modélisation (feuille ou noeud) des activités qu'elles englobent,
- l'activité englobant la composition constitue la racine de l'arbre.

La figure 6.1 montre l'exemple de transcription de la composition UTP (présentée à la section 4.3.2) en arbre.

Une fois l'arbre construit, les éléments de l'arbre sont ensuite enrichis d'informations de QdS provenant d'une part des informations contenues dans la section *INIT* des politiques, et d'autre part des informations délivrées par les fournisseurs de service. Le modèle des éléments de l'arbre est présentée dans la figure 6.2. Ce modèle possède :
- un nom et un index (correspondant au nom et de l'index de l'activité dans la composition BPEL),
- les éventuelles offres et exigences provenant des primitives « SET_OFFER »et « SET_REQUIREMENT »,
- dans le cas d'une feuille, alors il s'agit d'une activité *invoke* ou bien d'un sous-arbre (voire section 6.2.2). Les informations liées aux partenaires sont renseignées

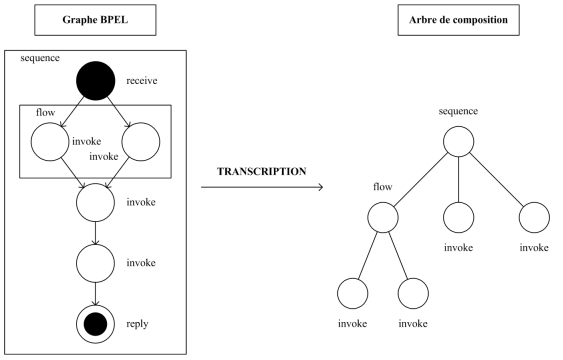

FIGURE 6.1: Transcription de la composition BPEL en arbre

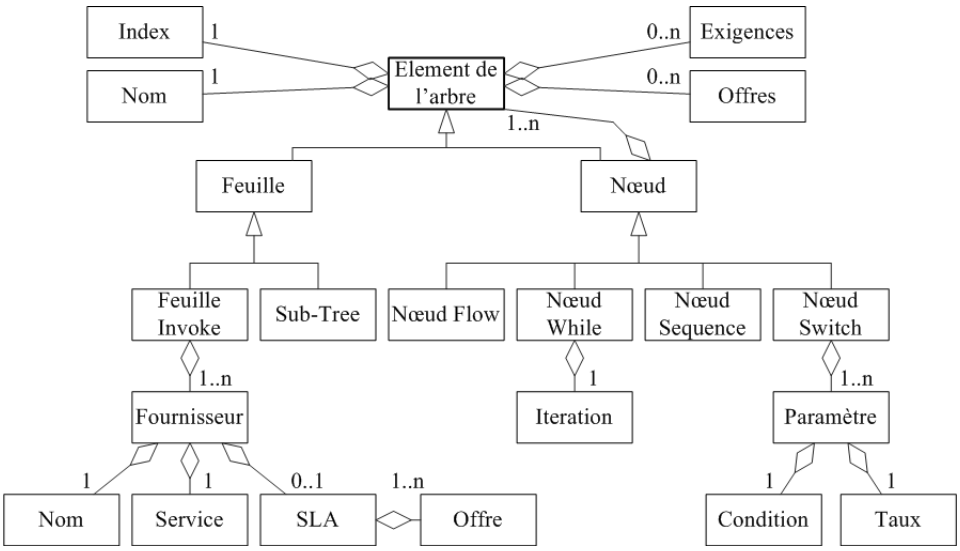

FIGURE 6.2: Modèle des éléments de l'arbre

dans l'arbre, et si une instruction « BIND » spécifie un fournisseur pour cette activité, alors celui-ci est immédiatement sélectionné.

– dans le cas d'un noeud, alors il s'agit d'une activité composite *Flow*, *Sequence*, *While* ou *Switch*. Les instructions « SET_RATE » et « SET_LOOP » permettent de compléter les informations des noeuds *While* et *Switch*.

Le listing 6.1 fait apparaitre deux instances de ce modèle en prenant comme exemple deux activités de la composition UTP présentée à la section 4.3.2. La première activité « LocationScope » est de type *flow* et la seconde est l'activité *invoke* « Grapher ». Les valeurs proviennent des politiques « sla_composite » et « choix_grapher » du listing 5.10.

```
NODE(Flow)
- Name = "LocationScope"
- Index = 0
- Offers = ([RELIABILITY:EXACTLYONCE], [RESPONSETIME < 25], [THROUGHPUT > 100],
            [COST = 35])
- Requirements = ([RESPONSETIME < 23], [THROUGHPUT > 110], [COST < 32])

LEAF(Invoke)
- Name = "Grapher"
```

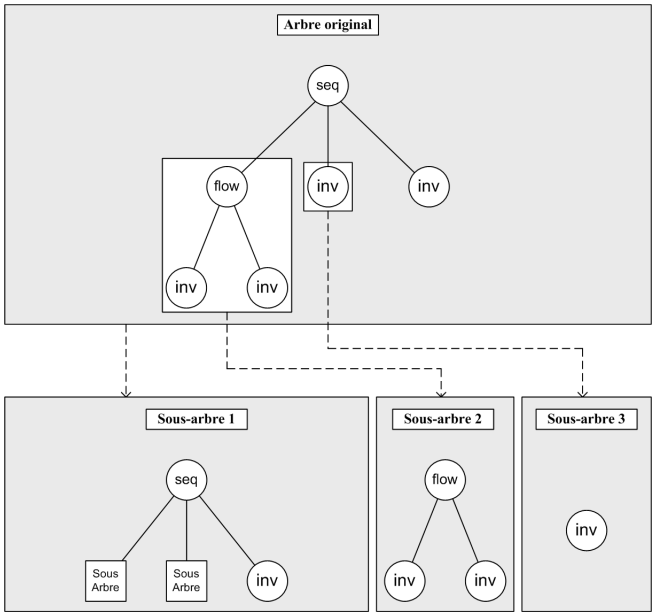

FIGURE 6.3: Décomposition en sous-arbres

```
- Index = 0
- Offers = null
- Requirements = ([COST < 5])
- Providers = {
    Provider["Mappy", "http://www....mappy", Offers = ([RESPONSETIME < 23],..)]
    Provider["GoogleMap", ....]
  }
```

Listing 6.1: Exemple d'instance d'éléments d'arbre

6.2.2 Stratégie de décomposition

Le deuxième traitement consiste à décomposer l'arbre en plusieurs sous-arbres et a pour but de réduire les espaces de recherche lors de la planification. Pour effectuer cette décomposition, l'ensemble des éléments de l'arbre sont parcourus. Pour chaque élément rencontré portant des offres, cet élément et ses potentiels fils sont détachés pour devenir un nouveau sous-arbre et l'ancien élément de l'arbre original est remplacé par une feuille de type « sous-arbre ». Le parcours s'applique ensuite récursivement dans chaque sous-arbre. La figure 6.3 illustre la transformation de l'arbre issue de la composition UTP en sous-arbres.

L'intérêt de cette décomposition est de réduire l'effort de planification des services en isolant les divers ensembles de services à replanifier. Cette isolation tire profit des offres attachées aux activités de la composition. Cette stratégie est sensiblement similaire à l'algorithme de « sélection par patron » présenté dans la section 3.3.4 (la problématique de la combinatoire liée à la recherche des services est également discutée dans cette section). Dans notre cas, au lieu de patrons, la planification des services est effectuée sur des sous-arbres. La complexité pour une recherche exhaustive avec cette stratégie correspond à $\sum_{i=0}^{n} s^{c_i}$ qui est de l'ordre $\Theta(s^{\max(c_i)})$ (avec n représentant le nombre de sous-arbres, $c_i, i\epsilon[1,n]$ le nombre de feuilles de type *invoke* présentes dans le sous-arbre i, et s le nombre de fournisseurs de service moyen). Par conséquent, en

plaçant stratégiquement des offres de QdS sur les activités de sa composition, un architecte intégrateur est capable de réduire significativement l'effort de recherche lié à la planification des services.

6.2.3 Planification des services

La planification des services est le traitement qui correspond à la recherche des services dont les offres de QdS satisfont l'ensemble des exigences portées par les activités du sous-arbre. Pour ce traitement, les sous-arbres générés sont exploités indépendamment les uns des autres. Un élément de type sous-arbre porte les offres qui ont été spécifiées par l'architecte intégrateur et le traitement de planification peut utiliser ces offres pour la satisfaction des exigences.

La structure sous forme d'arbre permet de calculer la QdS des activités composites de manière immédiate grâce à des règles de composition de QdS ainsi qu'aux paramètres liées aux activités BPEL. Le tableau 6.1 (inspiré des travaux présentés à la section 3.3) décrit ces règles de composition pour les diverses propriétés de QdS du langage QoSL4BP. Rappelons que nous avons fait le choix de nous intéresser aux valeurs moyennes des propriétés.

	Noeud Sequence	Noeud Flow	Noeud Switch	Noeud While
Temps réponse	$\sum_i tps_i$	$\max_i(tps_i)$	$\sum_i tps_i * taux_i$	$tps_{fils} * iter$
Débit	$\min_i(deb_i)$	$\min_i(deb_i)$	$\sum_i \frac{deb_i}{taux_i}$	$\frac{debit_{fils}}{iter}$
Disponibilité	$\prod_i dispo_i$	$\prod_i dispo_i$	$\sum_i dispo_i * taux_i$	$dispo_{fil}^{iter}$
Coût	$\sum_i cout_i$	$\sum_i cout_i$	$\sum_i cout_i * taux_i$	$cout_{fils} * iter$
Sécurité	$\cap_i meca_i$	$\cap_i meca_i$	$\cap_i meca_i$	$meca_{fils}$
Livraison	$\cap_i meca_i$	$\cap_i meca_i$	$\cap_i meca_i$	$meca_{fils}$

TABLE 6.1: Règles de composition des propriétés de QdS

A l'instar des quatre caractéristiques de QdS, nous avons choisi deux stratégies de recherche car elles se mettent naturellement en oeuvre avec notre modélisation. Il est toutefois envisageable de réutiliser les stratégies de recherche proposées dans l'état de l'art pour enrichir notre approche.

Recherche exhaustive avec *back tracking* Une première stratégie consiste à passer en revue toutes les combinaisons possibles d'offres fournisseurs du sous-arbre puis de propager les offres de QdS vers les activités composites du sommet de l'arbre grâce aux règles du tableau 6.1 de manière à comparer l'ensemble des offres et l'ensemble des exigences pour chacune des activités du sous-arbre. Cependant, afin de limiter le nombre de calculs et de tirer profit de l'expression d'éventuelles exigences placées sur des activités à des niveaux intermédiaires, il est possible d'appliquer une stratégie de *back tracking*. Pour cela, les exigences sont vérifiées de manière incrémentale vers les niveaux composites plus élevés. Lorsque les exigences ne sont plus respectées à un certain

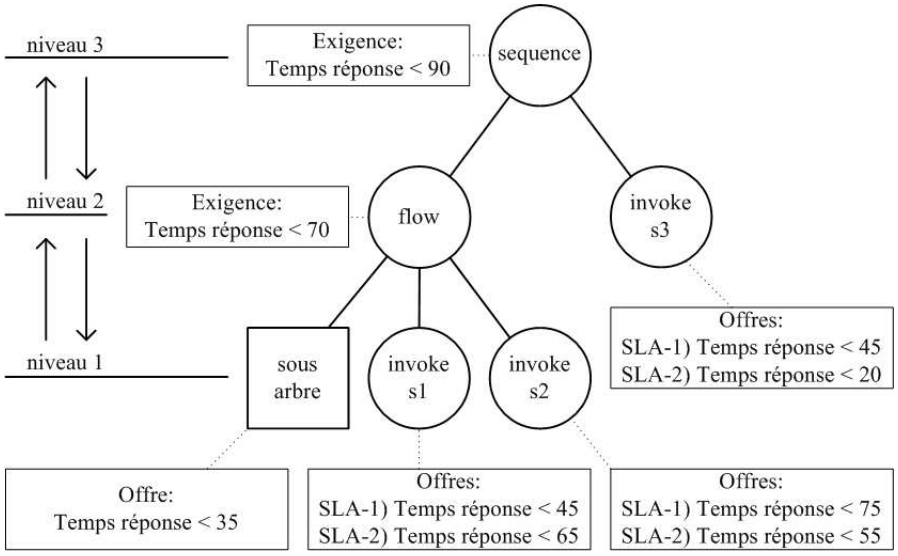

FIGURE 6.4: Exemple de mise en oeuvre de la planification

niveau composite, alors la combinaison partielle des offres fournisseurs est éliminée. La force du *back tracking* est d'éviter les combinaisons partielles ne pouvant aboutir, diminuant ainsi le temps d'exécution. Par conséquent, en plaçant stratégiquement des exigences de QdS sur les activités de sa composition, un architecte intégrateur est capable de réduire significativement l'effort de recherche lié à la planification des services. En particulier, cette stratégie tire profit de la capacité du langage QoSL4BP à spécifier des offres et des exigences à divers niveaux de granularité de la composition.

La figure 6.4 exhibe l'exemple d'un sous-arbre comportant trois niveaux. Des exigences sont placées sur les deuxième et troisième niveaux, tandis que les feuilles de type *invoke* possèdent chacune deux SLAs avec une offre de QdS. Le tableau 6.2 relate les étapes successives de calcul. Des offres des niveaux inférieurs sont choisies puis propagées vers les niveaux supérieurs. Dès lors qu'une exigence n'est plus satisfaite à un niveau supérieur, la combinaison partielle échoue et l'algorithme fait varier les offres situés à un niveau inférieur. Dès qu'une solution satisfait l'ensemble des exigences, le processus de planification aboutit.

Etape	ssArbre	$s1$	$s2$	$s3$	*flow* $(\max(pol, s1, s2))$	*sequence* $(flow + s3)$
1	35	45	75		75 (échec)	
2	35	45	55		55	
3	35	45	55	45	55	100 (échec)
4	35	45	55	20	55	75 (fin)

TABLE 6.2: Etapes de vérification des exigences

A l'issue des diverses planifications effectuées sur les sous-arbres, l'ensemble des services dont les offres de QdS satisfont aux exigences spécifiées dans les traitements statiques des politiques est identifié et l'architecte intégrateur passe un accord avec les fournisseurs de ces services. Dans le cas échéant, le traitement statique des politiques

échoue. L'architecte intégrateur doit alors revoir ses exigences de QdS ou bien agrandir les ensembles de fournisseurs potentiels pour chaque service de la composition.

Résolution par programmation par contraintes Le système obtenu par les règles projetées sur les divers sous-arbres, ainsi que les ensembles d'offres et d'exigences de QdS, correspond à un réseau de contraintes (assimilable à un CSP [1]) [Tsa93]. Dans ce réseau, les variables à résoudre correspondent aux offres des fournisseurs des différentes feuilles représentant des activités *invoke*. Une manière de traiter le système de contrainte consiste à mettre en oeuvre les techniques traditionnelles liées à la programmation par contraintes (CP [2]) [Dec03]. Les travaux de Jaeger et al., décrits à la section 3.3.4, étudient plusieurs algorithmes et heuristiques afin de résoudre un tel réseau de contraintes.

6.3 Etape de transformation de la composition de services

L'étape de transformation de la composition de services utilise les informations de la section *SCOPE* après que l'étape de recherche de services ait été validée. Cette étape prend en entrée le document BPEL de composition de services et les politiques QoSL4BP. Elle est constituée de deux traitements indépendants : d'une part des indirections sont tissées dans la composition de services (section 6.3.1), et d'autre part les partenaires des activités appartenant à un *SCOPE* de politique sont redirigés vers un intercepteur (section 6.3.2). Ces indirections et ces redirections ont pour rôle d'appeler la plateforme en charge des politiques QoSL4BP afin de mettre en oeuvre les traitements dynamiques de QdS à l'exécution. Présentée dans la section 4.2.4, cette stratégie permet que la plateforme d'exécution de la composition et celle en charge des politiques coopèrent de manière non intrusive, et donc qu'elles puissent évoluer librement l'une de l'autre. A l'issue de cette étape de transformation, la composition de services modifiée peut être déployée sur un moteur BPEL quelconque.

6.3.1 Tissage d'indirections dans la composition

Les politiques QoSL4BP s'appliquent sur des activités BPEL simples ou composites. Etant donné que le BPEL n'est pas un langage généraliste, il n'est pas possible de tisser directement dans le code de la composition le code lié aux préoccupations de QdS. Cependant, le langage BPEL permet d'effectuer des appels vers des services Web pouvant eux même exécuter du code généraliste. Il apparaît donc tout à fait réalisable d'invoquer le code lié à la gestion dynamique de la QdS en créant dans le code BPEL des indirections vers une interface de la plateforme exécutant les politiques QoSL4BP. Ainsi, des indirections (sous la forme d'activités *invoke*) sont tissées avant et après les activités ciblées par les politiques possédant des traitements dynamiques (*i.e* les politiques possédant une section *RULES*). Ces indirections appellent la plateforme en charge de l'exécution des politiques QoSL4BP correspondante. La figure 6.5 exhibe ce mécanisme de tissage d'indirection pour la politique « localisation_securite » s'appli-

1. Constraint Satisfaction Problem
2. Constraint Processing

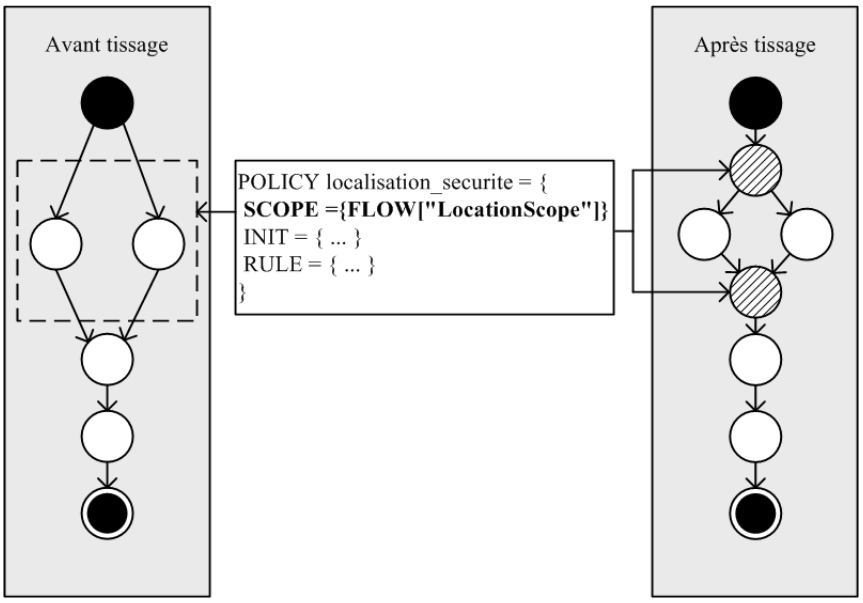

FIGURE 6.5: Exemple de tissage dans une composition de services

quant sur l'activité composite « *flow* LocationScope » de la composition UTP présentée à la section 4.3.2.

A l'instar de la programmation orientée aspect présentée dans la section 2.2.3, les politiques QoSL4BP et leur mise en oeuvre font ainsi apparaître :

1. Une coupe modularisée par le *SCOPE* de la politique et composée d'une ou plusieurs « activités de jonction », qui sont les endroits dans le code de base (le code BPEL) où sont insérées les indirections vers les politiques.
2. Des actions, correspondant à la section *RULES* des politiques, à mettre en oeuvre lorsque le flot d'exécution de la composition entre dans une activité de jonction et lorsqu'il la quitte. En particulier, ce mode de fonctionnement est similaire aux aspects de type *around*. Ce choix permet de garantir qu'avant et après l'activité de jonction, la QdS a bien été gérée par les traitements de la politique.
3. Un mécanisme de tissage permettant de faire le lien entre une activité de jonction du code de base et les traitements de QdS à mettre en oeuvre. Pour que la plateforme en charge des politiques soit informée de l'exécution de l'activité de jonction, deux indirections (sous forme d'activités de type « invoke ») sont tissées autour du point de jonction.

Lors de l'appel de la plateforme en charge de l'exécution des politiques QoSL4BP, des informations d'identification sont transmises afin que celle-ci puisse savoir quelle politique doit être mise en oeuvre. Une information d'identification est composée d'un triplet (nom de la composition, nom de l'activité et nom de la politique). Les listings 6.2 et 6.3 décrivent respectivement le code BPEL avant et après tissage des indirections. En particulier, le listing 6.3 montre comment s'effectue la préparation des informations d'identification ainsi que les indirections autour d'une activité de jonction. La plateforme appelée correspond à la plateforme ORQOS décrite dans la section 6.5.

```
<flow name="LocationScope">....</flow>
```

Listing 6.2: Code BPEL avant tissage des indirections

```
<sequence>
  <!--preparation du message d'identification␣de␣la␣politique-->
␣␣<assign>
␣␣␣␣<copy><from>utpserviceProcess</from>
␣␣␣␣<to␣part="Process"␣variable="PolicyID"/></copy>
␣␣␣␣<copy><from>LocationScope</from>
␣␣␣␣<to␣part="JoinActivity"␣variable="PolicyID"/></copy>
␣␣␣␣<copy><from>localisation_securite</from>
␣␣␣␣<to␣part="Policy"␣variable="PolicyID"/></copy>
␣␣</assign>

␣␣<!--notifyBefore-->
␣␣<invoke␣inputVariable="PolicyID"␣partnerLink="ORQOS"␣operation="notifyBefore"../>

␣␣<!--joinActivity-->
␣␣<flow␣name="LocationScope">....</flow>

␣␣<!--notifyAfter-->
␣␣<invoke␣inputVariable="PolicyID"␣partnerLink="ORQOS"␣operation="notifyAfter"../>
</sequence>
```

Listing 6.3: Code BPEL après tissage des indirections

6.3.2 Redirection des messages échangés entre partenaires

Afin de pouvoir mettre en oeuvre certains mécanismes spécifiés dans les règles des politiques (tels que le *monitoring* ou la sécurité), il est nécessaire de pouvoir observer et agir sur les messages échangés entre la composition et ses partenaires (clients et services). Ces échanges ne peuvent pas être manipulés directement dans la composition BPEL4WS qui n'offre pas l'expressivité suffisante. Cependant, étant donné que les messages sont échangés via une couche *middleware*, un mécanisme d'interception de type *proxy* peut être mis en oeuvre. Ainsi, les messages peuvent être interceptés s'ils ont été redirigés depuis le BPEL vers un intercepteur de la plateforme de gestion de QdS.

Mettre en place un intercepteur entre la composition et ses partenaires requiert de modifier la cible des invocations dans les activités du document BPEL de composition. Il s'agit donc de modifier les *partnerLink* des activités *invoke* inclues dans le *SCOPE* des politiques afin que les messages soient redirigés vers l'intercepteur. L'expressivité du langage BPEL permet d'assigner dynamiquement un *partnerLink*. Par ailleurs, afin de pouvoir associer les messages en lien avec les activités *invoke*, il est possible d'agréger des informations d'identification aux entêtes des messages SOAP envoyés par redirection. Ainsi lorsque l'intercepteur reçoit des messages SOAP, il peut faire le lien entre les messages SOAP reçus et les traitements à mettre en oeuvre. Ces informations consistent en un couple (nom de la composition et nom de l'activité). Le code BPEL correspondant au remplacement de l'activité d'invocation « InvokeGrapher » de la composition UTP (présentée à la section 4.3.2) par un *partnerLink* dynamique est présenté dans le listing 6.4.

```
<partnerLinks>
  <partnerLink name="proxyPL" partnerLinkType="policyns:proxyPT" partnerRole="proxyRole"/>
</partnerLinks>

<assign>
  <copy>
    <from>
      <wsa:EndpointReference
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
```

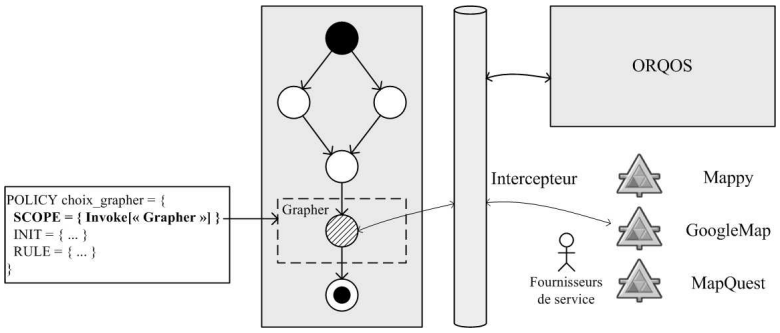

FIGURE 6.6: Exemple de redirection pour l'activité « InvokeGrapher »

```
10          xmlns:policyns="http://policy/">
            <wsa:Address>http://localhost:8082/services/proxyService</wsa:Address>
            <wsa:ReferenceProperties>
              <policyns:PolicyRef>
                <policyns:Composition>utpserviceProcess</policyns:Composition>
15              <policyns:Activity>InvokeGrapher</policyns:Activity>
              </policyns:PolicyRef>
            </wsa:ReferenceProperties>
            <wsa:ServiceName PortName="proxy">policyns:proxyService</wsa:ServiceName>
          </wsa:EndpointReference>
20      </from>
        <to partnerLink="proxyPL" />
    </copy>
  </assign>

25 <invoke partnerLink="proxyPL" portType="policyns:proxyPT" operation="intercept"
    inputVariable="grapherRequest" outputVariable="grapherResponse"/>
```

Listing 6.4: Code BPEL des redirections

La figure 6.6 fait apparaître la redirection de l'activité « InvokeGrapher » à l'exécution. A l'exécution, l'intercepteur de notre plateforme ORQOS fait le lien entre les fournisseurs et le *partnerLink* de l'activité.

6.4 Etape de mise en oeuvre des règles

L'étape de mise en oeuvre des règles utilise les informations de la section *RULES* à l'exécution de la composition de services. Ces traitements peuvent être mis en oeuvre grâce aux indirections tissées au pré-déploiement (section 6.3.1) ainsi qu'aux redirections (section 6.3.2). Le processus de synchronisation des politiques et de la composition de services est détaillé dans la section 6.4.1. La section 6.4.2 décrit la mise en oeuvre globale des règles des politiques.

Pour simplifier le traitement des règles, nous considérons que les diverses instances de *workflow* se déroulent séquentiellement. En effet, cette simplification permet d'éviter les considérations relatives aux stratégies de synchronisation des fils d'exécution des politiques (telles que la mise en oeuvre des sémaphores). Cette limitation fait partie de nos perspectives d'étude présentées au chapitre 8.

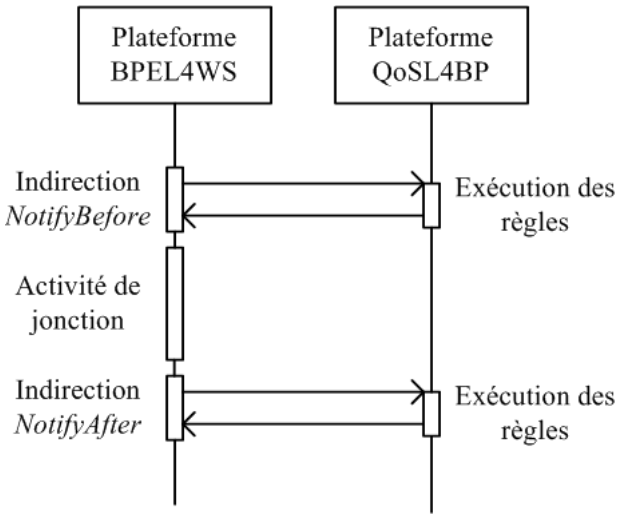

FIGURE 6.7: Synchronisation des traitements BPEL4WS et QoSL4BP

6.4.1 Synchronisation des règles QoSL4BP avec l'exécution du BPEL

L'étape de transformations de la composition de services est responsable du tissage des indirections placées avant et après les activités de jonction des *SCOPE* des politiques. A l'exécution, ces indirections appelent la plateforme responsable de la mise en oeuvre des politiques QoSL4BP. Les appels effectués par les indirections sont réalisés de manière synchrone afin que les règles s'exécutent tandis que le flot d'exécution du *workflow* est à l'arrêt. Cette synchronisation permet d'éviter les inconsistances qui découlerait de la gestion de la QdS effectuée en parallèle à l'exécution de la composition (par exemple dans le cas d'une replanification des services).

La figure 6.7 schématise l'exécution des traitements des activités BPEL et des traitements des politiques QoSL4BP : les règles de la politique sont appelées avant l'exécution de l'activité de jonction et le flot d'exécution BPEL doit attendre la fin de l'exécution des règles pour reprendre son cours. L'activité de jonction est exécutée, puis à sa terminaison, les règles sont appelées une nouvelle fois. Le flot d'exécution BPEL attend alors de nouveau la fin de l'exécution des règles pour reprendre son cours.

Une exception réside dans le cas où l'activité de jonction est la composition entière. Dans ce cas, elle comporte la première activité de type *receive* et la dernière activité de type *reply*. Or, il n'est pas possible d'insérer une indirection avant une activité de type *receive* ni après une activité de type *reply*. Dans ce cas précis, l'exécution des politiques a lieu lors de l'interception des messages échangés avec le client à l'entrée et à la sortie de la composition.

6.4.2 Algorithme de traitement de la section *RULES*

Le traitement d'une règle se décompose en deux étapes. La première étape consiste à évaluer la partie condition de la règle avec les valeurs du système. Si cette évaluation se révèle positive, la règle est dite activée, et la deuxième étape consiste alors à réaliser le traitement associé à la règle.

Cependant, le traitement de chacune des règles de la section RULES d'une politique affecte l'état du système et, par conséquent, peut potentiellement affecter le résultat de l'évaluation des autres règles. Il en résulte deux possibilités quant au résultat du traitement global des règles : soit le système converge vers un état stable (potentiellement à la suite d'un régime transitoire), soit le système ne converge pas vers un état stable et les règles sont répétées un nombre indéfini de fois sans aboutir.

Afin de comprendre le traitement que ce mécanismes implique, nous illustrons l'algorithme correspondant dans le listing 6.5.

```
EXECUTION_RULES (ETAT etat_systeme, RULE[] rules):

  VAR ETAT etat_precedent ;

  FAIRE:
    etat_precedent := etat_systeme ;
    POUR CHAQUE Regle r dans rules:
      VAR BOOLEAN resultat := EVALUER (r, etat_systeme) ;
      SI (resultat) ALORS EXECUTER (r, etat_systeme) ;
  TANT QUE (etat_precedent != etat_systeme)

FIN
```

Listing 6.5: Algorithme de traitement des règles par itération

Afin de ne pas stopper définitivement l'exécution de la composition de services, le langage QoSL4BP doit garantir la finitude de ses traitements. Cependant, sans modélisation précise des effets de chaque règle sur l'état du système et sans simulation avec un moteur de règles dans toutes les situations possibles, il n'est pas possible de garantir la finitude de l'algorithme d'itération sur l'état du système. Aussi, nous n'utilisons pas l'algorithme présenté dans le listing 6.5, et notre stratégie pour la mise en oeuvre des règles de la section *RULES* consiste à exécuter séquentiellement chacune des règles dans l'ordre selon lequel elles sont spécifiées sans les ré-évaluer. L'hypothèse actuelle pour la mise en oeuvre des règles est de considérer que l'architecte intégrateur garantit la cohérence dans l'exécution des règles en spécifiant un ordre adéquat dans le classement de ses règles. Cette hypothèse apparaît raisonnable compte tenu de l'expressivité actuelle du langage.

6.5 Mise en oeuvre avec la plateforme ORQOS

La plateforme ORQOS constitue notre implémentation du modèle d'exécution des politiques QoSL4BP présenté dans ce chapitre. Il peut toutefois exister d'autres implémentations de ce modèle. La section 6.5.1 présente les fonctionnalités de cette plateforme à travers l'étude de son architecture. La section 6.5.2 détaille les traitements permettant de mettre en oeuvre les instructions du langage QoSL4BP.

6.5.1 Présentation fonctionnelle

L'architecture de la plateforme ORQOS est composée de trois composants principaux que sont les gestionnaires d'événements, de règles et de traitements, ainsi que d'un registre en charge de stocker l'état de QdS du système. Ces composants sont eux-mêmes subdivisés en sous-composants comme exhibé dans la figure 6.8.

Composant « Gestionnaire d'événements » Le rôle du composant « Gestionnaire d'événements » est double. Tout d'abord, ce composant est responsable de faire

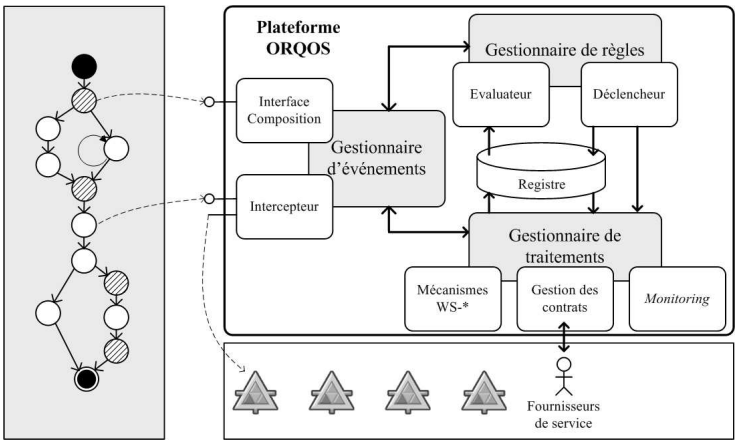

FIGURE 6.8: Architecture de la plateforme ORQOS

l'interface entre le moteur BPEL et les politiques QoSL4BP et d'intercepter les messages échangés entre le moteur BPEL et les participants. Ainsi les invocations effectuées par les indirections présentées à la section 6.3.1 ainsi que les redirections présentées à la section 6.3.2 contactent ce composant.

Son deuxième rôle consiste à répartir les informations reçues vers les composants « Gestionnaire de règles » et « Gestionnaire de traitements ». Il transmet au composant « Gestionnaire de règles » les entêtes des messages envoyés par les indirections tissées dans la composition, ainsi que les interceptions des messages du client. Comme présenté à la section 6.4.1, le composant « Gestionnaire d'événements » ne répond aux messages des indirections (ou à l'interception des messages du client) que lorsque les traitements dynamiques sont terminés, bloquant ainsi l'exécution de la composition pendant ces traitements. Il transmet également l'ensemble des messages reçus par les indirections et les interceptions au composant « Gestionnaire de traitements » afin que celui-ci puisse effectuer les traitements relatifs à ces messages (*monitoring*, mise en place de sécurité, etc.). Les messages interceptés sont bloqués tant que ces traitements ne sont pas terminés.

Composant « Gestionnaire de règles » Le composant « Gestionnaire de règles » a pour rôle d'exécuter les règles des politiques lorsqu'il reçoit un message du composant « Gestionnaire d'événements ». Pour identifier la politique dont il doit exécuter les règles, il utilise les informations d'identification transmises dans les messages (présentées dans la section 6.3.1 et 6.3.2). Il exécute ensuite les règles selon le traitement discuté dans la section 6.4.2. L'état de QdS du système est connu grâce aux informations contenues dans le registre de la plateforme ORQOS. Les informations qu'il contient sont utilisées pour l'évaluation des conditions des règles et l'exécution des règles a pour effet de le modifier ainsi que, potentiellement, de mettre en oeuvre des traitements du composant « Gestionnaire de traitements ». Une fois le traitement des règles terminé, ce composant rend la main au composant « Gestionnaire d'événements » afin que l'exécution de la composition puisse reprendre.

Dans l'exemple de la figure 6.5, les indirections transmettent le message SOAP présenté dans le listing 6.6. En recevant ces informations, le composant « Gestionnaire

de règles » sait qu'il doit exécuter les règles de la politique « localisation_securite » en initialisant la valeur de l'activité de jonction à « LocationScope ».

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:policyns="http://policy/">
  <SOAP-ENV:Body>

    <policyns:PolicyID>
      <policyns:Process> utpserviceProcess </policyns:Process>
      <policyns:JoinActivity> LocationScope </policyns:JoinActivity>
      <policyns:Policy> localisation_securite </policyns:Policy>
    </policyns:PolicyID>

  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 6.6: Exemple de message SOAP envoyé par une indirection

Composant « Gestionnaire de traitements » Le composant « Gestionnaire de traitements » a pour rôle de mettre en oeuvre les divers traitements spécifiés dans les règles des politiques. Ce composant est subdivisé en trois sous-composants ayant pour rôle respectif le traitement des mécanismes WS-*, la gestion des contrats et la gestion du *monitoring*. Chacun de ces sous-composants interagit avec le registre afin de mettre à jour l'état de QdS du système ou bien afin de récupérer des informations de QdS nécessaires pour leurs traitements respectifs. A l'instar des traitements dans le composant « Gestionnaire de règles », le composant « Gestionnaire de traitements » utilise les informations d'identification (présentées dans la section 6.3.1 et 6.3.2) transmises avec les messages reçus afin d'associer les informations de QdS aux activités de la composition. Les traitements de ce composant sont détaillés dans la section 6.5.2.

Registre « Etat du système » L'état de QdS du système est stocké dans un registre afin de décorreler l'exécution des règles de l'exécution des traitements de QdS. Ainsi, les mécanismes liés au *monitoring* peuvent procéder à la mise à jour des informations de QdS d'une activité à certains moments adéquats sans que les règles n'aient besoin d'attendre ces moments pour être évaluées. Inversement, si une règle déclenche un traitement tel que la mise en oeuvre d'un mécanisme de type jeton de sécurité pour une activité, ce jeton est stocké dans le registre jusqu'à ce que l'activité qui requiert ce jeton soit exécutée.

Ce registre contient les informations relatives aux spécifications de l'architecte intégrateur (modélisées par les informations présentées dans la figure 6.2), au *monitoring* à l'exécution, ainsi qu'aux mécanismes WS-*.

6.5.2 Présentation des traitements

Cette section présente les traitements du composant « Gestionnaire de traitements ». Il s'agit des briques de base permettant en oeuvre la gestion de la QdS.

Mécanismes WS-* La mise en oeuvre des mécanismes WS-* de sécurité et de garantie de livraison est effectuée par un sous-composant qui utilise les flux de messages SOAP que transmet le composant « Gestionnaire de traitements ». Ces messages proviennent des appels effectués par le client ou bien par les redirections placées dans la composition.

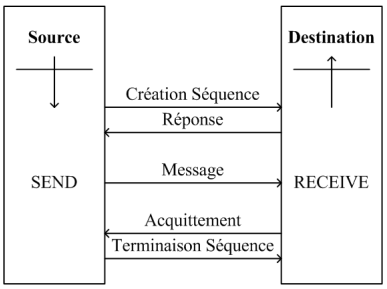

FIGURE 6.9: Mécanisme de garantie de livraison

En ce qui concerne la sécurité, la norme WS-Security [IBM04] spécifiant divers protocoles est implémentée par l'API WSS4J [Apa06b]. Les mécanismes de sécurité pris en compte par le langage QoSL4BP et la plateforme ORQOS sont de trois types :

- Jeton de sécurité de type « UserNameToken » : La mise en oeuvre de ce protocole consiste à insérer dans l'entête d'un message SOAP une balise contenant un jeton constitué d'une partie « nom d'utilisateur » et d'une autre partie « mot de passe » afin de permettre l'authentification de la source du message.
- Cryptage : La mise en oeuvre de ce protocole consiste à crypter ou bien décrypter le corps des messages SOAP à l'aide d'algorithmes de cryptage (tels que le triple DES ou bien RSA) utilisant une clé de cryptage, et permettant ainsi de garantir la confidentialité du contenu du message.
- Signature : les signatures servent à garantir l'intégrité du contenu d'un message SOAP et d'assurer au destinataire que ce contenu n'a pas changé en cours de route. Lorsqu'une signature est appliquée au contenu, une transformation utilise les données et balises du message SOAP pour créer une signature unique. Lors de la réception du message, le système client effectue une transformation de la signature XML, qui fait la distinction entre le contenu crypté avant signature et le contenu crypté après. Toutes les données cryptées après la signature sont décryptées, et l'intégrité des données est vérifiée en appliquant la même méthode de tranformation au contenu, par comparaison du résultat à la signature incluse dans le message SOAP.

En ce qui concerne la garantie de livraison, la norme WS-ReliableMessaging [IBM05] spécifiant divers protocoles est implémentée par l'API Sandesha [Apa06a]. Comme exhibé dans la figure 6.9, la première étape de ce protocole consiste tout d'abord à initialiser un contexte d'échange. Ensuite, la source envoie des messages augmentés des informations d'identification liées au contexte d'échange. Lorsque la destination reçoit ce message, elle garantit que celui-ci à bien été reçu en renvoyant un message d'acquittement à la source (dans le cas d'un protocole de type « EXACTLYONCE »), ou bien elle garantit que l'ordre dans lequel les messages ont été envoyés est bien respecté pour leur traitement à destination (dans le cas d'un protocole de type « INORDER »). Une fois les divers messages d'acquittement reçus, la source prévient la destination de mettre fin au contexte d'échange.

Gestion des contrats Pour gérer les contrats WS-Agreement avec le client et les fournisseurs de service, un sous-composant utilise l'implémentation du protocole WS-Agreement

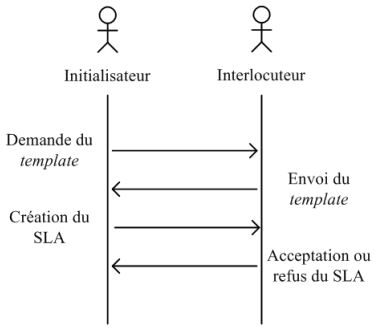

FIGURE 6.10: Protocole d'accord WS-Agreement

par le « WS-Agreement Framework » (WSAG4J) [SCA08]. Selon ce protocole, deux participants (un initiateur et un interlocuteur) dialoguent pour aboutir à un accord. La figure 6.10 exhibe ce protocole dont les étapes sont : la demande d'un *template* de la part de l'initiateur, la réponse de son interlocuteur, une proposition de SLA sur la base du *template* recueilli de la part de l'initiateur, et enfin l'acceptation ou le refus du SLA par son interlocuteur. Dans notre contexte, ce modèle se spécialise selon qu'il s'agit du client ou de l'architecte intégrateur qui initialise le protocole d'accord. Dans le cas où le client initialise l'échange avec l'architecte intégrateur, alors le client demande les offres de QdS que lui propose l'architecte intégrateur, puis, selon ses exigences, le client peut diminuer ou supprimer certaines des offres. Dans le cas où l'architecte intégrateur initialise l'échange avec un fournisseur de service, alors l'architecte intégrateur demande les offres de QdS que lui propose l'architecte intégrateur, puis, selon ses exigences, l'architecte intégrateur peut diminuer ou supprimer certaines des offres.

Monitoring La mise en oeuvre du *monitoring* s'effectue sur des activités simples *invoke* ou bien des activités composites. Dans le cas d'activités simples *invoke*, alors le *monitoring* utilise les deux événements liés aux messages transmis par les redirections du BPEL (*i.e* l'entrée et la sortie des messages SOAP). Dans le cas d'activités composites, alors le *monitoring* utilise les deux événements liés aux messages transmis par les indirections du BPEL (*i.e* les appels aux opérations « NotifyBefore » et « NotifyAfter »).

Le langage QoSL4BP et la plateforme ORQOS permettent d'observer trois propriétés de QdS de performance que sont le temps de réponse, le débit et la disponibilité. Dans le cas du temps de réponse, la mesure est effectuée en calculant la différence entre les deux moments des événements liés aux messages associés à l'activité observée. Dans le cas du débit, la mesure est effectuée en calculant d'une part le nombre d'occurrences du premier événement, et d'autre part le nombre d'occurences du second événement, sur une période de temps arbitraire. Les deux débits obtenus sont comparés : si le débit lié au premier événement est plus élevé que le débit lié au second événement, alors l'activité ne tient pas la charge et son débit effectif est le débit en sortie (*i.e* lié au second événement), tandis que dans le cas où les deux débits sont égaux, alors l'activité tient la charge et son débit effectif est considéré supérieur ou égal au débit en sortie. Enfin, dans le cas de la disponibilité, le taux de disponibilité d'une activité est calculée en mesurant, en pourcentage, le nombre d'occurences du second événement par rapport

au nombre d'occurences du premier événement. Ces traitements sont mis en oeuvre par des sondes en lien avec le registre « Etat du système ».

6.6 Bilan

A travers trois étapes de mise en oeuvre ainsi que notre plateforme ORQOS, ce chapitre a présenté le modèle d'exécution du langage QoSL4BP. Une particularité remarquable dans la mise en oeuvre de notre langage est qu'elle ne requiert aucune modification de la plateforme en charge de l'exécution de la composition de services.

1. La première étape consiste à identifier les services Web qui participeront à la composition lors de l'exécution. Cette étape s'effectue avant le déploiement de la composition et requiert en entrée le document BPEL de composition de services, les politiques QoSL4BP et la liste des services potentiels pour les partenaires de la composition ainsi que leur SLA. Tout d'abord, la composition est transcrite sous la forme d'un arbre portant les informations de QdS puis une stratégie de décomposition est ensuite pratiquée sur l'arbre afin d'obtenir éventuellement plusieurs sous-arbres de taille moindre. Alors, des stratégies de planification sont appliqués sur chacun des sous-arbres dans le but de sélectionner les services partenaires de la composition. A l'issue de l'étape de recherche, le processus échoue ou bien une liste de services partenaires à invoquer depuis la composition est retournée ainsi que, potentiellement, le SLA du service composite.
2. La seconde étape précède également le déploiement de la composition. Elle effectue une transformation de la composition de services en mettant en oeuvre deux traitements indépendants : d'une part des indirections sont tissées dans la composition de services, et d'autre part certains partenaires des activités *invoke* sont redirigés vers un intercepteur. Ces indirections et ces redirections ont pour rôle de permettre la mise en oeuvre des traitements dynamiques de QdS à l'exécution. Cette stratégie permet que la plateforme d'exécution de la composition et celle en charge des politiques soient indépendantes, et donc qu'elles puissent évoluer librement l'une de l'autre. A l'issue de cette étape de transformation, la composition de services modifiée peut être déployée sur un moteur BPEL.
3. La troisième étape se déroule à l'exécution de la composition. A ce stade les règles sont mises en oeuvre de façon synchrone avant et après certaines activités de la composition. A l'exécution, les événements et les informations issus des indirections et des redirections sont utilisés afin de mettre à jour la QdS du système et de pouvoir effectuer les traitements de QdS spécifiés par les règles.

Ce modèle d'exécution permet ainsi d'exécuter les préoccupations de gestion de QdS et de gestion de la composition, chacune encapsulée par un langage dédié (respectivement le langage QoSL4BP et le langage BPEL), par des plateformes non intrusives entre elles (respectivement la plateforme ORQOS et le moteur BPEL).

Chapitre 7

Développement de compositions de services avec ORQOS

Ce chapitre présente un scénario de mise en oeuvre complet pour la gestion de la QdS d'une composition de services avec notre approche. L'étude de ce scénario permet de réaliser une évaluation de notre approche en testant chacun des critères identifiés dans la section 3.1. Nous décrivons tout d'abord, dans la section 7.1, le scénario étudié. Le *workflow* et les exigences de gestion de QdS sont ensuite présentés dans la section 7.2. A la section 7.3, ces éléments sont projetés sous la forme de spécifications BPEL et QoSL4BP. Les trois étapes de mise en oeuvre avec notre approche sont déclinées dans la section 7.4. Enfin, la section 7.5 effectue une évaluation de notre approche, réalisée en tirant profit des exemples provenant du scénario. La section 7.6 conclut avec un bilan.

Sommaire

7.1 Introduction

L'objectif de ce chapitre est double. Il s'agit tout d'abord d'exhiber, à travers l'exemple d'un scénario, la mise en oeuvre complète de notre approche, de la spécification à l'exécution de ces spécifications. Le scénario « Dossier Médical Personnalisé »

(DMP) est un scénario inspiré du milieu médical. Le *dossier médical personnalisé* regroupe l'ensemble des données de santé liées à un patient. L'objectif du service DMP est de mettre à la disposition d'un membre du personnel médical certaines informations du dossier médical d'un patient relatives à une pathologie particulière de ce patient. Ainsi, quel que soit le centre hospitalier que fréquente le patient, le personnel soignant à accès à ses informations. Par ailleurs, pour des raisons de confidentialité, certaines informations sont limitées selon la catégorie et les droits du personnel soignant qui interroge le service DMP. Une fois accédées, les informations accessibles sont récupérées dans le dossier, archivées, puis envoyées au personnel soignant. Le scénario DMP est un scénario plus complet que le scénario UTP, présenté à la section 4.3.2, faisant intervenir un plus grand nombre de services et mettant en oeuvre d'autres exigences de QdS émises par l'architecte intégrateur.

Le deuxième objectif de ce chapitre consiste à réaliser une évaluation de notre approche en comparant les analyses du bilan de l'état de l'art avec l'analyse des capacités exhibées par notre approche. En particulier, pour réaliser cette comparaison, nous nous basons autant que possible sur les exemples de mise en oeuvre exhibés par le scénario DMP dans ce chapitre. Cette évaluation a pour objectif de valider notre approche.

7.2 Présentation du scénario « Dossier Médical Personnalisé »

7.2.1 Composition de services

La figure 7.1 exhibe le *workflow* associé au scénario DMP. Celui-ci fait intervenir sept services. Tout d'abord, deux services d'identification du patient ainsi que des informations liées à sa pathologie sont appelés. En parallèle, un service permet de récupérer le niveau d'autorisation que possède le personnel de santé pour ce patient et sa pathologie. Il en résulte une liste d'informations auxquelles le personnel de santé a accès. Chacune de ces informations permet d'accéder aux éléments du dossier qui sont récupérés un par un via un service d'accès aux informations médicales. Ce service est appelé tant qu'il reste des éléments dans la liste. Ensuite, ces éléments sont archivés par un service afin de créer un dossier formaté et compressé. Finalement, ce dossier est envoyé au personnel de santé via un service d'envoi par FTP ou par Mail.

7.2.2 Exigences liées à la gestion de la QdS

L'architecte intégrateur souhaite placer un certain nombre d'exigences de QdS sur ce scénario afin de gérer la QdS du *workflow* :

1. Tout d'abord, il souhaite que le service composite DMP fasse une offre de QdS à ses clients (temps de réponse inférieur à 200 ms, un débit supérieur à 100 requêtes par minute et une disponibilité supérieure à 90%). Afin de garantir cette offre, il souhaite appliquer des exigences légèrement plus élevées sur la composition de services DMP (temps de réponse inférieur à 180 ms, un débit supérieur à 110 requêtes par minute et une disponibilité supérieure à 92%). Cette stratégie consistant à surcontraindre les exigences par rapport à l'offre permet de pallier aux éventuelles défaillances du réseau (et donc de la QdS vue par le client),

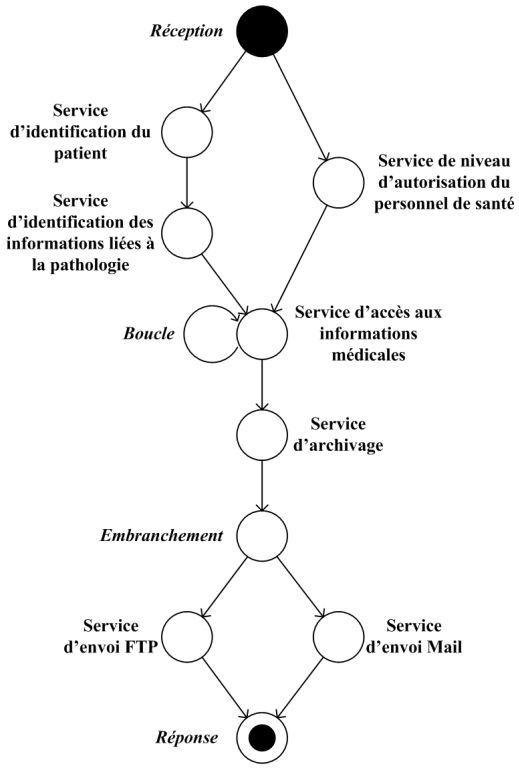

FIGURE 7.1: Composition « Dossier Médical Personnalisé »

ainsi qu'aux variations des paramètres tels que le nombre de boucles ou bien le taux d'utilisation des chemins empruntés à partir de l'embranchement. Par ailleurs, l'architecte intégrateur connait approximativement la valeur moyenne de ces paramètres et souhaite les initialiser à ces valeurs (5 boucles et 20% vs 80% pour les taux de l'embranchement). Si l'offre du service DMP était violée, il serait nécessaire d'effectuer une nouvelle planification des services. Dans le cas d'un échec, le client devrait être informé par une exception et l'architecte intégrateur devrait être notifié par une trace.

2. Les trois premiers services, appelés en parallèle, ont été repris d'une autre composition de services. La QdS de cette portion du *workflow* est connue et l'architecte intégrateur sait par expérience que cette zone peut garantir une certaine offre de QdS (temps de réponse inférieur à 80 ms, un débit supérieur à 150 requêtes par minute et une disponibilité supérieure à 98%). Il souhaite donc isoler cette partie du *workflow* de manière à gérer sa QdS indépendamment du reste de la composition de services DMP. En particulier, si l'offre de QdS de cette partie était violée, il serait nécessaire d'effectuer une nouvelle planification des services. Dans le cas d'un échec, l'architecte intégrateur devrait être notifié par une trace.
3. Le *partnerLink* du service d'archivage possède trois services auxquels il peut être rattaché. Cependant aucun de ces services ne possède de SLA. Par conséquent, l'architecte intégrateur souhaite effectuer lui-même le lien entre ce *partnerLink* et les services potentiellement utilisables. Par expérience, il sait que ces trois services possèdent des capacités différentes pour gérer les débits de requêtes. Il souhaite

donc faire varier le service à utiliser pour ce *partnerLink* selon le débit entrant de requêtes en direction du service d'archivage. Il sait qu'en rationalisant l'utilisation de ces services, il pourrait garantir pour cette activité une offre de QdS (temps de réponse inférieur à 30 ms, un débit supérieur à 120 requêtes par minute et une disponibilité supérieure à 98%).

4. Les services d'accès aux informations médicales requièrent des informations d'authentification avant de garantir l'identité de l'organisme souhaitant accéder à ces informations. Puisque les appels sont effectués depuis la composition de services DMP, alors celle-ci doit fournir un jeton d'authentification particulier (nom d'utilisateur « dmp_usr » et mot de passe « foo ») afin de pouvoir accéder à ces services. En cas d'échec, un mécanisme doit permettre de tracer l'erreur et de rejeter une exception vers l'utilisateur.
5. Afin de garantir la confidentialité des données qui sont transférées aux services en charge d'envoyer le dossier médical, l'architecte intégrateur souhaite que les données transmises à ces services soient cryptées selon un algorithme de type « 3DES » avec comme clé de cryptage « KeY5 ». En cas d'échec, un mécanisme doit permettre de tracer l'erreur et de rejeter une exception vers l'utilisateur. Par ailleurs, il est possible que ces services offrent des mécanismes de garantie de livraison. Si leur SLA contient de telles offres, alors l'architecte intégrateur souhaite mettre en oeuvre le mécanisme associé.

7.3 Développement de la composition DMP

7.3.1 Composition BPEL

Le listing 7.1 fait apparaître la projection du *workflow* présenté dans la section 7.2.1 sous la forme d'un document BPEL. Ce document contient les appels aux services décrits dans la figure 7.1 ainsi que des activités composites permettant de structurer le *workflow*. En particulier, l'activité *flow* « registry » contient les appels aux trois services d'identification et d'autorisation, tandis que l'activité *while* « FetchLoop » boucle sur l'appel au service d'accès aux informations médicales et que l'activité *switch* « SendReport » effectue un embranchement selon le mode d'envoi sélectionné par le client. L'activité *sequence* « DMPComposition » contient l'intégralité de la composition.

```
<process name="dmpserviceProcess" ..>
  <partnerLinks>
    <partnerLink name="PatientService" ../>
    <partnerLink name="PathologyService" ../>
    ..
  </partnerLinks>

  <variables>..</variables>

  <sequence name="DMPComposition">
    <receive ../>

    <flow name="Registry">
      <sequence>
        <invoke name="IdentifyPatient" partnerLink="PatientService" ../>
        <invoke name="IdentifyPathology" partnerLink="PathologyService" ../>
      </sequence>
      <invoke name="AuthorizationMedic" partnerLink="MedicService" ../>
    </flow>

    <while name="FetchLoop">
      <invoke name="Fetcher" partnerLink="FetcherService" ../>
```

```
    </while>

    <invoke name="Archiver" partnerLink="ArchiverService" ../>

    <switch name="SendReport">
      <case condition="condition1">
        <invoke name="SendFTP" partnerLink="FTPService" ../>
      </case>
      <case condition="condition1">
        <invoke name="SendMail" partnerLink="MailService" ../>
      </case>
    </switch>

    <reply ../>

  </sequence>
</process>
```

Listing 7.1: Code BPEL lié à la composition de services DMP

7.3.2 Politiques QoSL4BP

Les listings de cette section correspondent respectivement à la projection des spécifications de QdS émises par l'architecte intégrateur à la section 7.2.2 vers des politiques QoSL4BP.

1. Politique « set_global_qos » (listing 7.2) : cette politique définit dans sa partie statique les offres, les exigences et les paramètres définis par l'architecte intégrateur pour le *SCOPE* correspondant au *workflow* entier. Il souhaite qu'une stratégie de recherche de type *back tracking* soit mise en oeuvre pour la recherche des services. La section *RULES* précise la réaction liée à la violation de l'offre faite au client : une nouvelle planification des services doit avoir lieu. En cas d'échec de la planification, une trace et une exception sont effectuées.

```
POLICY "set_global_qos" = {
  SCOPE = { PROCESS }

  INIT = {
    SET_OFFER(([RESPONSETIME<200],
               [THROUGHPUT>100],
               [AVAILABILITY>90])) ;
    SET_REQUIREMENT(([RESPONSETIME<180],
                     [THROUGHPUT>110],
                     [AVAILABILITY>92])) ;
    SET_LOOP(WHILE["FetchLoop"]) = 5 ;
    SET_RATE(SWITCH["SendReport"], "condition1") = 20 ;
    SET_RATE(SWITCH["SendReport"], "condition2") = 80 ;
    PLANNING(JOIN_ACTIVITY, BACK_TRACKING) ;
  }

  RULES = {
    VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
      PLANNING(JOIN_ACTIVITY, BACK_TRACKING) ;
    CATCH("PLANNING EXCEPTION") -> {
      LOG("Echec de la planification globale.") ;
      THROW("La QdS du service DMP n'est plus maintenue.") ;
    }
  }
}
```

Listing 7.2: Politique set_global_qos

2. Politique « manage_registry_qos » (listing 7.3) : cette politique définit dans sa partie statique les offres et les exigences définies par l'architecte intégrateur pour le *SCOPE* correspondant à l'activité composite *flow* « registry ». Il souhaite qu'une stratégie de recherche de type *constraint programming* soit mise en oeuvre pour la recherche des services. La section *RULES* précise la réaction liée à la

violation de l'offre de cette activité composite : une nouvelle planification des services doit avoir lieu. En cas d'échec de la planification, une trace et une exception sont effectuées.

```
POLICY "manage_registry_qos" = {
  SCOPE = { FLOW["Registry"] }

  INIT = {
    SET_OFFER(([RESPONSETIME<80],
               [THROUGHPUT>150],
               [AVAILABILITY>98])) ;
    SET_REQUIREMENT(([RESPONSETIME<80],
                     [THROUGHPUT>150],
                     [AVAILABILITY>98])) ;
    PLANNING(JOIN_ACTIVITY, CP) ;
  }

  RULES = {
    VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
      PLANNING(JOIN_ACTIVITY, CP);
    CATCH("PLANNING␣EXCEPTION") ->
      LOG("Echec␣de␣la␣planification␣de␣l␣activite␣composite␣registry.") ;
      THROW("La␣QdS␣du␣service␣DMP␣n'est␣plus␣maintenue.") ;
    }
  }
}
```

Listing 7.3: Politique manage_registry_qos

3. Politique « archiver_binding » (listing 7.4) : cette politique définit dans sa partie statique les offres définies par l'architecte intégrateur pour le *SCOPE* correspondant à l'activité simple *invoke* « Archiver ». Par ailleurs, il précise qu'à l'initialisation le service « RARarchive » doit être utilisé comme *partnerLink* de cette activité. La section *RULES* met en oeuvre la politique de choix *partnerLink* à l'exécution selon le débit entrant de requêtes en direction de l'activité *invoke* « Archiver ». En cas de violation de l'offre de cette activité, une trace et une exception sont effectuées.

```
POLICY "archiver_binding" = {
  SCOPE = { INVOKE["Archiver"] }

  INIT = {
    SET_OFFER(([RESPONSETIME<30],
               [THROUGHPUT>120],
               [AVAILABILITY>98]));
    BIND(JOIN_ACTIVITY, "RARarchive") ;
  }

  RULES = {
    SENSOR(JOIN_ACTIVITY, THROUGHPUT) < 40 ->
      BIND(JOIN_ACTIVITY, "ZIParchive") ;
    (SENSOR(JOIN_ACTIVITY, THROUGHPUT) >= 40 AND SENSOR(JOIN_ACTIVITY,
      THROUGHPUT) < 80) -> BIND(JOIN_ACTIVITY, "RARarchive") ;
    SENSOR(JOIN_ACTIVITY, THROUGHPUT) >= 80 ->
      BIND(JOIN_ACTIVITY, "TGZarchive") ;

    VIOLATION_ACTIVITY(JOIN_ACTIVITY) ->
      LOG("Echec␣de␣la␣QdS␣du␣service␣Archiver.") ;
      THROW("La␣QdS␣du␣service␣DMP␣n'est␣plus␣maintenue.") ;
  }
}
```

Listing 7.4: Politique archiver_binding

4. Politique « authentication » (listing 7.5) : cette politique applique un mécanisme de type jeton de sécurité sur les activités *flow* « Registry » et *invoke* « Fetcher ». En cas d'échec du mécanisme de sécurité, une trace et une exception sont effectuées.

```
POLICY "authentication" = {
```

```
    SCOPE = { (FLOW["Registry"] OR INVOKE["Fetcher"]) }

    INIT = { SET_REQUIREMENT([SECURITY:TOKEN]) ; }

    RULES = {
      PERFORM_ACTIVITY(JOIN_ACTIVITY, [SECURITY:TOKEN], ("dmp_usr", "foo"));
      CATCH("SECURITY␣EXCEPTION") -> {
        LOG("Echec␣a␣l'autentification.") ;
        THROW("Le␣service␣DMP␣a␣echoue.") ;
      }
    }
  }
```

Listing 7.5: Politique authentication

5. Politique « encryption_and_reliable » (listing 7.6) : cette politique effectue un mécanisme de type cryptage sur l'activité *invoke* « Archiver » et sur toutes les activités *invoke* de l'activité composite « SendReport ». En cas d'échec du mécanisme de sécurité, une trace et une exception sont effectuées. Par ailleurs, pour chacune de ces activités *invoke*, le contrat de service établi avec le partenaire de l'activité *invoke* est lu afin de déterminer si le service offre un mécanisme de garantie de livraison de type « EXACTLYONCE ». Si c'est le cas, alors ce mécanisme est mis en oeuvre.

```
  POLICY "encryption_and_reliable" = {
    SCOPE = {
      (INVOKE[*] IN SWITCH["SendReport"])
    }

    INIT = { SET_REQUIREMENT([SECURITY:ENCRYPTION]) ; }

    RULES = {
      PERFORM_PROVIDER(JOIN_ACTIVITY, [SECURITY:ENCRYPTION], ("3DES","KeY5"));
      CATCH("SECURITY␣EXCEPTION") -> {
        LOG("Echec␣du␣cryptage.") ;
        THROW("Le␣service␣DMP␣a␣echoue.") ;
      }

      READ_PROVIDER(JOIN_ACTIVITY, [RELIABILITY:EXACTLYONCE]) ->
        PERFORM_PROVIDER(JOIN_ACTIVITY, [RELIABILITY:EXACTLYONCE]) ;
    }
  }
```

Listing 7.6: Politique encryption_and_reliable

7.3.3 Projet de composition

Un document « Projet de composition » est un document permettant de référencer l'ensemble des politiques QoSL4BP devant être intégrées à une composition de services ainsi que l'ensemble des services candidats, et de leur fournisseur potentiel, pouvant être utilisés pour chacun des *partnerLinks* de la composition. Ce document est exécuté par la plateforme ORQOS afin d'initialiser les étapes de mise en oeuvre des politiques QoSL4BP.

Le listing 7.7 fait apparaître le document de « Projet de composition » du scénario DMP. Celui-ci est constitué de trois sections. La première section permet de référencer la composition de services ciblée ainsi que l'adresse de service composite. La seconde section du document projet de composition contient des références pointant vers les documents de politique QoSL4BP qui doivent être pris en compte par la composition. Enfin la troisième section du document contient une liste de liens entre les *partnerLinks* de la composition et les services Web qui sont en mesure de les implémenter. Par ailleurs, pour chaque service Web, il peut exister une adresse vers un fournisseur de service permettant d'établir un contrat de service.

```
<project name="DMPProject">

  <composition name="UTPservice">
    <bpel>dmpservice.bpel</bpel>
    <url>http://localhost:8080/active-bpel/services/DMPService</bpel>
  </composition>

  <policies>
    <policy name="set_gobal_qos">global.qosl</policy>
    <policy name="manage_registry_qos">registry.qosl</policy>
    <policy name="archiver_binding">archiver.qosl</policy>
    <policy name="secure">secure.qosl</policy>
    <policy name="encryption_and_reliable">mechanisms.qosl</policy>
  </policies>

  <binding>
    <partnerlink name="PatientService">
      <service name="patient1">
        <wsdl>http://localhost:8080/active-bpel/s/patient1</wsdl>
        <provider>http://localhost:8080/active-bpel/p/patient1Provider</provider>
      </service>
      <service name="patient2">
        <wsdl>http://localhost:8080/active-bpel/s/patient2</wsdl>
        <provider>http://localhost:8080/active-bpel/p/patient2Provider</provider>
      </service>
      ..
    </partnerlink>
    <partnerlink name="PathologyService">
      ..
    </partnerlink>
    ..
  </binding>

</project>
```

Listing 7.7: Document Projet de composition du scénario DMP

7.4 Etapes de mise en oeuvre de la composition DMP

7.4.1 Etape de mise en oeuvre statique

Présentée dans la section 6.2, cette étape consiste à traiter l'ensemble des spécifications statiques des politiques QoSL4BP afin de sélectionner les services dont les offres de QdS répondent aux diverses exigences spécifiées par l'architecte intégrateur.

Tout d'abord, l'arbre de composition, exhibé dans la figure 7.2, est généré à partir du *workflow* BPEL décrit dans le listing 7.1.

Cet arbre est ensuite décomposé en autant de sous-arbres qu'il existe de politiques contenant des offres de QdS. Dans le cas du scénario DMP, trois politiques comportent des offres de QdS : les politiques « set_global_qos », « manage_registry_qos » et « archiver_binding ». Les figures 7.3, 7.4 et 7.5 exhibent respectivement ces sous-arbres ainsi que la projection des informations de QdS spécifiées dans les sections statiques des cinq politiques définies à la section 7.3.1. Le document « Projet de composition » est également utilisé afin de renseigner les services qu'il est possible d'appeler pour chaque activités *invoke* des sous-arbres. Par ailleurs, les fournisseurs de ces services sont appelés afin de récupérer leurs offres.

Une fois ces sous-arbres construits, des stratégies de planification de services spécifiées dans les politiques « set_global_qos » et « manage_registry_qos » sont mises en oeuvre. La politique « archiver_binding » n'effectue pas de planification puisque cette politique spécifie explicitement le service à utiliser pour l'activité *invoke* « Archiver ». Etant donné que les sous-arbres issus des politiques « set_global_qos » et « manage_registry_qos » contiennent chacun trois services, alors la complexité maxi-

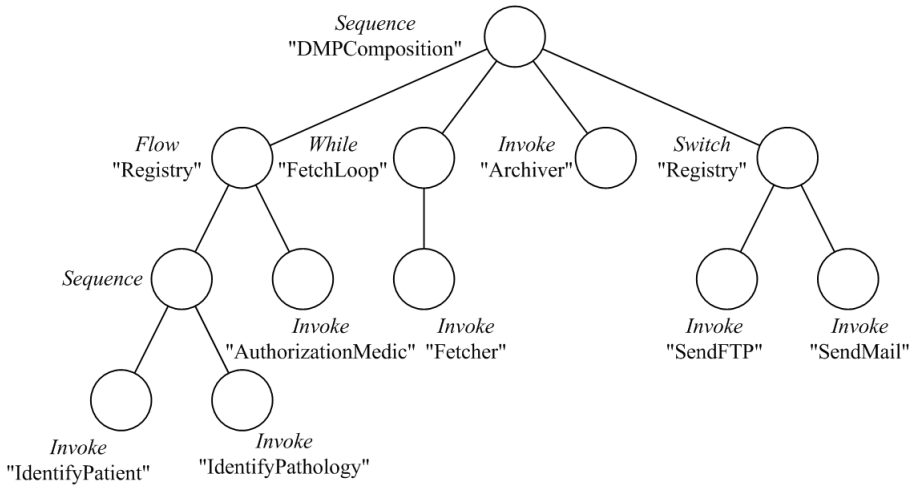

FIGURE 7.2: Arbre de la composition « DMP »

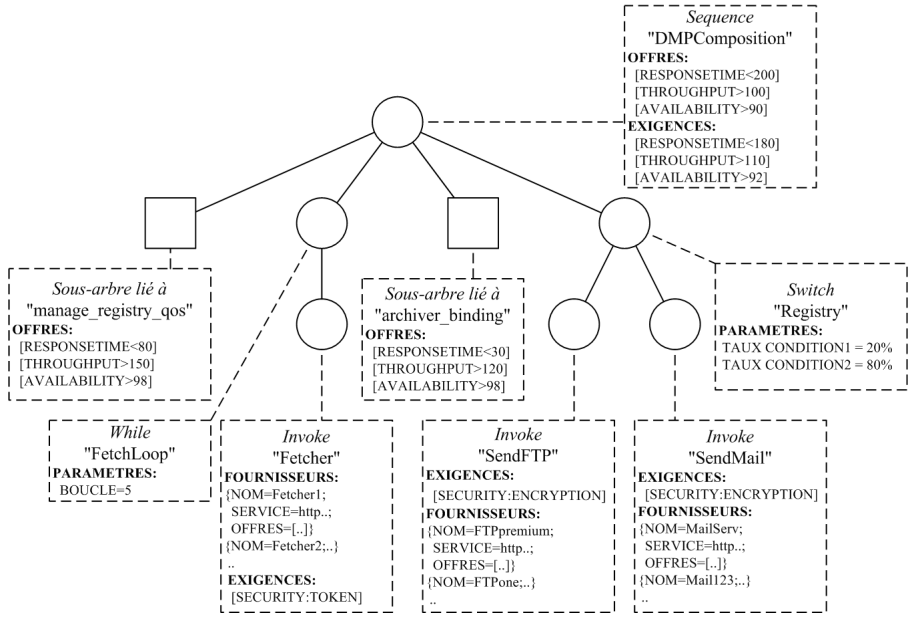

FIGURE 7.3: Sous-arbre issu de la politique « set_global_qos »

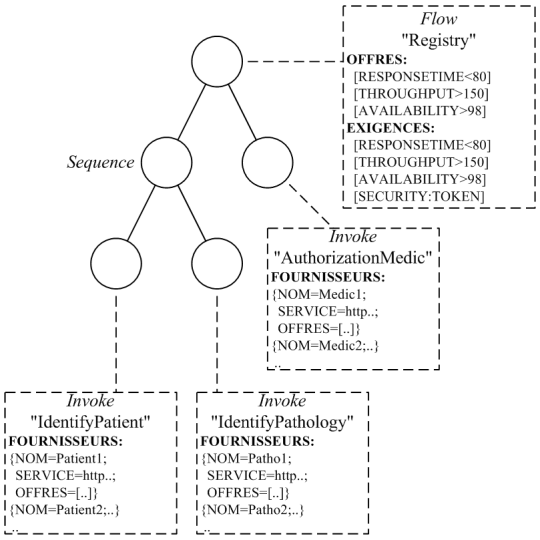

FIGURE 7.4: Sous-arbre issu de la politique « manage_registry_qos »

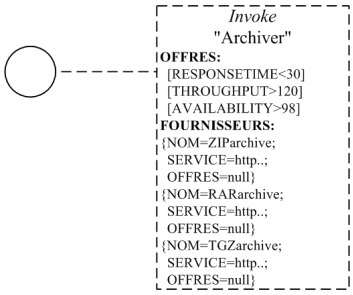

FIGURE 7.5: Sous-arbre issu de la politique« archiver_binding »

male de recherche pour chacun de ces sous-arbres est en $\Theta(s^3)$ où s est le nombre de différents services pouvant implémenter chaque partenaire des activités de la composition.

7.4.2 Etape de transformation

Présentée à la section 6.3, l'étape de transformation de la composition de services permet d'introduire des indirections vers les politiques QoSL4BP. La figure 7.6 exhibe la composition de services DMP tranformée. Pour chaque politique, les activités de jonction du *SCOPE* sont identifiées et des indirections sont insérées autour de ces activités. En ce qui concerne la politique « set_global_qos » (listing 7.2), étant donné que cette politique porte sur l'ensemble de la composition, sa mise en oeuvre s'effectue lors de l'interception des messages échangés entre les clients et la composition de services. Par ailleurs, l'ensemble des services de la composition DMP est redirigé vers l'intercepteur de la plateforme ORQOS.

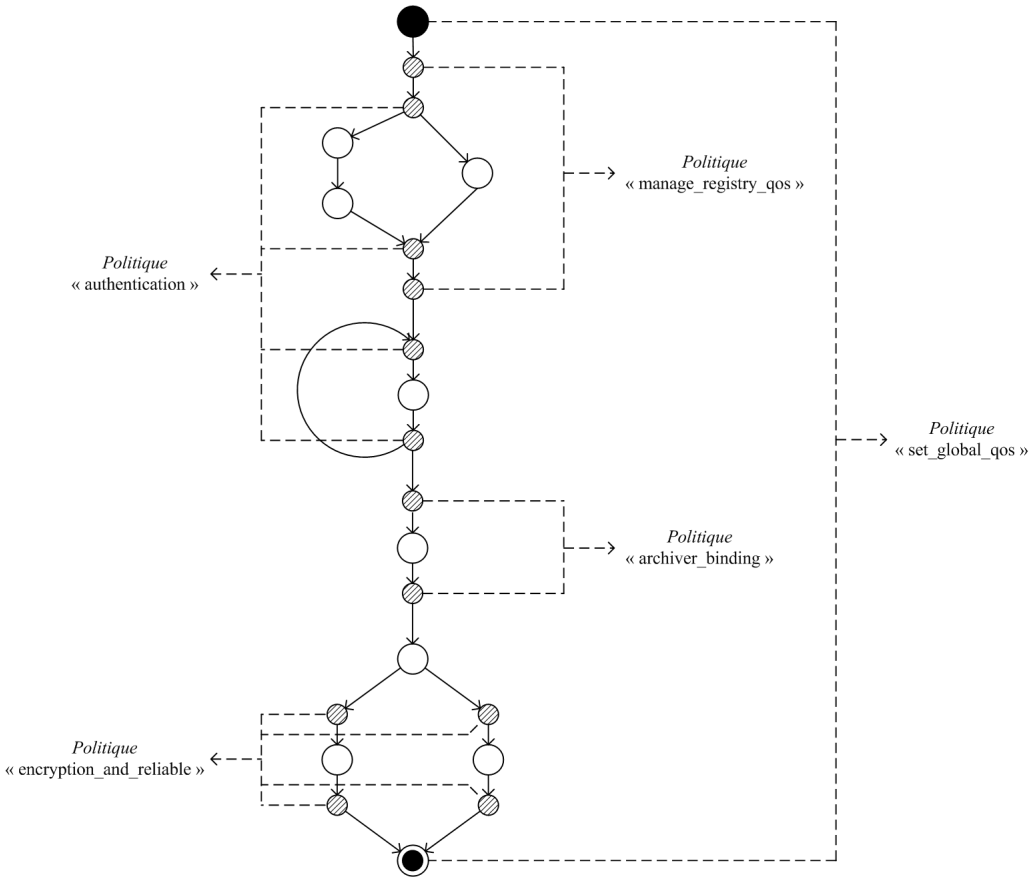

FIGURE 7.6: Tissage des indirections dans la composition de service DMP

7.4.3 Etape de mise en oeuvre dynamique

Une fois l'étape de transformation du document BPEL effectuée, il est alors possible de déployer la composition de services sur un moteur BPEL. Ce processus reste identique au processus de déploiement spécifique au moteur BPEL utilisé. La participation d'ORQOS n'est pas requise lors du déploiement, les politiques ayant déjà été chargée lors de l'étape statique de mise en oeuvre des politiques, initiée par le traitement du document « Projet de composition ».

Lors de l'exécution de la composition, les sections *RULES* des politiques sont activées par appels à la plateforme ORQOS. Nous illustrons la mise en oeuvre dynamique des politiques en déroulant le cas d'utilisation de la politique « manage_registry_qos ». La figure 7.7 illustre le traitement lié à la mise en oeuvre de la section *RULES* de cette politique. Le gestionnaire d'événements d'ORQOS est appelé avant et après l'exécution de l'activité de jonction *flow* « Registry ». Lors de ces appels, le gestionnaire déclenche les actions de *monitoring* liées à l'activité de jonction de manière à mettre à jour l'état de QdS du système. Les règles sont ensuite exécutées. Il s'agit de vérifier si les offres de QdS placées sur cette activité n'ont pas été violées. Dans le cas de l'appel effectué vers ORQOS après l'exécution de l'activité *flow* « Registry », nous supposons que le temps de réponse de l'activité a été suffisamment élevé pour que l'offre de QdS liée au

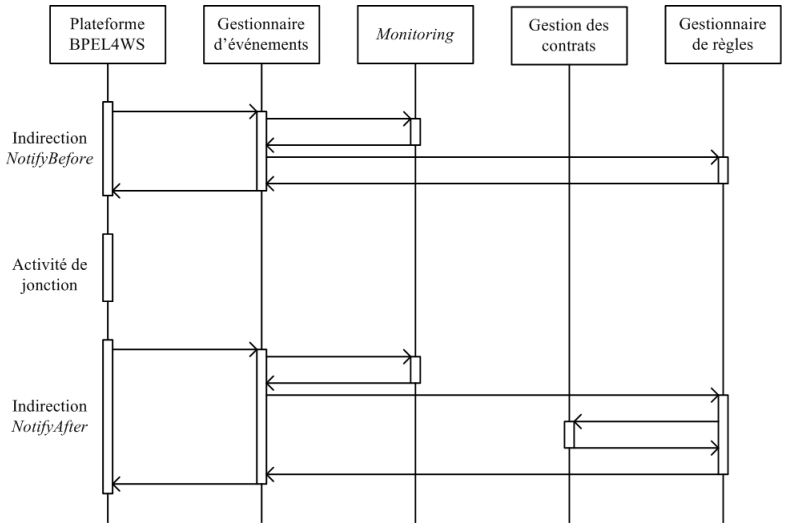

FIGURE 7.7: Mise en oeuvre de la politique « manage_registry_qos »

temps de réponse (inférieur à 80 ms) soit violée. Dans ce cas, lors de l'exécution des règles, l'activité de replanification est exécutée en appelant le composant en charge de la gestion des contrats. Celui-ci supprime l'ensemble des services actuels utilisés par les activités *invoke* appartenant au *flow* « Registry » et effectue une nouvelle recherche de services potentiels dont l'agrégation des offres de QdS puisse satisfaire aux exigences placées sur l'activité *flow* « Registry » lors de l'étape statique. Soit une autre configuration est trouvée et lorsque le gestionnaire de règles reprend l'exécution des règles de la politique « manage_registry_qos » il n'effectue pas d'action lié à la gestion d'exception, soit aucune configuration n'a été trouvée et alors le gestionnaire de règles a été alerté d'une exception de type « PLANNING EXCEPTION ». Dans ce cas, il reprend l'exécution des règles et la seconde règle est activée : une trace est effectuée puis une exception est rejetée vers la composition BPEL via le gestionnaire d'événement.

Pour l'exécution des compositions de services, nous avons utilisé le moteur d'orchestration ActiveBPEL avec un serveur TOMCAT. Pour les services Web applicatifs, nous avons utilisé la plateforme Axis. En déployant notre plateforme ORQOS, développée en Java, sur la même machine que le moteur ActiveBPEL, nous avons pu vérifier que l'exécution des indirections et des redirections peut être considérée comme négligeable par rapport aux coûts d'exécution des services applicatifs, déployés sur d'autres machines.

7.5 Evaluation

Pour chacun des critères mis en évidence lors de l'analyse des travaux connexes à la thèse et présentés à la section 3.1, nous effectuons une analyse de notre approche instanciée par l'exemple du scénario DMP.

Réutilisation de l'existant Le premier critère est celui de la réutilisation des plateformes et langages déjà existants. Lors du bilan de l'analyse des travaux déjà existants, nous avions pu remarquer que globalement les approches avaient convergé vers l'utili-

sation du langage BPEL pour la définition des compositions de services mais que, en revanche, une minorité de solutions faisaient explicitement référence aux travaux autour des SLAs et des WS-Policy. Notre approche met à profit les efforts déjà investis dans ces divers langages. Le langage BPEL est réutilisé pour la spécification de la composition de services, mais également pour affecter des traitements de gestion de QdS à certaines parties du *workflow* en désignant directement les activités BPEL ciblées. Par exemple, la politique « authentication » (listing 7.5) fait explicitement référence aux activités *flow* « Registry » et *invoke* « Fetcher » pour cibler la mise en oeuvre des traitements statiques et dynamiques. Par ailleurs, les SLAs (et les WS-Policy) sont également mis en oeuvre par notre approche et certains de leurs concepts sont intégrés dans le langage QoSL4BP. Ainsi, dans la politique « set_global_qos » (listing 7.2), l'architecte intégrateur effectue une proposition d'offres qui seront utilisés pour la création du SLA établi avec le client. Les mécanismes de sécurité et de garantie de livraison précisés dans les WS-Policy sont également lisibles dans les SLAs et peuvent être mis en oeuvre comme l'exhibe le code de la politique « encryption_and_reliable » (listing 7.6).

Séparation des préoccupations Lors de notre bilan sur les approches existantes, nous avions constaté un manque de clarté quant au domaine exact des préoccupations que ces approches permettent de capturer. Par ailleurs, les plateformes de mise en oeuvre requièrent des modifications du moteur d'exécution des compositions dans la majorité des cas, ce qui limite la réutilisation des outils développés autour du langage BPEL. Avec l'introduction du rôle d'architecte intégrateur, clairement isolé du rôle d'architecte de composition, notre approche introduit une meilleure séparation des préoccupations. En particulier, il est impossible pour chacun de ces deux rôles d'adresser ou d'interférer avec les préoccupations de l'autre. Par exemple, il n'est pas possible pour l'architecte intégrateur d'écrire des spécifications permettant de faire varier la structure de la composition de services, et inversement l'architecte de composition ne peut pas effectuer de *monitoring* ni mettre en place des mécanismes de sécurité dans la composition de services. De plus, le document Projet de composition permet de réaliser le lien entre les partenaires des activités BPEL de la composition, les services spécifiables avec QoSL4BP et les fournisseurs de services appelés par ORQOS. Cette stratégie permet de garantir l'isolation des préoccupations à la phase de spécification. Par ailleurs, lors de la mise en oeuvre, la transformation de la composition de services par ajout d'indirections et de redirections des appels vers la plateforme ORQOS permet d'exécuter les politiques de QdS aux moments adéquats sans transformer la plateforme BPEL. Cette stratégie de couplage lâche permet de garder chaque plateforme clairement isolée l'une de l'autre.

Couverture de gestion de la QdS A la différence des travaux de l'état de l'art, notre approche permet d'adresser des mécanismes qui ne se limitent pas soit à la gestion de la performance des services ou bien à la mise en oeuvre des normes WS-*. Il est ainsi possible de lire des contrats de service, d'effectuer des replanifications de service ou bien de mettre en oeuvre des mécanismes de type sécurité ou garantie de livraison. Ainsi, la politique « set_global_qos » (listing 7.2) permet de mettre en oeuvre la replanification des services de la composition, tandis que la politique « authentication » (listing 7.5) précise des informations pour la mise en oeuvre d'un mécanisme d'authentification sur

certaines parties de la composition. Cependant, notre plateforme ne propose pas de mécanismes de recherche de services aussi élaborés que dans les travaux tels que [Jae07] (présenté à la section 3.3.4). En ce qui concerne l'étendu des caractéristiques de QdS et des mécanismes qu'il est possible de mettre en oeuvre avec notre approche dans sa version actuelle, notre langage et sa mise en oeuvre permettent de gérer quatre caractéristiques de performance souvent exploitées dans les autres travaux de recherche (temps de réponse, débit, disponibilité et coût) ainsi que les mécanismes WS-* usuels (sécurité, garantie de livraison). Il semble toutefois envisageable d'étendre les traitements de notre approche à d'autres caractéristiques, mécanismes et métriques. Enfin, notre approche possède la particularité de permettre aux mécanismes liés à la QdS de cibler des parties de la composition avec une granularité variable. Par exemple, la politique « authentication » (listing 7.5) permet d'associer des exigences et des traitements de QdS sur les activités *flow* « Registry » et *invoke* « Fetcher »qui sont respectivement des activités simple et composite.

Dynamicité En ce qui concerne le critère de dynamicité, nous avions constaté que très peu de travaux considéraient à la fois le traitement des étapes statique et dynamique. Notre approche met clairement en évidence ces deux moments de prise en compte des traitements à travers, d'une part, les sections *INIT* et *RULES* du langage QoSL4BP, ainsi que, d'autre part, le processus de mise en oeuvre ORQOS qui a lieu à la fois au pré-déploiement et à l'exécution. Par exemple, la politique « archiver_binding » (listing 7.4) permet de sélectionner le service « RARarchive » à utiliser pour l'activité *invoke* « Archiver » lors du déploiement de la composition. Puis lors de l'exécution, l'évolution de la QdS du système fait varier le choix du service à utiliser selon les règles de la section *RULES* de cette politique. Certaines plateformes existantes, telles que AO4BPEL [CSHM06], eFlow et WS-Binder [CPE+06], permettent de modifier les spécifications à l'exécution. En cas de modification des spécifications, notre approche requiert de réitérer l'ensemble des étapes du processus de mise en oeuvre des politiques.

Expressivité Une remarque générale effectuée à l'occasion de l'étude des approches existantes était que l'expressivité des interfaces proposées aux utilisateurs afin de spécifier leurs traitements de QdS étaient soit trop restrictives (Trap/BPEL [ES06]), soit trop complexes (MASC [TEM07]). Le langage QoSL4BP propose une vingtaine de primitives réparties dans trois domaines, qui permettent d'adresser un domaine varié, homogène et consistant de la gestion de la QdS. En quelques instructions, il est possible de d'implémenter les spécifications de QdS de la composition DMP. Par ailleurs, son expressivité basée sur les objectifs et les règles permettent d'exprimer avec quelques instructions des mécanismes potentiellement complexes. Par exemple, dans le cas de la politique « manage_registry_qos » (listing 7.3), la section *RULES* spécifie en une seule règle et deux primitives que si les offres de l'activité de jonction sont violées, alors une replanification doit être effectuée en faisant intervenir un mécanisme de recherche par programmation par contraintes. Par ailleurs, le langage QoSL4BP permet d'adresser la gestion des accords, l'observation de QdS avec manipulation de paramètres tels que le nombre de boucles des activités *while*, et les mécanismes WS-* ce qui lui confère une richesse relative. La déclaration d'objectifs et de règles permet un spécification aisée. En revanche, le langage QoSL4BP ne permet pas de définir des variables

ou des instructions de contrôle de l'exécution de la politique comme le propose par exemple WS-CoL [BGP05] présenté à la section 3.2.2. Cependant, du fait que le langage ne permet pas de manipuler de tels concepts, il est plus facile de faire évoluer les spécifications rédigées dans les politiques. Ainsi, par exemple, dans la politique « authentication » (listing 7.5), si l'architecte intégrateur veut changer le jeton de sécurité pour l'authentification des appels aux services du *SCOPE* de cette politique, il lui suffit de modifier l'argument de l'instruction liée à la mise en oeuvre du mécanisme de jeton de sécurité.

7.6 Conclusion

Dans ce chapitre, nous avons montré comment le langage QoSL4BP et la plateforme ORQOS s'utilisent en pratique et nous avons évalué notre proposition à partir de quelques exemples d'un scénario de composition.

Dans un premier temps, nous avons présentés comment les besoins de l'architecte de composition et ceux de l'architecte intégrateur peuvent être spécifiés à l'aide des langages BPEL et QoSL4BP. Par ailleurs, nous avons présenté le document « Projet de composition » qui permet à ORQOS d'effectuer le lien entre le document BPEL, les politiques QoSL4BP à prendre en compte, et les services qui peuvent être appelés pour chaque partenaire des activités de la composition de services. Nous avons ensuite détaillé comment ces spécifications étaient exécutées lors des trois étapes du processus de mise en oeuvre de la plateforme ORQOS.

Nous avons ensuite effectué une évaluation de notre approche en nous basant sur les limites mises en évidence lors du bilan de l'état de l'art (section 3.4) et les exemples d'utilisation fournis par le scénario DMP. Pour chacun des cinq critères identifiés pour effectuer l'analyse des travaux existants, nous avons pu identifier les atouts ou les lacunes de notre approche. Ainsi, en ce qui concerne le critère de **réutilisation de l'existant**, notre approche permet de travailler avec le langage BPEL ainsi que les contrats SLA, sans les modifier. L'étude du critère de **séparation des préoccupations** montre que d'une part que les spécifications, avant pré-déploiement, sont isolées et que les plateformes coopèrent de manière non intrusive à l'exécution. La **couverture de gestion de la QdS** de notre approche a la particularité d'adresser à la fois les mécanismes liés à la gestion des caractéristiques de performance ainsi que des caractéristiques telles que la sécurité ou la garantie de livraison. Les travaux de l'état de l'art ont tendance à isoler ces mécanismes, limitant de fait leur prise en charge de la QdS dans les compositions de services. De plus, notre approche permet de cibler des activités BPEL complexes ou simples, ce qui permet de faire varier la granularité des traitements appliqués à la composition. Pour le critère de **dynamicité**, notre approche permet de spécifier des traitements statiques et dynamiques qui sont respectivement mis en oeuvre au pré-déploiement et à l'exécution de la composition de services. En particulier, ces traitements sont séparés en deux sections dans les politiques QoSL4BP. Enfin, en ce qui concerne le critère d'**expressivité**, le langage QoSL4BP vise, avec un nombre de primitives relativement restreint, à permettre à l'architecte intégrateur d'adresser des traitements variés et homogènes pour la gestion de la QdS. Sa structure, sous la forme d'objectifs et de règles, permet de décrire de façon déclarative les spécifications sans se préoccuper de spécifier les algorithmes de mise en oeuvre. Suf-

fisamment riche pour adresser des caractéristiques de performance et des mécanismes WS-*, il est également aisé de faire évoluer les spécifications du fait que le code écrit reste relativement court. En contrepartie, l'utilisation de QoSL4BP nécessite l'apprentissage d'un nouveau langage de la part de ses utilisateurs. De plus, le langage peut parfois souffrir d'un manque d'ouverture dans des cas précis d'utilisation (par exemple, la prise en compte de caractéristiques ou de mécanismes propres au domaine métier de la composition de services).

Chapitre 8

Conclusion et perspectives

Résumé des contributions

Dans cette thèse, nous nous sommes intéressés à la gestion de la QdS dans les compositions de services. En particulier, l'objectif de cette thèse est de fournir, d'une part, un moyen sûr, efficace et réutilisable pour spécifier la gestion de la QdS dans les compositions de services, et d'autre part des outils de mise en oeuvre statique et dynamique de ces spécifications. Nous avons tout d'abord noté qu'il existe des outils de prise en compte de la QdS au niveau des services unitaires (émergence des contrats de service, mécanismes WS-*) et que la spécification des compositions de services a, en particulier, convergé vers le langage BPEL. Parce que la QdS devient un facteur déterminant pour la sélection des services Web et que les compositions de services apportent une valeur ajoutée significative, il est donc intéressant de trouver un moyen de spécifier et de mettre en oeuvre la QdS dans ces compositions.

Lors de l'étude des approches déjà existantes, nous avons mis en évidence cinq critères pertinents pour l'étude de la gestion de la QdS dans les compositions. Tout d'abord pour le critère de **réutilisation**, les approches ne réutilisent pas assez les travaux effectués autour des SLAs. Par ailleurs, beaucoup de travaux insistent sur le principe de **séparation des préoccupations**, mais peu effectuent véritablement une étude des préoccupations et des rôles autour des compositions de services. De ce fait, bien que les spécifications soient physiquement isolées, le domaine des préoccupations capturées n'est pas suffisamment mis en évidence. En outre, les plateformes de mise en oeuvre requièrent des modifications du moteur d'exécution des compositions dans la majorité des cas, ce qui limite la réutilisation des outils développés autour du langage BPEL. La **couverture du domaine de QdS** prise en compte par les travaux existants est relativement inégale et aucun travail ne propose véritablement d'adresser à la fois la sélection des services et la mise en oeuvre de mécanismes WS-* tels que la gestion de la sécurité. Du point de vue de la **dynamicité**, nous avons pu constater que peu d'approches prenaient en charge à la fois la phase statique et la phase dynamique. Généralement les outils proposés ne s'adressent véritablement qu'à une seule des étapes de mise en oeuvre. Enfin pour ce qui est de l'**expressivité** des interfaces proposées aux utilisateurs afin de spécifier des traitements de QdS, souvent celle-ci est trop complexe ou limitée.

Par rapport aux approches déjà existantes dans l'état de l'art, nous nous sommes

orientés vers une solution mettant en valeur les particularités suivantes :

1. L'application du principe de séparation des préoccupations débouchant sur la mise en évidence de rôles aux enjeux clairement identifiables. En particulier, cette étude a permis d'isoler un ensemble de traitements et d'informations pertinents pour la gestion de la QdS, dans un rôle appelé « Architecte intégrateur ». Ce rôle, différent de celui qui rédige la composition de services, a pour tâche de donner les spécifications liées à la gestion statique et dynamique de la QdS de la composition. Pour cela, il possède une vue « boîte grise » de la composition. L'architecte intégrateur effectue la sélection des fournisseurs de service appropriés et communique les offres de QdS du service composite au client. Il veille au respect de ces offres et donc pour cela, il doit observer la QdS de la composition. Il doit également pouvoir mettre en oeuvre des mécanismes liés à la QdS (tels que la sécurité) sur des parties du *worklflow*. Enfin, le fait d'avoir une vue boite grise lui permet de gérer finement la QdS de sa composition : il peut ainsi spécifier des exigences de QdS plus fines sur certaines parties de sa composition, effectuer des plus-values (par exemple, pour dégager un bénéfice ou laisser une marge de sécurité pour gérer les performances de QdS), ou encore appliquer des stratégies de replanification à des niveaux intermédiaires (par exemple, sur des activités composites).
2. L'utilisation d'une approche langage dédié permettant de capturer les préoccupations du rôle d'architecte intégrateur. Ce langage, appelé QoSL4BP, est conçu comme un langage de politique offrant des idiomes de haut niveau encapsulant l'expertise liée à la gestion de la QdS. Le langage QoSL4BP fait apparaître, dans son modèle de données, les activités BPEL ainsi que les contraintes de QdS, tandis que ses primitives se décomposent en trois sous domaines que sont la gestion des accords, l'observation de la QdS et la gestion des mécanismes de QdS (sécurité, garantie de livraison, exception). Le langage QoSL4BP permet de structurer les instructions sous la forme de politiques comportant une cible (région d'application de la politique sur la composition de services), une section regroupant les traitements statiques mis en oeuvre au pré-déploiement de la composition et une section regroupant les traitements dynamiques mis en oeuvre à l'exécution de la composition. Par ailleurs, le langage QoSL4BP offre certaines propriétés intéressantes pour la gestion de la QdS. Tout d'abord, la mise en oeuvre du langage garantit la synchronisation des traitements entre gestion de QdS et exécution de la composition de services. Ensuite, le langage vérifie le typage des activités BPEL ciblées par les traitements de QdS, ainsi que la bonne prise en compte des exceptions pouvant être rejetées. Enfin, le langage garantit la finitude des traitements de QdS et exploite un ensemble de règles permettant une composition cohérente des politiques.
3. Une plateforme ORQOS pour la mise en oeuvre des politiques QoSL4BP. Cette plateforme est non intrusive vis à vis des plateformes BPEL déjà existantes, ce qui permet de réutiliser les travaux déjà effectués autour du langage BPEL. Son processus de mise en oeuvre est à la fois statique et dynamique. Il fait notamment intervenir un procédé de transformation de programme afin d'insérer dans le document BPEL, au pré-déploiement, des indirections vers l'exécution des politiques de QdS. Ainsi, le mélange des préoccupations (BPEL et politiques QoSL)

s'effectue avec selon un couplage lâche. A l'exécution, les traitements des politiques sont mis en oeuvre de façon synchrone avant et après les activités de la composition ciblées.

Nous avons pu mettre en application notre approche dans deux scenarii d'application que nous avons développés. Il s'agit du scénario « Urban Trip Planner » (UTP) appartenant au monde des télécommunications, ainsi que du « Dossier Médical Personnalisé » (DMP) appartenant au monde médical. Le cas du scénario UTP exhibe des exigences que pourrait soumettre un opérateur des télécommunications souhaitant mettre à disposition de ses clients un service pour téléphone mobile. Il s'agit d'un cas commercial dans lequel un opérateur des télécommunications cherche à assurer la fiabilité et la rentabilité de son service. Ce scénario a notamment été réutilisé dans le projet RNTL FAROS [PCW07]. L'exemple DMP pourrait être un exemple de composition de services implémentée par le Ministère de la santé dans l'objectif de permettre l'accès aux informations médicales de patients. Ce scénario s'intéresse davantage à des propriétés de type confidentialité et garantie de livraison, ainsi que des propriétés de type performance.

Perspectives

Bien que les principes de conception de notre approche langage dédié nous semblent pertinents et ne devraient pas être modifiés fondamentalement, il semble possible d'envisager des améliorations. Certaines de ces améliorations correspondent à des limitations de l'implémentation actuelle, d'autres à des extensions facilement intégrables dans le cadre actuel, et d'autres à des évolutions plus profondes. Ainsi, nous avons classé les perspectives sous la forme de perspectives à court terme et de perspectives à long terme.

Perspectives à court terme

Extension du langage QoSL4BP Comme nous l'évoquions dans la section 2.2.4, une difficulté majeure lors de la conception de langages dédiés consiste à délimiter l'expressivité du langage. En effet, un langage dédié se doit d'offrir une expressivité suffisamment importante, pour que l'utilisateur puisse adresser l'ensemble des préoccupations du domaine capturé, et cependant suffisamment limitée afin d'offrir un certain nombre de garanties. Dans le cas du langage QoSL4BP, une étude de domaine a été effectuée en préliminaire de son élaboration. Cependant, son expressivité est limitée à un certain nombre de traitements et de données qui sont susceptibles d'être étendus. Dans la version actuelle du langage, nous avons volontairement restreint ces traitements et ces données dans le but de simplifier et de clarifier notre approche.

Une piste d'amélioration des travaux de la thèse serait donc d'étudier les extensions potentielles du langage QoSL4BP afin que celui-ci offre à l'utilisateur une plus grande variété de traitements et de données. Par exemple, le modèle de SLA utilisé peut être plus complexe que celui que nous utilisons dans les travaux de thèse. Entre autre, les SLAs peuvent contenir des offres soumises à des hypothèses. Il est également envisageable de prendre en compte de nouvelles caractéristiques de QdS et de protocoles WS-*. Une autre piste pour l'extension des traitements capturés par le langage QoSL4BP serait d'encapsuler les différentes stratégies de sélection des services présen-

tées dans l'état de l'art (tel que l'algorithme génétique proposé par [CPEV05]), afin que l'utilisateur puisse invoquer les méthodes proposées par ces travaux à travers l'interface que nous lui mettons à disposition. D'un point de vue structure, le langage QoSL4BP pourrait également évoluer afin de proposer aux utilisateurs davantage de structures de contrôle (telles que la prise en compte d'alternatives dans les règles).

Ouverture de la plateforme ORQOS L'extension du langage QoSL4BP implique que la plateforme d'exécution correspondante, ORQOS, puisse mettre en oeuvre les nouvelles spécifications. Par exemple, ORQOS pourrait intégrer de nouveaux protocoles de négociation de SLA ou encore les algorithmes de recherche de services présentés dans l'état de l'art (section 3.3, notamment ceux présentés dans [Jae07]). Les multiples traitements liés aux nombreuses normes WS-* pourraient également être rajoutés. Etant donné que ces divers traitements sont susceptibles d'évoluer ou d'être étendues, il serait envisageable de repenser la plateforme ORQOS tel un *framework* dans lequel les traitements correspondants aux extensions du langage QoSL4BP pourraient être modifiés ou ajoutés.

Perspectives à long terme

Analyses statiques du langage QoSL4BP La version actuelle du langage QoSL4BP ne tire pas complètement profit de sa nature de langage dédié. L'un des avantages de l'utilisation d'un DSL au pouvoir d'expression volontairement limité est de permettre d'effectuer des analyses statiques plus poussées qu'un langage généraliste au moment de la compilation. Actuellement, le langage QoSL4BP offre des garanties telles que la finitude, la synchronisation ou la composition des politiques. Cependant, ces garanties proviennent essentiellement des techniques d'implémentation utilisées.

Une piste d'amélioration serait, par exemple, d'utiliser une approche de type « model checking » afin de tester l'exécution des règles dans toutes les situations possibles, au moment de leur compilation. Il serait ainsi possible, avant exécution, de vérifier si les règles ne vont pas boucler indéfiniment et donc violer la garantie de finitude du langage. En particulier, nous pourrions utiliser un moteur de règles afin de valider la stabilité des règles écrites par l'architecte intégrateur.

Formalisation des interactions entre politiques Dans le cas d'un langage modularisant une préoccupation sous la forme de modules isolés, comme c'est le cas pour les politiques QoSL4BP, la problématique de la composition entre les différents modules est relativement complexe. Dans la version actuelle des travaux, nous avons volontairement restreint cette étude par une stratégie d'interférence limitée (décrite à la section 5.5).

Une piste d'amélioration consisterait à effectuer une meilleure analyse des interférences entre politiques en procédant à la formalisation des divers traitements qu'il est possible de spécifier avec le langage QoSL4BP puis en modélisant les interactions entre ces traitements. Ainsi, il serait possible de faire émerger un modèle précis des interactions entre politiques. Il semblerait également possible d'ajouter une sémantique concernant la manière dont les politiques doivent être composées. Ainsi, les politiques spécifiées dans le document projet de composition pourraient être agrégées via des opérateurs spécifiques.

Un autre élément d'amélioration pourrait être la prise en compte des interférences entre instances de politiques. Dans la version actuelle de la mise en oeuvre du langage, nous faisons l'hypothèse que les diverses instances du *workflow* sont traitées en séquence. Sans cette hypothèse, les diverses instances de politiques peuvent être exécutées en parallèle. A l'instar des interférences entre politiques, les interférence entre instances de politiques constitue une problématique complexe. Elle requiert notamment d'étudier les mécanismes de synchronisation (tels que les sémaphores) devant être mis en oeuvre afin de garantir une bonne exécution des traitements de QdS et du *workflow*.

Réutilisation avec d'autres types d'architecture Dans ces travaux de thèse, nous nous sommes intéressés en particulier à la gestion de la QdS dans les compositions de services. Cependant, il pourrait être envisagé de gérer la QdS d'autres types d'architecture. Ainsi, nous pourrions envisager l'application des politiques QoSL4BP sur des langages modélisant des compositions de composants (langages de description d'architecture (ADL [1]), SCA [2], Fractal) ou encore sur des chorégraphies de services. Une piste d'amélioration serait donc d'étudier comment les travaux effectués dans cette thèse pourraient être projetés sur d'autres types d'architecture.

Réutilisation dans le cadre d'une approche dirigée par les modèles Dans le cadre d'une approche dirigée par les modèles, il est possible d'envisager un modèle de haut niveau décrivant des spécifications liées à la composition de services ainsi qu'à la gestion de la QdS. Une perspective d'étude de nos travaux serait la prise en compte de la transformation qui pourrait être effectuée sur de telles spécifications afin de générer la composition de services et les politiques QoSL4BP correspondantes.

1. Architecture Description Languages
2. Service Component Architecture

Bibliographie

[ACD+07] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web services agreement specification (ws-agreement), 2007. `https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.graap-wg/docman.root.published_documents.web_services_agreement_specifica/doc14574`.

[Aks96] Mehmet Aksit. Separation and composition of concerns in the object-oriented model. *ACM Comput. Surv.*, page 148, 1996.

[Apa06a] Apache. Apache sandesha, 2006. `http://ws.apache.org/sandesha/`.

[Apa06b] Apache. Apache wss4j, 2006. `http://ws.apache.org/wss4j/`.

[AVMM04] Rohit Aggarwal, Kunal Verma, John Miller, and William Milnor. Constraint driven web service composition in meteor-s. In *SCC '04 : Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 23–30, Washington, DC, USA, 2004. IEEE Computer Society.

[BG07] Luciano Baresi and Sam Guinea. Dynamo and self-healing bpel compositions. In *ICSE COMPANION '07 : Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 69–70, Washington, DC, USA, 2007. IEEE Computer Society.

[BGP05] Luciano Baresi, Sam Guinea, and Pierluigi Plebani. Ws-policy for service monitoring. In *TES : Technologies for E-Services*, volume 3811 of *LNCS*, pages 72–83. Springer, 2005.

[BGP07] Luciano Baresi, Sam Guinea, and Pierluigi Plebani. Policies and aspects for the supervision of bpel processes. In *CAiSE : International Conference on Advanced Information Systems Engineering*, volume 4495 of *LNCS*, pages 340–354. Springer, 2007.

[BM02] Luciano Porto Barreto and Gilles Muller. Bossa : a language-based approach to the design of real-time schedulers. In *10th International Conference on Real-Time Systems (RTS'2002)*, pages 19–31, Paris, France, mar 2002.

[BRL07] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux. A declarative approach for qos-aware web service compositions. In Bernd J. Kraemer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Fifth International Conference on*

Service-Oriented Computing (ICSOC), volume 4749 of *LNCS*, pages 422–428. Springer, 2007.

[CIjJ+00] Fabio Casati, Ski Ilnicki, Li jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. eflow : A platform for developing and managing composite e-services. In *AIWoRC*, pages 341–348, 2000.

[CM04] Anis Charfi and Mira Mezini. Aspect-oriented web service composition with AO4BPEL. In *Proceedings of the 2nd European Conference on Web Services (ECOWS)*, volume 3250 of *LNCS*, pages 168–182. Springer, September 2004.

[CMSA02] Jorge Cardoso, John Miller, Amit Sheth, and Jonathan Arnold. Modeling quality of service for workflows and web service processes. Technical Report UGACS-TR-02-002, Computer Science Department, University of Georgia, 2002.

[Coi06] Pierre Cointe. *Les langages a objets*. Vuibert, October 2006.

[Con04] Charles Consel. *Domain-Specific Program Generation ; International Seminar, Dagstuhl Castle*, chapter From A Program Family To A Domain-Specific Language, pages 19–29. Number 3016 in Lecture Notes in Computer Science, State-of-the-Art Survey. Springer-Verlag, 2004.

[CPE+06] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Francesco Perfetto, and Maria Luisa Villani. Service composition (re)binding driven by application-specific qos. In Asit Dan and Winfried Lamersdorf, editors, *ICSOC*, volume 4294 of *LNCS*, pages 141–152. Springer, 2006.

[CPEV05] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *GECCO*, pages 1069–1075. ACM, 2005.

[CS01] Fabio Casati and Ming-Chien Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3) :143–163, May 2001.

[CSHM06] Anis Charfi, Benjamin Schmeling, Andreas Heizenreder, and Mira Mezini. Reliable, secure, and transacted web service compositions with ao4bpel. In *Proceedings of the 4th IEEE European Conference on Web Services (ECOWS)*, December 2006.

[CSM+04] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics : Science, Services and Agents on the World Wide Web*, 1(3) :281–308, April 2004.

[Dec03] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[Dij82] Edsger W. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing : A Personal Perspective*, pages 60–66. Springer-Verlag, 1982.

[DLBS01] Pierre-Charles David, Thomas Ledoux, and Noury M. Bouraqadi-Saadani. Two-step weaving with reflection using aspectj. oct 2001.

[Dub02] Jean-Jacques Dubray. Bpel metamodel, 2002. `http://www.ebpml.org/bpel4ws.htm`.

[EFB01] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming. *CACM : Communications of the ACM*, 44(10) :29–32, 2001.

[EMT06] Abdelkarim Erradi, Piyush Maheshwari, and Vladimir Tosic. Policy-driven middleware for self-adaptation of web services compositions. In *Middleware*, volume 4290 of *LNCS*, pages 62–80. Springer, 2006.

[ES06] Onyeka Ezenwoye and Seyed Masoud Sadjadi. Trap/bpel : A framework for dynamic adaptation of composite services. Technical Report FIU-SCIS-2006-06-02, 2006.

[ETM07] Abdelkarim Erradi, Vladimir Tosic, and Piyush Maheshwari. Masc - .net-based middleware for adaptive composite web services. In *ICWS International Conference on Web Services*, pages 727–734. IEEE Computer Society, 2007.

[HL95] Walter L. Hursch and Cristina Videira Lopes. Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA, USA, February 1995.

[HS05] Michael N. Huhns and Munindar P. Singh. Service-oriented computing : Key concepts and principles. *IEEE Internet Computing*, 9(1) :75–81, 2005.

[Hug89] John Hughes. Why functional programming matters. *Computer Journal*, 32(2) :98–107, 1989.

[IBM04] IBM. Web services security, 2004. `http://www.ibm.com/developerworks/library/specification/ws-secure/`.

[IBM05] IBM. Web services reliable messaging, 2005. `http://www.ibm.com/developerworks/library/specification/ws-rm/`.

[ISO] ISO/IEC. Cd 15935 information technology : Open distributed processing - reference model - quality of service. (cd ballo), october 1998.

[ISO98] ISO/IEC. Itu-t recommandation x.641 – iso/iec 13236 : Information technology – quality of service : Framework, 1998.

[Jae07] Michael C. Jaeger. *Optimising Quality-of-Service for the Composition of Electronic Services*. PhD thesis, Berlin University of Technology, jan 2007.

[jJMS02] Li jie Jin, Vijay Machiraju, and Akhil Sahai. Analysis of service-level agreement for web services. Technical Report HPL-2002-180, 2002.

[JRGM04] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. Qos aggregation for web service composition using workflow patterns. In *EDOC '04 : Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International (EDOC'04)*, pages 149–159, Washington, DC, USA, 2004. IEEE Computer Society.

[KdRB91] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The art of metaobject protocol*. MIT Press, Cambridge, MA, USA, 1991.

[KFY+06] Pranam Kolari, Tim Finin, Yelena Yesha, Kelly Lyons, Jen Hawkins, and Stephen Perelgut. Policy management of enterprise systems : A requirements study. In *POLICY '06 : Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 231–234, Washington, DC, USA, 2006. IEEE Computer Society.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G. Griswold. An overview of aspectj. In *ECOOP '01 : Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353, London, UK, 2001. Springer-Verlag.

[KL03] Alexander Keller and Heiko Ludwig. The wsla framework : Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.*, 11(1) :57–81, 2003.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[LDK04] Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona : an architecture and library for creation and monitoring of ws-agreements. In *ICSOC '04 : Proceedings of the 2nd international conference on Service oriented computing*, pages 65–74, New York, NY, USA, 2004. ACM Press.

[LSE03] D.Davide Lamanna, James Skene, and Wolfgang Emmerich. Slang : A language for defining service level agreements. In *9th IEEE Workshop on Future Trends in Computing Systems*, pages 100–106, San Juan, Puerto Rico, 2003. IEEE Computer Society Press.

[Mae87] Pattie Maes. Concepts and experiments in computational reflection. *SIGPLAN Not.*, 22(12) :147–155, 1987.

[MC96] Jacques Malenfant and Pierre Cointe. Aspect-oriented programming versus reflection : a first draft, 1996.

[MHS05] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4) :316–344, 2005.

[MK03] Hidehiko Masuhara and Gregor Kiczales. A modeling framework for aspect-oriented mechanisms. In *ECOOP '03 : Proceedings of the 17th European Conference on Object-Oriented Programming*, volume 2734, pages 2–28. Springer-Verlag, July 2003.

[MRC+00] Fabrice Mérillon, Laurent Réveillère, Charles Consel, Renaud Marlet, and Gilles Muller. Devil : an idl for hardware programming. In *OSDI'00 : Proceedings of the 4th conference on Symposium on Operating System Design & Implementation*, pages 2–2, Berkeley, CA, USA, 2000. USENIX Association.

[OT99] Harold Ossher and Peri Tarr. Multi-dimensional separation of concerns in hyperspace. Technical Report RC 21452(96717)16APR99, 1999.

[Pap03] Mike P. Papazoglou. Service -oriented computing : Concepts, characteristics and directions. In *WISE '03 : Proceedings of the Fourth International Conference on Web Information Systems Engineering*, page 3, Washington, DC, USA, 2003. IEEE Computer Society.

[PCW07] Noel Plouzeau, Franck Chauvel, and Guillaume Waignier. Specification du meta-modele pivot. Technical Report F.2.1, RNTL Faros, jul 2007.

[PEV+06] Massimiliano Di Penta, Raffaele Esposito, Maria Luisa Villani, Roberto Codato, Massimiliano Colombo, and Elisabetta Di Nitto. Ws binder : a framework to enable dynamic binding of composite web services. In *SOSE '06 : Proceedings of the 2006 international workshop on Service-oriented software engineering*, pages 74–80, New York, NY, USA, 2006. ACM.

[PSDF01] Renaud Pawlak, Lionel Seinturier, Laurence Duchien, and Gerard Florin. Jac : A flexible solution for aspect-oriented programming in java. In *REFLECTION '01 : Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, pages 1–24, London, UK, 2001. Springer-Verlag.

[SCA08] Fraunhofer Institute SCAI. wsag4j organizational pom, 2008. `http://packcs-e0.scai.fraunhofer.de/mss-project/wsag4j/project-info.html`.

[Slo94] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2 :333–360, 1994.

[SM05] Seyed Masoud Sadjadi and Philip K. McKinley. Using transparent shaping and web services to support self-management of composite systems. In *ICAC '05 : Proceedings of the Second International Conference on Automatic Computing*, pages 76–87, Washington, DC, USA, 2005. IEEE Computer Society.

[SMCS04] Seyed Masoud Sadjadi, Philip K. McKinley, Betty H. C. Cheng, and R. E. Kurt Stirewalt. Trap/j : Transparent generation of adaptable java programs. In *CoopIS : International Conference on Cooperative Information Systems*, volume 3291 of *LNCS*, pages 1243–1261. Springer, 2004.

[Smi84] Brian Cantwell Smith. Reflection and semantics in lisp. In *POPL '84 : Proceedings of the 11th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 23–35, New York, NY, USA, 1984. ACM.

[SN98] Roy W. Schulte and Yefim V. Natis. Service oriented architectures, part 1 and 2, 1998. `http://www.gartner.com/`.

[SOA] Soap specifications. `http://www.w3.org/TR/soap/`.

[Szy98] Clemens Szyperski. *Component software : beyond object-oriented programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[TEM07] Vladimir Tosic, Abdelkarim Erradi, and Piyush Maheshwari. Ws-policy4masc - a ws-policy extension used in the masc middleware. In *IEEE SCC*, pages 458–465. IEEE Computer Society, 2007.

[Tho02] Dave A. Thomas. Reflective software engineering - from mops to aosd. *Journal of Object Technology*, 1(4) :17–26, 2002.

[TPP02] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. Wsol - web service offerings language. In *CAiSE '02/ WES '02 : Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, pages 57–67, London, UK, 2002. Springer-Verlag.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages : An annotated bibliography. *SIGPLAN Notices*, 35(6) :26–36, 2000.

[W3C03] W3C. Qos for web services – requirements and possible approaches, 2003. `http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/`.

[Wes01] Andrea Westerinen. Rfc 3198 : Terminology for policy-based management, Nov 2001.

[WSA] Web service architecture. `http://www.w3.org/TR/ws-arch/`.

[WSD] Wsdl specifications. `http://www.w3.org/TR/wsdl/`.

[YL05a] Tao Yu and Kwei-Jay Lin. A broker-based framework for qos-aware web service composition. In *EEE '05 : Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 22–29, Washington, DC, USA, 2005.

[YL05b] Tao Yu and Kwei-Jay Lin. Service selection algorithms for composing complex services with multiple qos constraints. pages 130–143. 2005.

[ZBD+03] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW '03 : Proceedings of the 12th international conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.

[ZBN+04] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5) :311–327, 2004.

Une Approche Déclarative pour la Gestion de la Qualité de Service dans les Compositions de Services

Fabien Baligand

L'avènement des architectures orientées service tend à promouvoir un style d'architecture logicielle où des services, exposant des fonctionnalités accessibles à l'aide de protocoles hautement standardisés, sont composés avec un couplage lâche. Dans un tel contexte où les services sont aisément amenés à être remplacés ou utilisés par un grand nombre d'utilisateurs, la notion de Qualité de Service (QdS), qui s'intéresse à la qualité de la relation entre un service et ses clients, constitue un enjeu majeur. La QdS regroupe diverses préoccupations telles que la sécurité, la garantie de livraison, la performance (temps de réponse ou accessibilité) ou encore le coût. Bien qu'il existe déjà d'importants travaux autour des compositions de services, qui ont notamment permis l'élaboration du standard BPEL4WS, le problème de la gestion de la QdS dans les compositions de services manque de solution flexible, réutilisable et offrant un degré d'abstraction approprié.

L'objectif de cette thèse est de faciliter la gestion de la QdS dans les compositions de services en s'appuyant sur une meilleure séparation des préoccupations. Pour cela nous proposons QoSL4BP, un langage dédié qui permet la spécification de politiques de QdS à l'échelle des compositions de services. Ces politiques gèrent des contraintes et des mécanismes de QdS statiquement et dynamiquement à l'aide d'un ensemble limité de primitives de haut niveau. En encapsulant l'expertise liée au domaine de la gestion de QdS et en offrant une expressivité de haut niveau, le langage QoSL4BP permet une spécification aisée, flexible et réutilisable de la gestion des contrats de QdS et des mécanismes liés à la QdS. Ce langage est mis en oeuvre par notre plateforme ORQOS qui coopère de manière non intrusive avec les moteurs d'orchestration. Ainsi, au moment du déploiement d'une composition de services, ORQOS sélectionne les services de la composition, selon leurs offres de QdS et les exigences spécifiées dans les politiques QoSL4BP. A l'exécution, les politiques QoSL4BP permettent de réagir aux variations de QdS et de mettre en oeuvre des mécanismes liés à la gestion de QdS. Deux scénarii, appartenant respectivement aux domaines des télécommunications et du médical, permettent de valider l'approche proposée.

Mots-clés : Architecture orientée service, Séparation des préoccupations, Langage dédié, Contrat

The advent of Service Oriented Architectures tends to promote a new kind of software architecture where services, exposing features accessible through highly standardized protocols, are composed in a loosely coupled way. In such a context, where services are likely to be replaced or used by a large number of clients, the notion of Quality of Service (QoS), which focuses on the quality of the relationship between a service and its customers, becomes a key challenge. QoS deals with multiple concerns such as security, reliability, performance (response time or availability), or even the cost of the service. Although much work has been carried out in the field of service composition, in particular leading to the elaboration of the BPEL4WS standard, the challenge of QoS management in service compositions still lacks flexible and reusable solutions offering a suitable level of abstraction.

This thesis aims to ease QoS management in service compositions through a better separation of concerns. For this purpose, we designed QoSL4BP, a domain-specific language which allows QoS policy specification on service compositions. Such policies handle QoS constraints and mechanisms, both at pre-deployment time and runtime, by use of a limited set of high level primitives. By encapsulating the expertise of QoS management domain and by offering a high level expressivity, the QoSL4BP language enables an easy, flexible and reusable specification of both QoS contracts management and QoS related mechanisms implementation. This language is executed by our ORQOS platform, which cooperates in a non-intrusive way with orchestration engines. Thus, at pre-deployment time, the ORQOS platform performs service planning depending on services QoS offers and on the QoS requirements in QoSL4BP policies. At runtime, the QoSL4BP policies make it possible to react to QoS variations and to enact QoS management related mechanisms. Two scenarii, belonging to telecommunications and medical domains, validate our approach.

Keywords : Service-oriented architecture, Separation of concerns, Domain-specific language, Service level agreement