UNIVERSITÉ DU QUÉBEC À MONTRÉAL

SURVEILLANCE DYNAMIQUE DE COMPOSITIONS DE SERVICES WEB A L'AIDE DE PROTOCOLES DE COMPORTEMENT

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

WASSIM JENDOUBI

OCTOBRE 2010

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

Remerciements tout particuliers pour mon directeur de recherche Monsieur Guy Tremblay, pour sa grande disponibilité, pour ses précieux conseils et pour son soutien indéfectible qui m'a permis de venir à bout de ce travail de recherche.

Je tiens également à remercier mes professeurs du département d'informatique de l'Université du Québec à Montréal qui m'ont formé tout au long de ces années de maîtrise; notamment Monsieur Étienne Gagnon qui a considérablement contribué à mon initiation à la recherche.

J'aimerais enfin saluer les membres du laboratoire de recherche sur les technologies du commerce électronique (LATECE) de l'UQAM; ils m'ont permis d'évoluer dans une atmosphère propice au travail et d'avoir tant de discussions fructueuses qui m'ont aiguillé lors de mes recherches.

TABLE DES MATIÈRES

LISTE DES FIGURES . . . v

LISTE DES TABLEAUX . . . vii

RÉSUMÉ . . . ix

INTRODUCTION . . . 1

CHAPITRE I
SERVICES WEB ET CONTRATS . . . 3

1.1 Architecture orientée service . . . 3

1.1.1 Web Services Description Language . . . 5

1.1.2 Business Process Execution Language . . . 7

1.2 Programmation par contrat . . . 9

1.3 Vérification de contrats de services Web . . . 10

CHAPITRE II
CONTRATS DE SERVICES WEB ET VÉRIFICATION DYNAMIQUE . . . 12

2.1 Contrats de services Web . . . 12

2.1.1 Contrats fonctionnels . . . 12

2.1.2 Contrats de qualité de service . . . 15

2.1.3 Contrats d'infrastructures . . . 16

2.2 Vérification dynamique des contrats de services Web . . . 16

CHAPITRE III
SURVEILLANCE DYNAMIQUE DES PROTOCOLES DE COMPORTEMENT . . . 19

3.1 Exigences pour un cadre d'application de surveillance dynamique . . . 19

3.2 Instances de processus et événements BPEL . . . 20

3.3 Protocoles de comportement . . . 20

3.4 Surveillance dynamique . . . 21

3.5 Architecture . . . 24

CHAPITRE IV
BPEL.RPM : UNE MISE EN OEUVRE D'UN SYSTÈME DE SURVEILLANCE DYNAMIQUE DE PROTOCOLES DE COMPORTEMENT . . . 26

4.1 Formalisme et édition de protocoles . . . 26

4.2 Module de détection des événements . . . 28

4.3 Module de surveillance dynamique . . . 31

4.3.1 Représentation dynamique des protocoles 29

4.3.2 Surveillance dynamique 30

4.3.3 Connexion aux modules de détection 31

4.4 Module de compensation 33

Chapitre V
ÉTUDES DE CAS 36

5.1 Utilisation de BPEL.RPM 36

5.2 Service de traitement de demandes de prêt 37

5.3 Service de réservation de voyage 40

5.4 Service de vente aux enchères 46

CONCLUSION 54

Annexe A
MODÈLE DU DOMAINE DE NOTRE SYSTÈME DE SURVEILLANCE DYNAMIQUE
DE PROTOCOLES DE COMPORTEMENT 56

Annexe B
DIAGRAMME DE CLASSES DE BPEL.RPM 57

Annexe C
CODE SOURCE BPEL D'UN SERVICE DE VENTE AUX ENCHÈRES 58

Bibliographie 62

LISTE DES FIGURES

1.1 Services Web. 4

1.2 Métamodèle WSDL (47). 7

1.3 Java Business Integration. 9

2.1 Relation entre contrats de chorégraphie, contrats d'orchestration et protocoles de comportement. 15

3.1 Extrait du modèle du domaine : instances de processus et événement BPEL. . . 20

3.2 Extrait du modèle du domaine : protocoles de comportement. 22

3.3 Représentation graphique (diagramme d'activités) d'un exemple de protocole de comportement. 23

3.4 Représentation en automate d'un exemple de protocole de comportement. 23

3.5 Architecture : première décomposition fonctionnelle. 24

3.6 Architecture : deuxième décomposition fonctionnelle. 25

4.1 Architecture : Décomposition fonctionnelle. 27

4.2 Diagramme de paquetage de BPEL.RPM. 27

4.3 Architecture du module de détection d'événements associé à BPEL.RPM. 30

4.4 Protocole représenté par un automate fini déterministe. 32

4.5 Diagramme de séquence de la logique de surveillance dynamique de BPEL.RPM. 35

4.6 Extrait du diagramme de classes de BPEL.RPM : relation entre le module de détection d'événements et celui de surveillance dynamique. 36

4.7 Extrait du diagramme de classes de BPEL.RPM : relation entre le module de compensation et celui de surveillance dynamique. 35

5.1 Processus d'utilisation de BPEL.RPM. 37

5.2 Service de traitement de demandes de prêt. 38

5.3 Service de traitement de demandes de prêt : protocole "Global". 38

5.4 Service de traitement de demandes de prêt : automate représentant le protocole "Global". 40

5.5 Service de réservation de voyage. 42

5.6 Service de réservation de voyage : protocole "Has no reservations". 43

5.7 Service de réservation de voyage : automate représentant le protocole "Has no reservations". 43

5.8 Service de vente aux enchères. 49

5.9 Service de vente aux enchères : protocole "Global". 50

5.10 Service de vente aux enchères : automate représentant le protocole "Global". . . . 50

A.1 Modèle du domaine de notre système de surveillance dynamique de protocoles de comportement. 56

B.1 Diagramme de classes de BPEL.RPM. 57

LISTE DES TABLEAUX

4.1 Syntaxe abstraite des expressions régulières utilisées dans BPEL.RPM. 30

4.2 Exemple d'une table de correspondance d'un protocole de comportement. 32

5.1 Service de traitement de demandes de prêt : table de correspondance du protocole "Global". 42

5.2 Service de réservation de voyage : table de correspondance du protocole "Has no reservations". 48

5.3 Service de vente aux enchères : table de correspondance du protocole "Global". . 55

LISTE DES LISTAGES

1.1 Interface Java d'un service (SimpleDirectory). 5

1.2 Extrait d'une interface WSDL d'un service (SimpleDirectory). 6

3.1 Code source d'une activité BPEL. 21

4.1 Exemple d'un protocole de comportement pour BPEL.RPM. 29

5.1 Service de traitement de demandes de prêt : définition du protocole "Global". . . 43

5.2 Service de traitement de demandes de prêt : messages de surveillance du protocole "Global". 44

5.3 Service de réservation de voyage : définition du protocole "Has no reservations". 47

5.4 Service de réservation de voyage : messages de surveillance du protocole "Has no reservations". 50

5.5 Service de réservation de voyage : messages de surveillance du protocole "Has no reservations" — cas d'un itinéraire non vierge. 51

5.6 Service de vente aux enchères : définition du protocole "Global". 54

5.7 Service de vente aux enchères : messages de surveillance du protocole "Global". . 56

C.1 Service de vente aux enchères : code source. 61

RÉSUMÉ

Dans ce travail nous proposons une adaptation du paradigme de la programmation par contrat — contrats exprimés sous forme de protocoles de comportement — au contexte des architectures orientées services, et ce à travers la conception d'un cadre d'applications (*framework*) supportant l'ensemble du processus de contractualisation, à savoir, la définition des contrats, la surveillance dynamique et la réaction en fonction du respect ou non des règles établies.

La solution proposée permet de détecter les ruptures de contrat à chaud, c'est-à-dire en cours d'exécution des compositions de services, ouvrant ainsi la porte à l'instauration de mécanismes dynamiques de compensation. Les contrats surveillés représentent des protocoles de comportements de processus BPEL, ce qui permet de définir des contraintes sur l'ordre d'exécution des opérations publiques des services partenaires. Nous en présentons également une mise en œuvre, BPEL.RPM, qui est adaptable, dans le sens où elle peut aisément intégrer des modules externes de compensation, mais qui est aussi portable, puisqu'elle fonctionne indépendamment de l'environnement d'exécution des services Web.

Mots clés: services Web, programmation par contrat, surveillance dynamique, BPEL.

INTRODUCTION

Depuis quelques années, l'utilisation des architectures orientées services (AOS) s'est généralisée dans les grandes entreprises internationales. Ainsi, selon une étude statistique menée par Gartner en 2008 auprès de 200 compagnies mondiales, 78 pour cent des répondants disaient avoir déjà eu recours à des AOS dans une partie de leurs organisations ou planifiaient de le faire dans les 12 prochains mois (27). Une architecture orientée service est une construction logicielle mettant en relation plusieurs composants autonomes et distribués. Cette séparation spatiale et temporelle entraine un grand besoin de confiance entre les différents acteurs (30).

Au-delà des systèmes informatiques distribués, la problématique de confiance a toujours été au centre des échanges humains, et depuis plus de 2000 ans, le droit romain a su y répondre (33) en formalisant la notion de contrat, i.e., un « accord de volonté par lequel une ou plusieurs personnes s'obligent envers une ou plusieurs autres à exécuter une prestation de service » (48). Dans ce travail nous tenterons d'appliquer le concept de contractualisation aux architectures orientées services en proposant un cadre d'applications (*framework*) supportant l'ensemble du processus, notamment la définition des contrats, la surveillance dynamique et la réaction en fonction du respect ou non des règles établies.

La notion de programmation par contrat est loin d'être nouvelle dans le monde du génie logiciel, effectivement, plusieurs autres études abordent la problématique de son application aux architectures orientées services. Notre travail, quant à lui, se distingue par la nature des contrats surveillés, à savoir les protocoles de comportement qui permettent de spécifier un certain ordre d'exécution des différentes opérations visibles de l'extérieur d'une composition de services telles que l'invocation d'une opération ou la réponse à une requête d'un service partenaire. La principale caractéristique de notre modèle réside dans sa capacité à détecter les ruptures de contrat à chaud, c'est-à-dire en cours d'exécution des processus ouvrant ainsi la porte à l'instauration de mécanismes dynamiques de compensation. L'originalité de notre modèle se situe également dans sa mise en œuvre qui a donné naissance à BPEL.RPM, un cadre d'applications de surveillance de protocoles de comportement, adaptable, dans le sens où il permet aisément l'intégration de modules externes de compensation, mais aussi portable, puisqu'il fonctionne indépendamment de l'environnement d'exécution des services Web.

Dans le premier chapitre de ce mémoire nous présenterons quelques notions et technologies liées aux architectures orientées services et au paradigme de la programmation par contrat. Par la suite, dans le deuxième chapitre, nous exposerons une taxinomie des contrats de services Web ; nous verrons en effet qu'il existe plusieurs niveaux et plusieurs formes de contrats. Ce chapitre contiendra également une présentation de quelques projets de recherche ayant mis en œuvre des systèmes de surveillance dynamique pour certains types de contrats. Nous proposerons dans un troisième chapitre un cadre d'applications de surveillance dynamique de protocoles de comportement, dont la mise en œuvre sera détaillée dans le quatrième chapitre. Enfin, le cinquième et dernier chapitre sera consacré à l'analyse du comportement de BPEL.RPM à travers trois études de cas.

Chapitre I

SERVICES WEB ET CONTRATS

Les problématiques et les solutions présentées tout au long de ce mémoire touchent à plusieurs notions du génie logiciel, certaines liées aux architectures, d'autres à des paradigmes de programmation, ou à des plateformes technologiques bien spécifiques. Tous ces éléments seront brièvement présentés dans le chapitre qui suit.

Nous commencerons par la présentation des architectures orientées services et, plus particulièrement, des concepts et technologies liés aux services Web. Par la suite, nous introduirons le paradigme de programmation par contrat, avec un accent particulier sur la vérification de protocoles.

1.1 Architecture orientée service

Une architecture orientée service est une architecture logicielle distribuée mettant en jeu plusieurs services. Ces derniers sont exposés par leurs fournisseurs à des clients.

Un service est essentiellement caractérisé par :

- une forte cohésion interne assurant l'indépendance du service ;
- un couplage externe faible garantissant à la fois une interopérabilité entre des acteurs hétérogènes et une substitution plus aisée d'un service par un autre.

Plusieurs techniques permettent de réaliser de telles architectures distribuées, on cite parmi elles CORBA (*Common Object Request Broker Architecture*) (28), RMI (*Remote method invocation*) (53) ou DCOM (*Distributed Component Object Model*) (37). Dans notre cas, on s'intéressera aux architectures basées sur les services Web, où on retrouve essentiellement deux tendances (49) :

- les services basés sur une architecture REST (*REpresentational State Transfer*) (24) ;
- les services utilisant le protocole d'échange d'informations SOAP (3). Cette solution, soutenue par le consortium W3C (*World Wide Web Consortium*) (60), est illustrée par la figure 1.1.

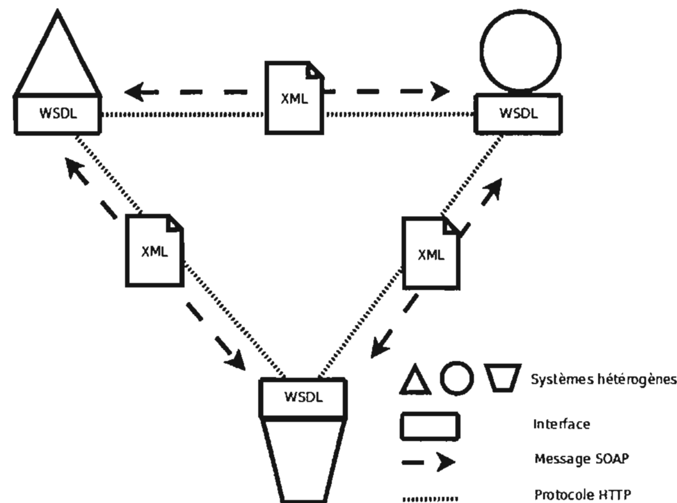

FIGURE 1.1 Services Web.

Pour le reste de notre travail, nous nous limiterons à la définition d'un service Web donnée par le W3C, que nous avons traduite ainsi :

Définition 1 *Un service Web est un système logiciel conçu pour supporter des interactions interopérables de machine à machine à travers un réseau. Il est doté d'une interface dont le format (en particulier WSDL) est compréhensible par un autre système logiciel. Les autres systèmes interagissent avec le service Web selon la manière prescrite par sa description en utilisant des messages SOAP, typiquement acheminés à travers HTTP et une sérialisation XML en conjonction avec d'autres standards Web. (61)*

Il est à noter qu'un service Web peut être aussi bien atomique que composé, et c'est justement à ce dernier type de services que nous nous sommes intéressés dans notre travail. Nous présenterons donc, dans les deux sous-sections qui suivent, les langages WSDL et BPEL qui permettent respectivement de définir des interfaces pour des services Web et des compositions de services Web [1].

1. Pour plus de détails sur la notion de composition de services Web, appelée aussi contrat d'orchestration,

```
public interface SimpleDirectory {
    public String getNameFromID(int getNameFromIDInput);
}
```

Listage 1.1 Interface Java d'un service (SimpleDirectory).

1.1.1 Web Services Description Language

Web Services Description Language (WSDL) (17) est un langage de description d'interface de services Web. Il permet de concentrer toutes les informations nécessaires à l'invocation d'un service dans un seul fichier.

À la manière d'une interface Java qui exposerait l'ensemble des opérations publiques d'une API, une interface WSDL se contente de représenter un processus sous forme d'un ensemble de points d'accès (`endpoints`). Les opérations et messages du service sont définis dans une première section dite abstraite, et ce à travers les éléments suivants :

- `<types>` : types des messages envoyés ou reçus par le service ;
- `<message>` : données d'entrée ou de sortie d'une opération du service ;
- `<operation>` : méthode du service pouvant accepter et/ou retourner zéro ou plusieurs messages ;
- `<portType>` : collection d'opérations fournies par un point d'accès du service.

Les listages 1.1 et 1.2 d'une interface Java et d'un extrait du fichier WSDL équivalent illustrent l'analogie entre ces deux concepts.

Pour compléter la définition des points d'accès du service, les opérations sont associées à des protocoles réseau et à des formats de messages dans une section du document WSDL généralement qualifiée de concrète (17), et ce à travers les éléments suivants :

- `<port>` : point d'accès au service, défini par son adresse ;
- `<service>` : ensemble de point d'accès ;
- `<binding>` : association de points d'accès à des opérations du service et à des protocoles et formats de messages.

La figure 1.2 (47) montre un métamodèle simplifié de WSDL que nous avons légèrement édité pour mieux illustrer les relations entre les différents éléments de la section abstraite et

---

voir section 2.1.1, page 13.

```
<!--===================TYPES===================-->
<types>
    <xs:schema xmlns:tns="http://simpledirectory.sample.bpm.com/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
        targetNamespace="http://simpledirectory.sample.bpm.com/">
        <xs:element name="getNameFromIDInput" type="xs:int"/>
        <xs:element name="getNameFromIDResponse"
        type="xs:string"/>
    </xs:schema>
</types>
<!--===================MESSAGES=================-->
<message name="getNameFromIDInput">
    <part name="parameters" element="tns:getNameFromIDInput"></part>
</message>
<message name="getNameFromIDResponse">
    <part name="parameters" element="tns:getNameFromIDResponse"></part>
</message>
<!--=================PORT TYPES================-->
<portType name="SimpleDirectory">
    <operation name="getNameFromID">
        <ns5:PolicyReference xmlns:ns5="http://www.w3.org/ns/ws-policy"
        URI="#SimpleDirectoryPortBinding_getNameFromID_WSAT_Policy">
        </ns5:PolicyReference>
        <input message="tns:getNameFromIDInput"></input>
        <output message="tns:getNameFromIDResponse"></output>
    </operation>
</portType>
```

Listage 1.2 Extrait d'une interface WSDL d'un service (SimpleDirectory).

ceux de la section concrète.

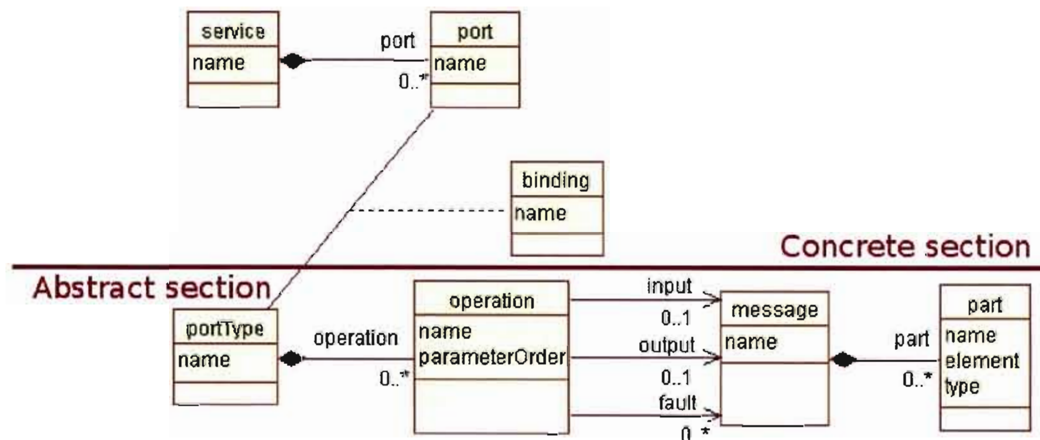

FIGURE 1.2 Métamodèle WSDL (47).

1.1.2 Business Process Execution Language

Web Services Business Process Execution Language (BPEL) (4) est un langage de composition de service Web. Il permet de créer un nouveau service en ordonnançant d'autres services Web identifiés par leurs interfaces WSDL. Autre aspect important : il permet de bien identifier les partenaires avec lesquels le service va interagir, ainsi que le type d'interactions qu'il aura avec eux, à savoir, client vs. fournisseur. D'un point de vue externe, ce sont donc les diverses interactions avec ces partenaires qui caractérisent le comportement du service BPEL.

Un processus BPEL peut être composé aussi bien d'activités internes, telles que des initialisations de variables, que d'appels à d'autres services, et c'est à ce dernier cas que nous nous sommes intéressés dans notre travail, à savoir, les interactions avec les partenaires. Cette relation entre un processus BPEL et ses partenaires peut se résumer à un échange de messages, mis en œuvre par les activités BPEL suivantes :

- `<invoke>` : invocation d'un service Web ;
- `<receive>` : réception bloquante d'un message ;
- `<reply>` : envoi d'une réponse synchrone ;
- `<pick>` : attente d'un événement parmi un ensemble pouvant être composé d'alarmes `<onAlarm>` ou de messages à recevoir `<onMessage>`.

Les services partenaires et leurs opérations sont référencés par deux balises :

- `<partnerLink>` définissant la « structure de la relation » (4) avec le service partenaire ;

- <operation> identifiant l'opération invoquée au niveau du service partenaire.

Environnement d'exécution

Plusieurs plateformes permettent d'exécuter des compositions de services BPEL. Pour mettre en œuvre certaines parties de notre solution [2], nous avons opté pour l'intergiciel libre OpenESB (52). OpenESB est un *Enterprise Service Bus* (ESB) respectant la norme *Java Business Integration* (JBI), définie dans la spécification JSR 208 (51).

Un ESB permet l'intégration de plusieurs logiques métier hétérogènes. Pour ce faire, un JBI tel que OpenESB dispose de deux types de composants :
- engins d'exécution (*service engine*) qui permettent d'exécuter les différentes logiques métier (BPEL, SQL, JEE, etc.) ;
- composants de connexion (*binding component*) implémentant les différents standards de communication (FTP, HTTP, JDBC, etc.).

Les messages reçus par les composants de connexion sont acheminés vers les engins concernés à travers un mécanisme de routage (*Normalized Message Router* ou *NMR*). Ces relations sont illustrées par la figure 1.3, où on voit aussi que l'intergiciel OpenESB est doté d'une base de données, `bpelseDB`. Cette base de persistance permet à l'engin d'exécution BPEL de garder une trace des exécutions de processus (50).

1.2 Programmation par contrat

La notion de programmation par contrat (*design by contract*) a été introduite par Meyer (36). Ce paradigme de programmation repose essentiellement sur l'utilisation d'une notation formelle permettant de spécifier des assertions et sur un mécanisme assurant leur évaluation dynamique. Même si plusieurs langages supportent la notion d'assertion (18), Eiffel reste encore à notre connaissance le seul langage industriel incorporant d'une manière native un système complet de définition et de vérification des contrats.

Pour utiliser la programmation par contrat avec des langages qui, à la base, ne le permettaient pas, plusieurs approches peuvent être considérées :

---

2. Se référer au chapitre 3, page 18, pour une présentation de la solution envisagée et au chapitre 4, page 25, pour celle de sa mise en œuvre.

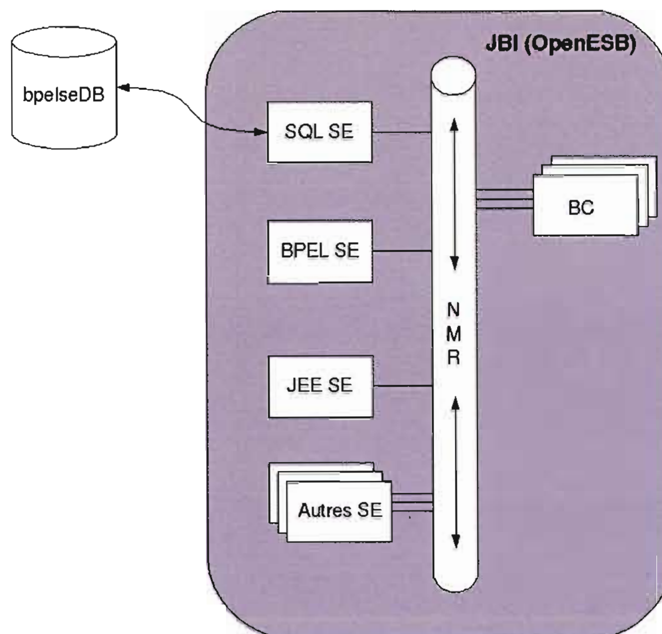

FIGURE 1.3 Java Business Integration.

- la modification du langage ;
- l'utilisation d'une bibliothèque ;
- l'interfaçage avec un langage permettant l'implémentation des contrats ;
- l'utilisation d'un préprocesseur, approche qui procède par un système d'annotations et qui ne nécessite aucune modification du compilateur ou de l'interpréteur du langage concerné.

Dans la prochaine section, mais également dans le prochain chapitre, nous verrons comment le paradigme de programmation par contrat a été abordé dans les architectures orientées services.

1.3 Vérification de contrats de services Web

Selon Tremblay et Chae (58), les contrats, et plus précisément les protocoles de service Web, ont deux intérêts majeurs :

- la vérification, aussi bien statique que dynamique ;
- la documentation, pour expliciter la manière avec laquelle un service pourrait être utilisé.

La vérification statique consiste à s'assurer qu'une implémentation d'un processus en particulier est conforme à un contrat prédéfini.

La vérification, ou surveillance dynamique (*run-time monitoring*) qui est l'objet de notre travail, permet de détecter les violations du contrat à chaud, c'est-à-dire en cours d'exécution du processus ouvrant ainsi la porte à l'instauration de mécanismes de récupération.

Tremblay et Chae (58) ont souligné la limite de l'utilisation des pré/post-conditions pour exprimer la sémantique des compositions de services Web. Cette limitation est due à l'impossibilité d'exprimer des séquences complexes d'opérations en se basant uniquement sur des pré/post-conditions d'autant plus quand les opérations en question sont sans états (*stateless*). Ce besoin est bien exprimé dans la spécification de WSCI, à savoir celui d'un contrat qui « décrit le comportement observable d'un service, ainsi que les règles d'interactions avec lui depuis l'extérieur [...] assez précis et sans ambiguïté pour que les acteurs externes sachent à chaque étape du processus, ce que ce dernier devra ou pourra, envoyer ou recevoir, à l'étape suivante. » (6) [3]. Les protocoles de comportement (voir section 2.1.1, page 12) remplissent parfaitement ce rôle, et c'est ce type de contrat que nous utiliserons dans notre travail pour exprimer la sémantique nécessaire à la surveillance des compositions de services Web.

Conclusion

Nous avons présenté dans ce chapitre différents concepts et technologies, particulièrement, les architectures orientées services, certains concepts et outils qui s'y rattachent et la programmation par contrat, notions nécessaires à la compréhension de la suite du mémoire.

Dans le prochain chapitre, nous verrons comment la notion de programmation par contrat est présente dans certains standards de services Web. D'autre part, nous y aborderons quelques projets de recherche ayant mis en œuvre des systèmes de vérification dynamique de contrats de services Web.

3. Notre traduction.

CHAPITRE II

CONTRATS DE SERVICES WEB ET VÉRIFICATION DYNAMIQUE

Dans les chapitres subséquents, nous exposerons notre solution de surveillance dynamique de contrats de services Web. Étant donné l'existence d'une grande diversité de contrats, nous nous efforcerons, en premier lieu, de classifier les différents contrats pouvant s'appliquer aux services Web, pour présenter par la suite quelques projets de recherche ayant mis en œuvre des systèmes de surveillance dynamique pour certains types de contrats de services Web.

2.1 Contrats de services Web

Nous présenterons dans cette section différents types de contrats liés aux services Web. Pour cela, nous nous baserons sur le travail de Tosic et Pagurek (55) qui distingue trois types de contrats: fonctionnels, de qualités et d'infrastructures. À noter que cette décomposition est inspirée des quatre niveaux de spécifications pour les objets et composants distribués décrits par Beugnard et al. (15).

2.1.1 Contrats fonctionnels

Les contrats fonctionnels décrivent ce que fait un service Web. Ceci peut être réalisé aussi bien avec une approche ontologique, par exemple avec OWL-S (*Semantic Markup for Web Services*) (16) ou WSMO (20), ou en combinant plusieurs types de contrats : contrats syntaxiques, contrats comportementaux, contrats de synchronisation, contrats d'orchestration et contrats de chorégraphie.

Contrats syntaxiques

Les contrats syntaxiques définissent les informations nécessaires à l'invocation du service. Avec des contrats WSDL (17) on pourra décrire un service Web en indiquant le protocole de communication et le format de message utilisé, mais surtout en précisant les signatures des opérations publiques.

Contrats comportementaux

Les contrats comportementaux définissent les exigences pour une bonne exécution des services Web. À la manière de WSCoL (10) et WSOL (56), ces exigences peuvent être définies par des assertions.

WSCoL permet d'annoter des processus écrits en BPEL (4) dans le but de définir des exigences fonctionnelles à base de préconditions et de postconditions. Ces annotations seront transformées par un préprocesseur en code BPEL supplémentaire, qui enrichira le code original et permettra d'évaluer dynamiquement les contrats. En cas de violation de contraintes, une stratégie de recouvrement d'activités écrites en WSReL (56) est exécutée. Dynamo est l'implémentation de ce système, une version orientée aspect de l'engin d'exécution ActiveBPEL (23) et qui s'appuie sur l'engin de règles JBoss Rule (38) pour assurer l'autocorrection (*self-healing*).

WSOL permet de définir plusieurs classes de services pour un processus ce qui permettra de moduler l'offre en fonction des besoins du client. À l'intérieur de ces classes de services les exigences seront définies grâce à des préconditions, postconditions et conditions futures (*future-condition*). Ce dernier type d'assertion, introduit pour la première fois dans WSOL (57), permet de retarder l'évaluation d'une postcondition, ce qui peut être intéressant dans plusieurs cas pratiques tels que la confirmation des livraisons (42). En plus des exigences fonctionnelles, WSOL permet d'inscrire dans les classes de services des exigences de qualité de service et d'authentification.

Contrats de synchronisation (protocoles de comportement)

Dans notre solution, nous proposons de surveiller des protocoles de comportement (*behavior protocols*) (46) appelés aussi interfaces comportementales (*behavioral interface*) (21) ou contrats de synchronisation (55). Ils permettent de spécifier des protocoles d'utilisation pour les différentes opérations d'un service Web en définissant les dépendances entre les différentes

opérations et les contraintes d'exécutions concurrentes. Ces contrats visent donc à définir les différents ordonnancements possibles, i.e., les contraintes sur l'ordre d'exécution des opérations. On retrouve aussi ces notions dans les *abstract process* de BPEL et les *Collaboration Protocol Profile* d'ebXML (41; 5) qui mettent l'accent sur le comportement observable de l'extérieur du service Web.

Contrats d'orchestration

Les contrats d'orchestration peuvent être considérés comme l'implémentation des protocoles de comportement. Ils partagent avec eux leur vue centrée sur un seul acteur, mais diffèrent principalement par leur orientation métier, puisqu'ils ne se limitent pas aux comportements observables du service Web mais spécifient également les comportements internes nécessaires à son exécution.

Même si les *executable process* de BPEL restent la référence pour ce type de contrats plusieurs autres langages permettent de les spécifier : XLANG, WSFL, XPDL et BPML (59).

Contrats de chorégraphie

Les contrats de chorégraphie décrivent la collaboration entre plusieurs services et leurs clients (généralement d'autres services Web) pour réaliser un objectif commun (21). Il s'agit d'une vue collaborative des échanges entre processus, qui permet à chaque partie de décrire sa part de la collaboration (43). Contrairement aux protocoles de comportement et aux contrats d'orchestration qui se concentrent sur le comportement d'un seul acteur, les contrats de chorégraphie se focalisent sur les messages échangés par les divers participants pour obtenir une vue globale des interactions.

Différentes approches permettent de réaliser cela. Ainsi, WSCI (6) utilise les informations inscrites dans les fichiers WSDL pour reconstituer le schéma global d'échange de messages. Par contre, WS-CDL (2) ainsi que ebXML, à travers son *Multiparty Collaboration Model* (22), ont opté pour la centralisation des descriptions, et ce en définissant les échanges par couple de processus. Ce ci permet dès lors de retrouver le schéma collaboratif global en regroupant les différentes descriptions.

Dans la figure 2.1, nous présentons le chevauchement des trois points de vue : chorégraphie, orchestration et protocoles de comportement.

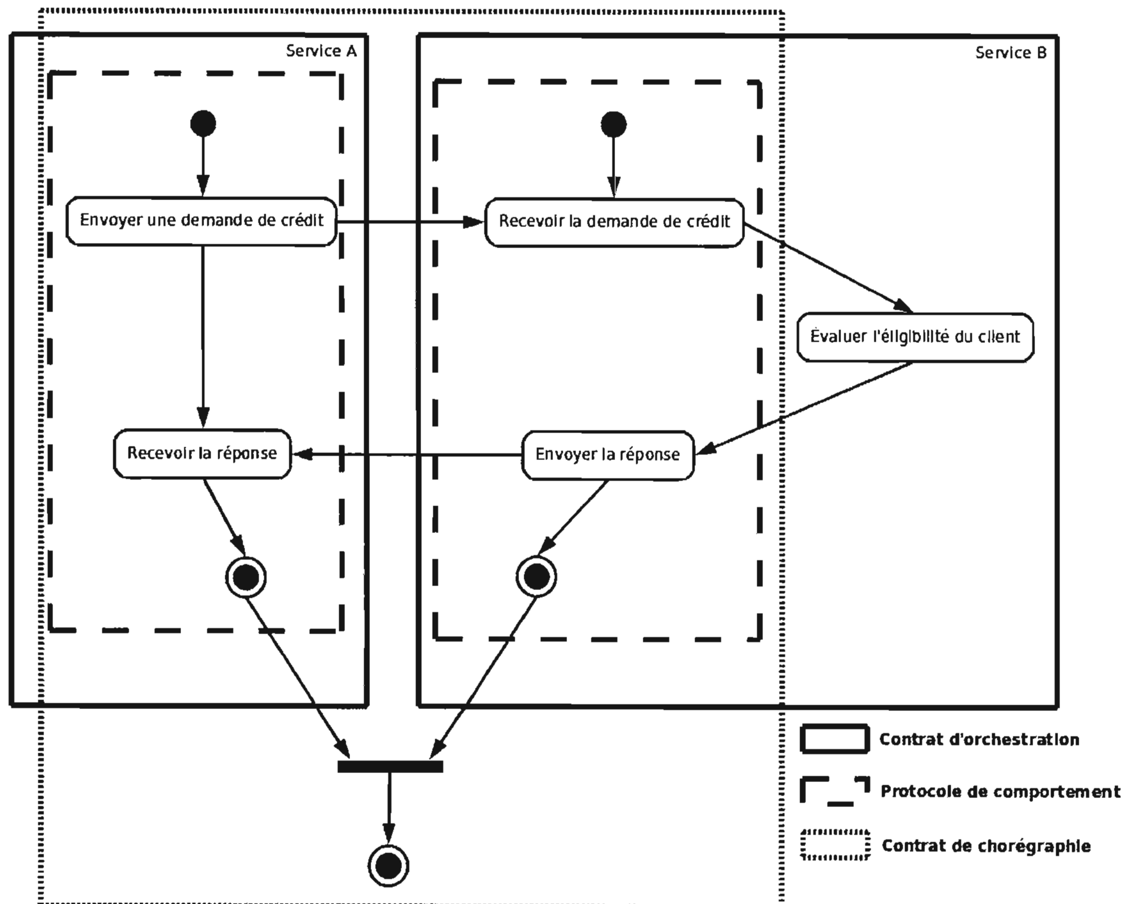

FIGURE 2.1 Relation entre contrats de chorégraphie, contrats d'orchestration et protocoles de comportement.

2.1.2 Contrats de qualité de service

Les contrats de qualité de service permettent de différencier entre des services offrant les mêmes fonctionnalités, en se basant sur leurs propriétés non fonctionnelles telles que la performance, la disponibilité, la fiabilité ou le coût — une liste plus exhaustive de ces attributs est fournie par Maximilien et Singh (34).

La surveillance de la qualité de service se divise généralement en trois phases (31) :

1. la définition des contraintes de qualité de service.
2. la surveillance à l'exécution des contraintes définies et de la conformité de leurs valeurs avec ce qui est exigé dans les contrats.
3. la réaction en cas de violation d'une exigence.

Pour réaliser cela, WSLA (19) se base sur la définition de contrats de niveau de service (*Service Level Agreement*) (62) entre le fournisseur et le demandeur du service. Par contre, WSOL définit les contraintes de qualité au niveau des classes de services, de la même manière qu'il le fait pour les exigences fonctionnelles et pour les exigences d'authentifications. Ainsi, pour WSLA, l'offre de service correspond à un accord entre le fournisseur et le demandeur du service, et pour WSOL, elle fait partie de la définition même du service.

2.1.3 Contrats d'infrastructures

Les contrats d'infrastructures, comme WSDL dans sa partie concrète, peuvent servir à définir les protocoles de communication utilisés pour encapsuler (par exemple SOAP) et transporter (par exemple HTTP) les messages entre services Web ou, dans le cas de WS-Policy (7), à définir des politiques de sécurité. Ils servent aussi à définir des politiques de gestion des services Web, ce qui permet de spécifier les entités responsables de la surveillance de l'exécution des services incluant souvent le contrôle des contraintes définies dans les autres contrats.

2.2 Vérification dynamique des contrats de services Web

Plusieurs projets de recherche ont apporté des réponses à la problématique de surveillance dynamique de contrats de service Web. Ces approches se distinguent aussi bien par la nature des contrats qu'elles supervisent que par leurs mises en œuvre de la surveillance dynamique. Dans

cette section, nous présentons quelques-uns de ces projets :

- Dynamo, présenté dans la section précédente, est une version orientée aspect de l'engin d'exécution ActiveBPEL (23). Cette plateforme repose sur deux langages :
  - WSCoL, qui permet de définir des exigences fonctionnelles en annotant des processus BPEL par des pré/post-conditions. Ces annotations sont transformées par un pré-processeur en code BPEL supplémentaire, ce qui enrichira le code original et permettra de le surveiller. Le pré-processeur transforme les activités concernées par de telles annotations de manière à assurer un appel au module de surveillance en lui passant les variables concernées par les annotations (9).
  - WSReL, qui permet de mettre en œuvre des stratégies de recouvrement enclenchées lors de la violation des contraintes associées au processus (11).
- WSOI (*Web Service Offerings Infrastructure*) est une extension du serveur Apache Axis (32). Cette solution rend possible la surveillance des contraintes WSOL présentées dans la section précédente. Dans cette approche, la surveillance est effectuée par de « tierces parties » (56), c'est-à-dire des modules indépendants du client et du fournisseur du service. Ces modules interceptent les messages SOAP échangés par les différents intervenants et analysent leur contenu pour vérifier leur conformité avec les exigences fonctionnelles et de qualité de service définies dans le contrat surveillé.
- ASTRO est un ensemble d'outils utilisables dans diverses phases du cycle de vie des compositions de services Web (45). Son module de surveillance dynamique WS-mon (44) permet de superviser des propriétés temporelles, statistiques et booléennes applicables aussi bien à l'instance en cours d'exécution qu'à l'ensemble des instances exécutées (8). Ces propriétés sont utilisées pour générer des programmes Java déployés dans l'environnement d'exécution pour intercepter les messages échangés par les processus surveillés (8). Les messages interceptés sont comparés aux propriétés prédéfinies, ce qui permet de détecter l'exécution des événements et des comportements surveillés (8).
- BP-Suite (14) est un outil de gestion de processus BPEL, basé sur des systèmes de requêtes (13) qui permettent d'interroger la spécification du processus, de surveiller son déroulement et d'analyser ses journaux d'exécution. Le module qui nous intéresse est BP-Mon (*Business Processes Monitoring*) (14), lequel est responsable de la surveillance dynamique. Pour cela, il dispose d'un langage de requête graphique permettant de définir les schémas d'exécution à surveiller. Ces définitions sont utilisées pour générer un processus BPEL indépendant, qui surveille les instances en exécution.

- Le cadre d'application présenté par Gan et al. (26) permet de surveiller des protocoles d'échanges entre services Web, spécifiés visuellement par des diagrammes de séquences UML. Les diagrammes de séquences sont transformés en automates ce qui permet de les confronter aux événements détectés pendant l'exécution des processus. Cette démarche est sans doute celle qui s'approche le plus de notre proposition.

Conclusion

Dans ce chapitre, nous avons essayé de présenter une classification des contrats de services Web, parmi lesquels nous retiendrons les protocoles de comportement, c'est-à-dire, le type de contrats fonctionnels qui sera traité dans notre approche. Aussi, nous avons vu comment d'autres projets de recherches ont essayé d'adresser des problématiques semblables. Dans le prochain chapitre, c'est de notre propre solution de surveillance dynamique de protocoles de comportement qu'il sera question.

CHAPITRE III

SURVEILLANCE DYNAMIQUE DES PROTOCOLES DE COMPORTEMENT

Le présent chapitre est consacré à notre proposition pour un cadre d'applications (*framework*) permettant la construction de systèmes de surveillance dynamique de protocoles de comportement associés à des services BPEL. Nous commencerons par présenter les exigences du cadre d'application, en second lieu nous détaillerons son modèle du domaine, par la suite nous expliquerons la logique sur laquelle se base le système de surveillance dynamique pour finir avec une proposition pour l'architecture d'un tel système.

3.1 Exigences pour un cadre d'application de surveillance dynamique

Notre cadre d'applications devra essentiellement répondre aux deux exigences fonctionnelles suivantes :

- permettre la définition de protocoles de comportement associés à des processus BPEL;
- réagir en cours d'exécution des processus BPEL en fonction du respect ou non des contrats définis.

En terme d'exigences non fonctionnelles il devra :

- être portable, en fonctionnant indépendamment du choix de l'engin d'exécution;
- être générique, en étant capables d'interagir avec des représentations différentes d'événements et d'activités BPEL;
- être adaptable, en offrant la possibilité d'intégrer des modules externes de compensation.

3.2 Instances de processus et événements BPEL

Un processus BPEL est composé de plusieurs activités. Il est physiquement représenté par un fichier XML destiné à être exécuté par un engin d'exécution. On parlera alors d'instance de processus. Une instance est caractérisée par un flux d'événements, où chaque événement correspond à l'exécution d'une activité du processus. Les liens entre processus (`Process`), activités (`Activity`), instances de processus (`ProcessInstance`) et événements (`Event`) sont représentés dans l'extrait du modèle du domaine de la figure 3.1 [1].

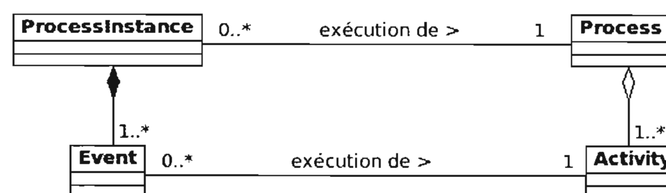

FIGURE 3.1 Extrait du modèle du domaine : instances de processus et événement BPEL.

3.3 Protocoles de comportement

Un protocole de comportement (voir aussi section 2.1.1) est un ordonnancement de certaines opérations visibles de l'extérieur d'une composition de services ; on peut également parler d'activités de comportement (*behavior activity*). Dans le cas des processus BPEL, ces activités peuvent être identifiées par :

- l'opération invoquée (`operation`) ;
- la « structure de la relation » (4) (`partnerLink`) qui lie le service au partenaire sollicité ;
- le schéma de communication (`exchangePattern`) entre les deux services. Ce dernier peut avoir trois formes :
  - entrant (`in`) : quand l'activité est une réception de message (cas des activités BPEL `<receive>` et `<onMessage>`) ;
  - sortant (`out`) : quand l'activité est une invocation d'un service asynchrone (cas de l'activité BPEL `<invoke>` quand l'attribut `outputVariable` n'est pas défini) ou quand elle constitue un envoi de réponse (cas de l'activité BPEL `<reply>`) ;
  - appel synchrone (`outIn`) : quand l'activité est une invocation d'un service synchrone (cas de l'activité BPEL `<invoke>` quand l'attribut `outputVariable` est spécifié).

1. L'ensemble du modèle du domaine est présenté dans l'annexe A, page 56.

```
<invoke name="InvokeSeller"
    partnerLink="SellersRepresentative"
    operation="biddingSeller"
    xmlns:tns="http://simplesellersrepresentative.bpm.com/"
    portType="tns:SimpleSellersRepresentative"
    inputVariable="BiddingSellerIn"
    outputVariable="BiddingSellerOut"/>
```

Listage 3.1 Code source d'une activité BPEL.

Pour illustrer cela, soit l'activité BPEL dont le code source est présenté dans le listage 3.1 et qui exprime une invocation synchrone de l'opération `biddingSeller` à partir d'un service dont la lien est définit par le `partnerLink SellersRepresentative`. Cette activité pourra être représentée, de façon abstraite et simplifiée, comme suit :
`outIn:biddingSeller@SellersRepresentative`

Nous considérons qu'un protocole de comportement est représentable statiquement par une expression régulière ou un automate auquel on associerait un alphabet d'activités. La figure 3.2 montre un extrait du modèle du domaine représentant les relations entre le protocole de comportement (`BehaviorProtocol`), l'automate (`Automaton`) qui le constitue et l'alphabet d'activités (`MonitoredActivities`) auquel il est associé[2].

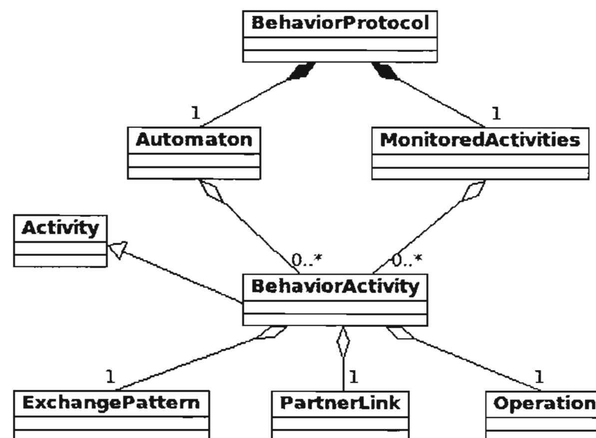

FIGURE 3.2 Extrait du modèle du domaine : protocoles de comportement.

2. L'ensemble du modèle du domaine est présenté dans l'annexe A, page 56.

3.4 Surveillance dynamique

La représentation des protocoles de comportement par des automates nous permettra de vérifier dynamiquement si les processus BPEL en exécution respectent ou non les protocoles auxquels ils sont associés. Nous utiliserons des automates finis déterministes dont les transitions et les états correspondront respectivement aux activités BPEL surveillées et aux états des protocoles associés. Ainsi, en parcourant l'automate au fur et à mesure de l'exécution du processus, on saura :

- qu'un contrat a démarré quand l'automate correspondant sera dans son état initial;
- qu'un contrat a avancé correctement quand l'automate correspondant sera dans un état non-final;
- qu'un contrat s'est achevé avec succès quand l'automate correspondant sera dans un état final;
- qu'un contrat a échoué quand l'automate correspondant rencontrera une erreur.

La figure 3.3 montre une visualisation graphique d'un exemple de protocole de comportement et la figure 3.4 montre comment ce même protocole pourrait être représenté par un automate[3].

La surveillance dynamique est liée à la capacité de déceler l'exécution des événements BPEL au niveau de l'engin. À chaque fois que le système détectera l'exécution d'un événement il procèdera comme suit :

1. il récupérera l'activité dont l'événement détecté est une instance;
2. il vérifiera si l'activité récupérée fait partie du protocole surveillé;
3. si c'est le cas, il passera l'activité à l'automate associé au protocole;
4. il réagira en fonction du nouvel état de l'automate :
   - si l'automate est dans son état initial, le système déclenchera l'action associée au démarrage du protocole;
   - si l'automate est dans un état non final, le système déclenchera l'action associée à l'avancement du protocole;
   - si l'automate est dans son état d'acceptation, le système déclenchera l'action associée à la réalisation du protocole;
   - si l'automate rencontre une erreur (l'activité ne permet aucune transition possible à

3. L'étude de cas dont sont issus ces deux exemples sera détaillée dans la section 5.3, page 46.

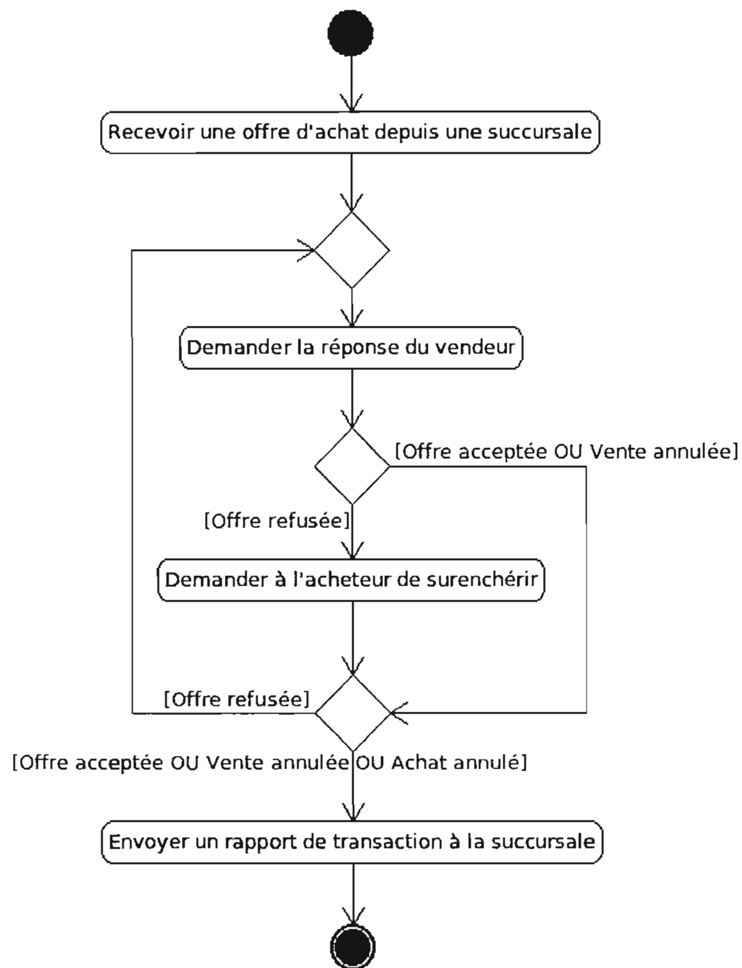

FIGURE 3.3 Représentation graphique (diagramme d'activités) d'un exemple de protocole de comportement.

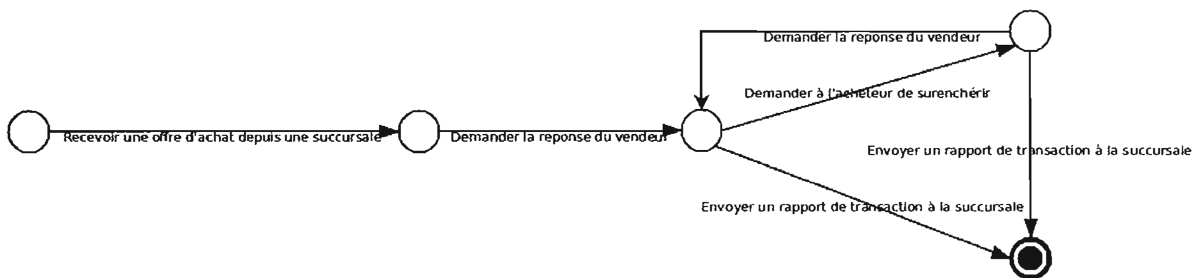

FIGURE 3.4 Représentation en automate d'un exemple de protocole de comportement.

partir de l'état courant), le système déclenchera l'action associée à l'échec du protocole.

3.5 Architecture

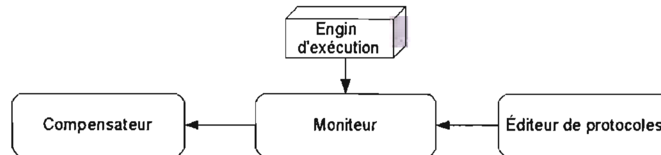

FIGURE 3.5 Architecture : première décomposition fonctionnelle.

Une première décomposition fonctionnelle possible, illustrée par la figure 3.5, nous permet de distinguer trois modules essentiels :

- l'éditeur de protocoles ;
- le moniteur qui aura :
  - à détecter l'exécution des événements au niveau de l'engin ;
  - à surveiller dynamiquement la satisfaction des protocoles ;
- le compensateur, qui devra réagir en fonction du respect ou non des protocoles surveillés.

Toutefois, une telle organisation rendrait le module moniteur dépendant de l'engin d'exécution, ce qui entre en contradiction avec notre exigence de portabilité (voir section 3.1, page 18). Pour atténuer cette dépendance, nous avons ajouté un autre niveau de composition, en isolant la partie du moniteur qui dépend de l'engin d'exécution à savoir celle responsable de la détection des événements. Ainsi, pour pouvoir utiliser notre solution avec un engin d'exécution en particulier, il faudra d'abord développer un module de détection d'événements approprié à ce dernier. Par conséquent, notre architecture sera composée :

- d'un module éditeur de protocole ;
- d'un module détecteur d'événements, spécifique à un engin d'exécution ;
- d'un module moniteur ;
- d'un module compensateur.

Cette nouvelle structuration est illustrée par la figure 3.6.

Conclusion

Ce chapitre nous a permis d'expliciter notre proposition pour un cadre d'applications de surveillance dynamique de protocoles de comportement, en termes de représentation de concepts liés aux processus BPEL, en termes de logique de traitement, mais aussi en termes de choix

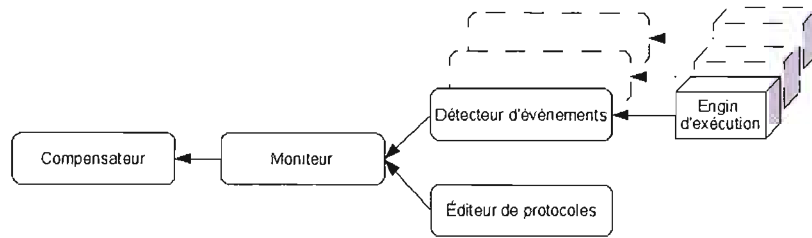

FIGURE 3.6 Architecture : deuxième décomposition fonctionnelle.

architecturaux, le tout en respectant les exigences définies en préambule. À présent, nous allons détailler la mise en œuvre de la solution proposée.

Chapitre IV

BPEL.RPM : UNE MISE EN OEUVRE D'UN SYSTÈME DE SURVEILLANCE DYNAMIQUE DE PROTOCOLES DE COMPORTEMENT

Nous appelons *BPEL Runtime Protocol Monitor* (BPEL.RPM) notre preuve de concept de la solution présentée dans le chapitre précédent. Cette mise en œuvre respecte la décomposition fonctionnelle décrite en amont (section 3.4) et illustrée par la figure 3.5. La figure 4.2 reprend quant-à-elle l'organisation en paquetages de BPEL.RPM. On obtient ainsi les correspondances suivantes :

- au module détecteur d'événements correspond le paquetage `eventhandler` ;
- au module moniteur correspond le paquetage `protocolmonitor` ;
- au module compensateur correspond le paquetage `compensator`.

Dans l'implémentation actuelle, il n'y a pas d'éditeur de protocoles. Nous utilisons des fichiers XML, édités manuellement et dont la syntaxe sera expliquée dans la section suivante. Dans le reste du chapitre, nous détaillerons la mise en œuvre des trois modules constituant BPEL.RPM, cités dans le paragraphe précédent.

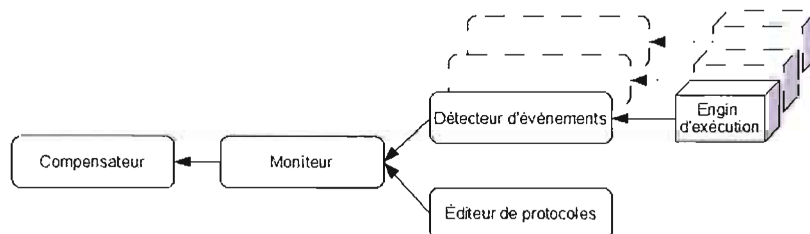

FIGURE 4.1 Architecture : Décomposition fonctionnelle.

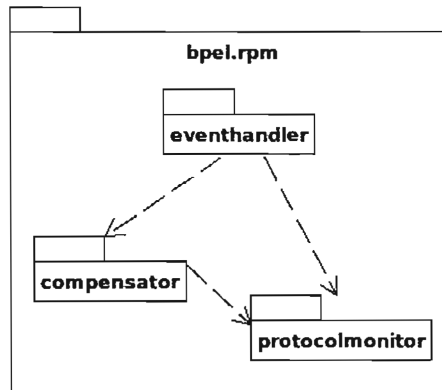

FIGURE 4.2 Diagramme de paquetage de BPEL.RPM.

4.1 Formalisme et édition de protocoles

Dans BPEL.RPM, les protocoles sont définis dans un fichier XML et, comme le montre le listage 4.1, les deux éléments principaux de cette représentation sont les suivants :

- `<MonitoredActivities>`, l'alphabet du protocole, c'est-à-dire l'ensemble des activités surveillées par ce dernier ; [1]
- `<RegExp>`, l'expression régulière exprimant l'ordonnancement des activités ;

La syntaxe utilisée pour les expressions régulières permettant la définition des protocoles de comportement est présentée dans le tableau 4.1. Elle est inspirée :

- de la représentation d'une activité de comportement dans le modèle du domaine, présenté dans le chapitre précédent (section 3.2, page 19) ;
- de la syntaxe utilisée par l'API `dk.brics.automaton` (40) [2] [3] pour décrire des expressions régulières ;

---

1. Nous avons décidé d'exprimer explicitement l'ensemble des activités surveillées dans une section à part du protocole pour des raisons d'intelligibilité. L'implémentation actuelle n'a pas besoin de cette information et peut extraire l'ensemble des activités automatiquement de l'expression régulière. Par contre, de telles indications sont pertinentes dans le cas où on aurait besoin de surveiller certaines activités non présentes dans l'expression régulière.

2. BPEL.RPM utilise l'API `dk.brics.automaton` pour gérer les automates associés aux protocoles surveillés.

3. L'expression de concurrence ne faisant pas partie des opérations sur les expressions régulières gérées par l'API `dk.brics.automaton`, celle-ci peut être remplacée par des séquences d'entrelacements (39).

```
<Process>
    <ProcessName>SimpleAuctionService</ProcessName>
    <Protocol>
        <ProtocolName>Global</ProtocolName>
        <MonitoredActivities>
            <Activity>
                in:SimpleAuctionServiceOperation@AuctionBranch
            </Activity>
            <Activity>
                outIn:biddingSeller@SellersRepresentative
            </Activity>
            <Activity>
                outIn:getBuyerOffer@BuyersRepresentative
            </Activity>
            <Activity>
                out:SimpleAuctionServiceOperation@AuctionBranch
            </Activity>
        <MonitoredActivities/>
        <RegExp>
            in:SimpleAuctionServiceOperation@AuctionBranch ;
            (outIn:biddingSeller@SellersRepresentative ;
            outIn:getBuyerOffer@BuyersRepresentative ?)+
            out:SimpleAuctionServiceOperation@AuctionBranch
        </RegExp>
    </Protocol>
</Process>
```

Listage 4.1 Exemple d'un protocole de comportement pour BPEL.RPM.

- des expressions de chemins (*path expressions*) (58) qui permettent de décrire des ordonnancements d'opérations.

```
========================================================
Soit a et b, deux activités BPEL de comportement :
--------------------------------------------------------
a ; b  | exécution de l'activité a, suivie de celle de l'activité b.
a | b  | exécution de l'activité a, ou bien celle de l'activité b.
a [] b | exécution en parallèle des activités a et b.
a ?    | zéro ou une occurrence de l'activité a.
a *    | zéro ou plusieurs occurrences de l'activité a.
a +    | une ou plusieurs occurrences de l'activité a.
a {n}  | n occurrences de l'activité a.
========================================================
```

TABLE 4.1 Syntaxe abstraite des expressions régulières utilisées dans BPEL.RPM.

4.2 Module de détection des événements

La mise en œuvre du détecteur d'événements dépend du choix de la plateforme d'exécution. Dans notre preuve de concept, nous interagissons avec l'intergiciel Open ESB et son engin d'exécution BPEL SE [4]. BPEL SE permet d'enregistrer dynamiquement des informations sur les instances BPEL en cours d'exécution (35). Ces informations sont sauvegardées dans `bpelseDB`, une base de données associée à l'engin d'exécution. C'est justement cette caractéristique que nous avons exploitée pour implémenter notre détecteur d'événements. Comme le montre la figure 4.3, le module de détection d'événements est composé, dans sa partie intégrée à l'environnement d'exécution, d'un déclencheur (*trigger*) associé à la base de données `bpelseDB` et qui, à chaque fois que celle-ci enregistrera l'exécution d'un événement BPEL, lancera une fonction stockée (*User-Defined Function* ou *UDF*). Cette routine transmettra à travers une interface de connexion (*socket*) toutes les informations liées à l'exécution de l'événement. Du côté de BPEL.RPM, à chaque fois que le serveur recevra les informations émises par la partie cliente, il les passera aux autres modules responsables de la supervision des protocoles.

---

4. Pour plus de détails sur cette plateforme, se référer à la section 1.1.1, page 7.

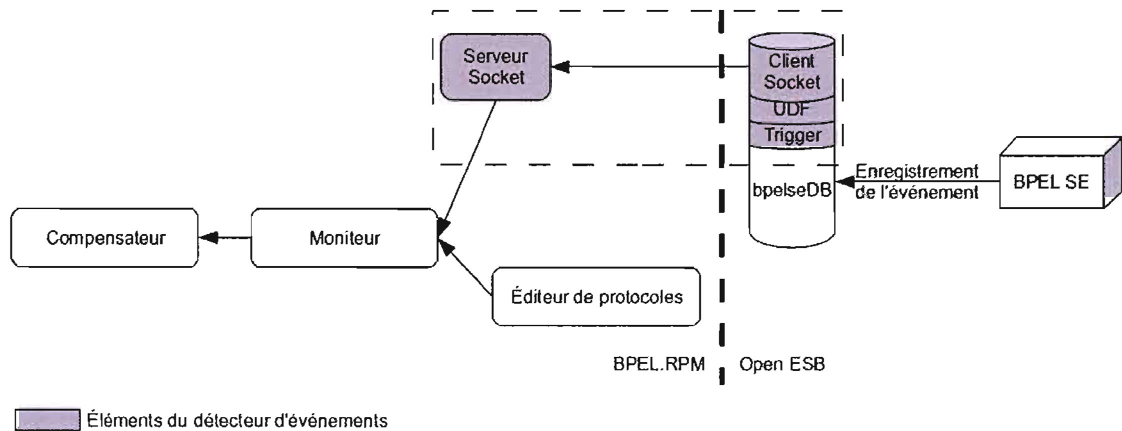

FIGURE 4.3 Architecture du module de détection d'événements associé à BPEL.RPM.

4.3 Module de surveillance dynamique

Le module de surveillance dynamique implémenté au sein du paquetage `protocolmonitor` est l'élément le plus important de BPEL.RPM puisque, comme le montre la figure 4.2, il sera utilisé par les autres parties du système. Il a les responsabilités suivantes :

- représenter dynamiquement les protocoles de comportement sous forme d'automates ;
- réagir en cours d'exécution aux événements signalés par le module de détection en s'assurant que les traces d'exécution partielles soient conformes aux automates précédemment générés.

Cette partie du système a été implémentée de manière à pouvoir assurer une généricité :

- en entrée, puisqu'elle pourra être associée à des modules de détection conçus pour différents engins d'exécutions ;
- en sortie, puisqu'elle pourra interagir avec différents modules de compensation.

4.3.1 Représentation dynamique des protocoles

Durant l'exécution de l'application, les protocoles exprimés physiquement par des fichiers XML contenant des expressions régulières d'activités seront représentés dynamiquement par une association de deux structures :

- un automate fini déterministe ;
- une table de correspondance associant chaque symbole d'une transition à l'activité qu'il représente.

La figure 4.4 et le tableau 4.2 représentent respectivement un exemple d'automate[5] et un exemple de table de correspondance[6], tous deux produits à partir du même protocole[7].

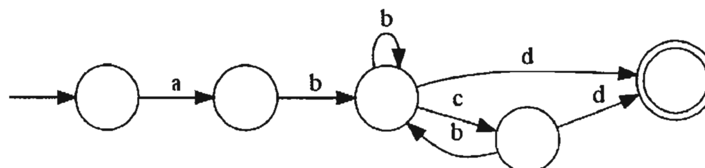

FIGURE 4.4 Protocole représenté par un automate fini déterministe.

```
=======================================================
RegExp
-------------------------------------------------------
a(bc?)+d
=======================================================
Identifier | Activity
-------------------------------------------------------
a          | in:SimpleAuctionServiceOperation@AuctionBranch
b          | outIn:biddingSeller@SellersRepresentative
c          | outIn:getBuyerOffer@BuyersRepresentative
d          | out:SimpleAuctionServiceOperation@AuctionBranch
=======================================================
```

TABLE 4.2 Exemple d'une table de correspondance d'un protocole de comportement.

4.3.2 Surveillance dynamique

Pour garantir une surveillance dynamique, le module doit être capable de réagir au fur et à mesure que l'unité de détection lui signale une nouvelle exécution d'événement. Pour mettre en œuvre ce comportement, nous avons implémenté le patron de conception observateur (25).

5. La figure représentant l'automate est générée par l'outil Graphviz (1) à partir d'un extrait du journal de BPEL.RPM.

6. La table de correspondance est extraite du journal de BPEL.RPM.

7. L'étude de cas dont sont issus ces deux exemples sera détaillée à la section 5.3, page 46.

Ainsi, dès qu'une exécution d'événement est signalée par le module de détection :

- les instances de la classe observée `Monitor` associées au processus en exécution reçoivent l'objet représentant l'événement ;
- chaque instance récupère l'activité dont l'événement détecté est une instance, puis vérifie si l'activité récupérée fait partie du protocole qu'elle surveille ;
- si c'est le cas :
  - l'événement est ajouté à la trace d'exécution partielle ;
  - une instance de message composée de la séquence de messages qui se sont produits depuis le démarrage de l'instance et de l'événement déclencheur est créée ;
  - l'activité extraite de l'événement détecté est passée à l'automate ;
  - en fonction du nouvel état de l'automate, tous les objets implémentant `MonitorListener` (l'interface de l'observateur) et qui sont enregistrés en tant qu'écouteurs auprès de l'instance en action de la classe `Monitor` reçoivent une notification avec le message précédemment créé :
    - si l'automate est dans son état initial, c'est l'implémentation de `protocolStarted( MonitoringMessage monitoringMessage )` qui est sollicitée signalant le démarrage du protocole surveillé ;
    - si l'automate est dans un état non final, c'est l'implémentation de `protocolStepped( MonitoringMessage monitoringMessage )` qui est sollicitée signalant l'avancement du protocole surveillé ;
    - si l'automate est dans son état final, c'est l'implémentation de `protocolCompleted( MonitoringMessage monitoringMessage )` qui est sollicitée signalant la réalisation du protocole surveillé ;
    - si l'automate rencontre une erreur — l'activité ne permet aucune transition possible à partir de l'état courant —, c'est l'implémentation de `protocolFaulted( MonitoringMessage monitoringMessage )` qui est sollicitée signalant l'échec du protocole surveillé.

Le diagramme de séquence de la figure 4.5 résume cette logique de traitement.

4.3.3 Connexion aux modules de détection

Pour qu'un module de détection puisse s'intégrer à BPEL.RPM, il devra au préalable étendre les représentations abstraites d'un événement et d'une activité du module de surveillance. Ces représentations sont définies par les classes abstraites `AbstractActivity` et `AbstractEvent`.

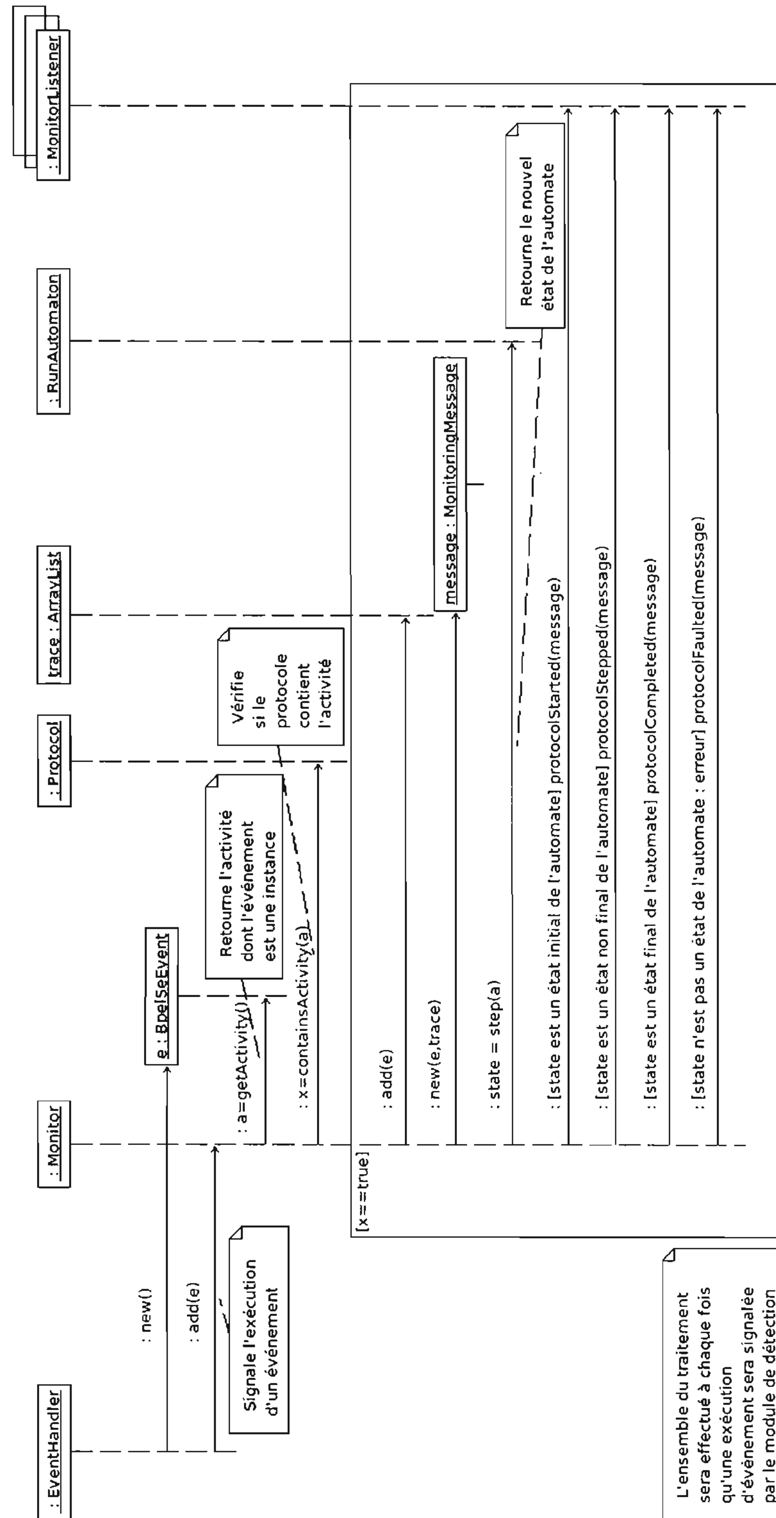

FIGURE 4.5 Diagramme de séquence de la logique de surveillance dynamique de BPEL.RPM.

Ceci assure en partie :

- la portabilité de l'application qui pourra interagir avec tout engin d'exécution doté d'un module de détection étendant les représentations abstraites d'une activité et d'un événement ;
- la généricité du module de surveillance qui pourra accepter des événements et des activités hétérogènes.

L'extrait simplifié du diagramme de classes de la figure 4.6 illustre ces propriétés dans l'implémentation actuelle du système[8]. Les classes `BehavorActivity` et `BPELSeEvent` définissent respectivement les représentations concrètes d'une activité comportementale et de son exécution au niveau de l'engin BPEL SE. En faisant de ces classes des sous-types des représentations abstraites d'un événement et d'une activité du module de surveillance, le détecteur d'événements pourra en soumettre les instances au module de surveillance sans briser la logique du traitement.

Dans la prochaine section, nous verrons comment cette exigence de généricité a été implémentée pour le module de compensation.

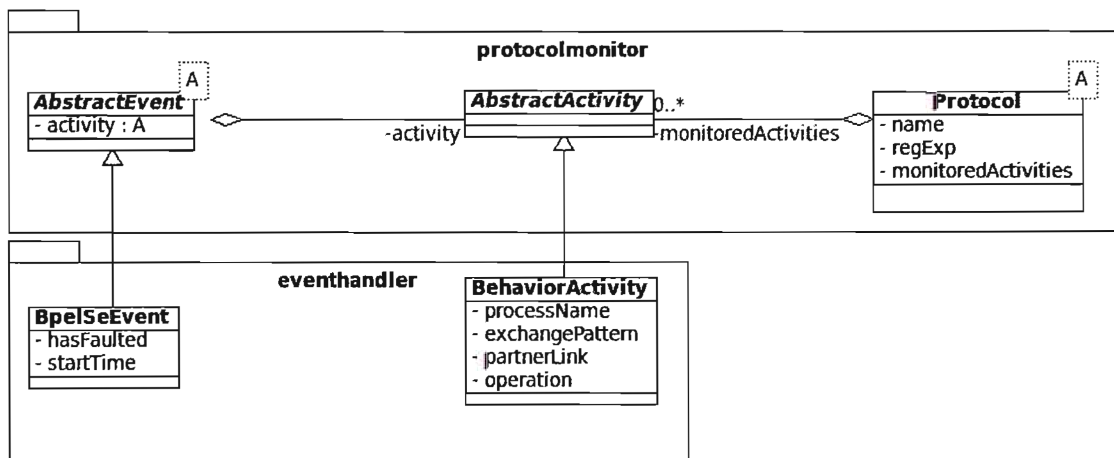

FIGURE 4.6 Extrait du diagramme de classes de BPEL.RPM : relation entre le module de détection d'événements et celui de surveillance dynamique.

4.4 Module de compensation

Le module de compensation est celui qui encapsule les différentes stratégies de réaction face au respect ou non des protocoles surveillés. Dans notre cas, on distingue quatre types de

8. L'ensemble du diagramme de classes est présenté dans l'annexe B, page 57.

réponses différenciés par leurs moments de déclenchement :

- au démarrage d'un protocole ;
- lors de l'avancement d'un protocole ;
- à la fin d'un protocole ;
- lors de l'échec d'un protocole.

Pour mettre en place ces différentes stratégies il suffira d'implémenter l'interface `MonitorListener` en définissant respectivement les quatre méthodes suivantes :

- `protocolStarted( MonitoringMessage monitoringMessage )`
- `protocolStepped( MonitoringMessage monitoringMessage )`
- `protocolCompleted( MonitoringMessage monitoringMessage )`
- `protocolFaulted( MonitoringMessage monitoringMessage )`

Afin d'avoir plus de souplesse, il est possible de passer par un adaptateur (25) ce qui permettra de choisir parmi une ou plusieurs stratégies de compensation à implémenter.

L'extrait simplifié du diagramme de classes de la figure 4.7 nous montre les relations entre le module de compensation et celui de surveillance dynamique[9]. Dans notre preuve de concept, les stratégies de réaction ont été mise en œuvre dans la classe `ConcreteMonitorListener`[10].

Conclusion

Nous avons donc vu, au cours de ce chapitre, comment les différents modules d'un système de surveillance dynamique de protocoles de comportement ont été mis en œuvre dans BPEL.RPM. L'architecture modulaire obtenue nous a permis de créer un système facilement adaptable pour différents engins d'exécution et différents mécanismes de compensation.

Nous verrons dans le prochain chapitre comment BPEL.RPM se comporte, et ce à travers trois études de cas.

9. L'ensemble du diagramme de classes est présenté dans l'annexe B, page 57.

10. La logique de compensation ne faisant pas partie de notre recherche, nous nous sommes contentés dans notre preuve de concept de traiter cet aspect de façon rudimentaire, à savoir simplement afficher un résultat.

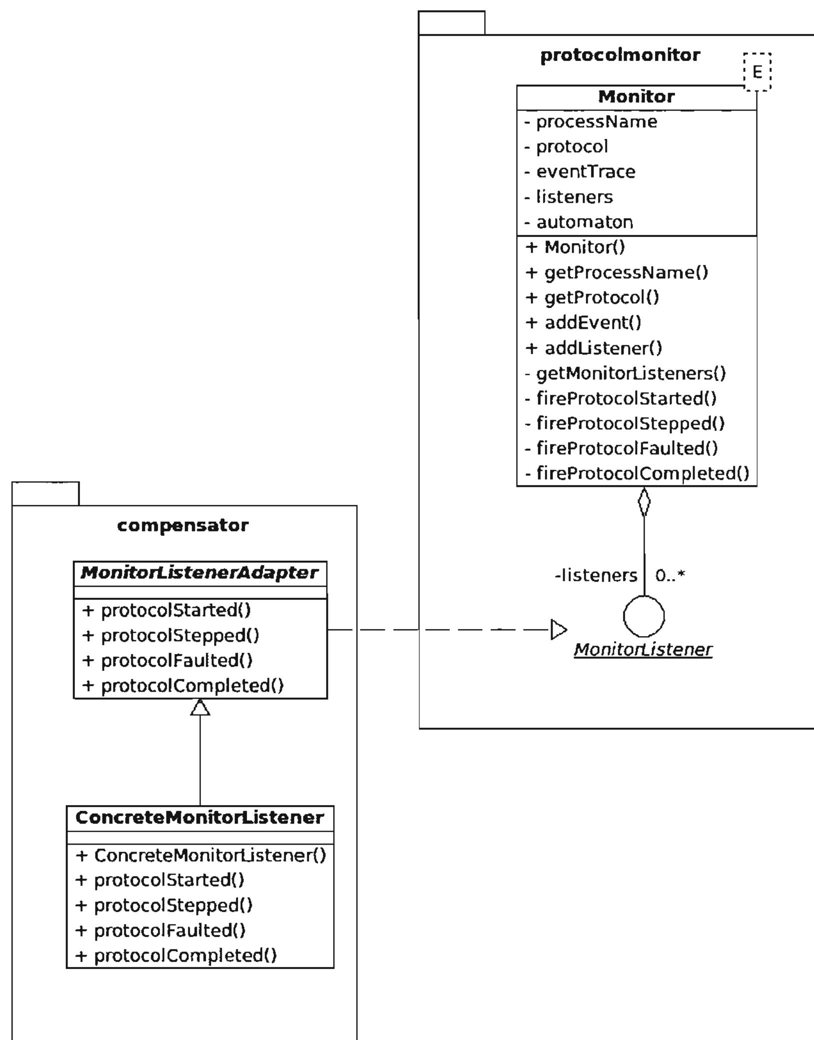

FIGURE 4.7 Extrait du diagramme de classes de BPEL.RPM : relation entre le module de compensation et celui de surveillance dynamique.

CHAPITRE V

ÉTUDES DE CAS

Dans ce chapitre nous commencerons par récapituler le mécanisme d'utilisation de BPEL.RPM, par la suite nous examinerons son comportement par rapport à trois processus BPEL. Nous présenterons, à chaque fois, la logique métier du service testé et celles des protocoles surveillés. Par la suite, à partir d'extraits de journaux système[1], nous montrerons comment BPEL.RPM représente en interne les protocoles surveillés. Enfin, nous commenterons les messages affichés dynamiquement par BPEL.RPM pour chacun des protocoles surveillés.

5.1 Utilisation de BPEL.RPM

La figure 5.1 résume le processus d'utilisation de BPEL.RPM et les flux de données qui en résultent. L'utilisateur commence par déployer le service BPEL dans l'environnement d'exécution. Il fournit, en même temps, à BPEL.RPM les fichiers XML contenant les protocoles de comportement à surveiller. Ceux-ci sont transformés en automates grâce à l'API `dk.brics.automaton`. Au fur et à mesure du déroulement du processus BPEL, le module de détection signale l'exécution des événements au composant responsable de la surveillance dynamique. Ce dernier, disposant à la fois des automates représentant les protocoles et des signalements d'exécution d'événements, peut déterminer si l'instance en cours d'exécution respecte ou non le contrat qui lui est assigné. L'information obtenue est dès lors transmise au module de réaction qui l'intègre aux messages affichés à l'utilisateur.

---

[1]Les automates présentés dans ce chapitre sont générés par l'outil Graphviz (1) à partir d'extraits de journaux de BPEL.RPM et les tables de correspondance sont directement extraites des mêmes journaux.

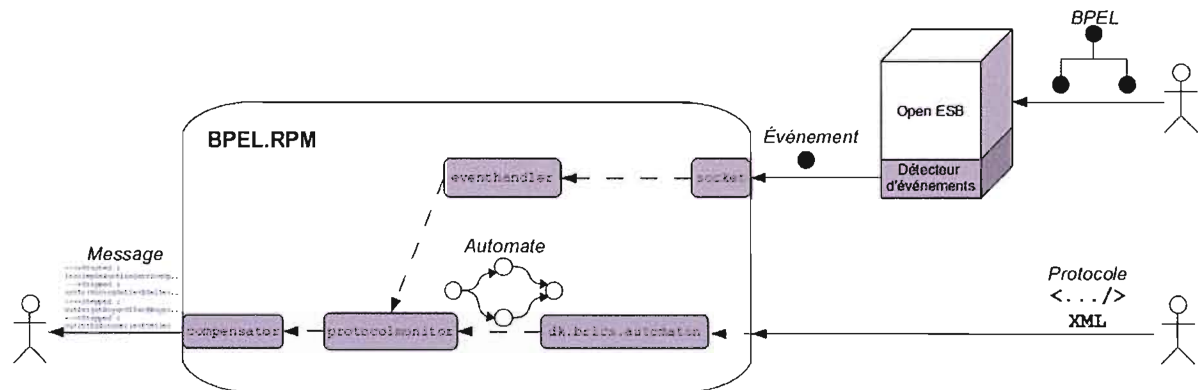

FIGURE 5.1 Processus d'utilisation de BPEL.RPM.

5.2 Service de traitement de demandes de prêt

L'exemple d'un service de traitement de demandes de prêt (*loan processing service*) (12) que nous présentons fait partie de la documentation en ligne de l'environnement de développement NetBeans. Dans cette étude de cas, un client fait une demande de crédit qui sera refusée ou acceptée en fonction de l'analyse des informations fournies. Cet examen est effectué par un service externe de traitement.

La figure 5.2 montre la représentation que fait l'éditeur NetBeans du code source BPEL. On y voit que le service interagit avec deux partenaires :

- `EjbImplementation`, qui fournit les informations de crédits et qui recevra la réponse finale ;
- `BpelImplementation`, qui, une fois sollicité, analysera les informations de crédit et décidera d'accorder ou non le prêt demandé.

La figure 5.3 et le listage 5.1 constituent respectivement une représentation graphique du protocole surveillé ainsi que son code source. On voit les activités suivantes :

- réception de message ;
- appel à un service synchrone ;
- envoi de réponse.

Ce protocole représente une vision globale du processus de demandes de prêt. En ce sens, il aurait pu être défini par le concepteur du service pour servir de référentiel à l'implémentation.

BPEL.RPM représentera ce protocole par un automate et une table de correspondance, illustrés respectivement par la figure 5.4 et la table 5.1. Pendant l'exécution, comme le montre le

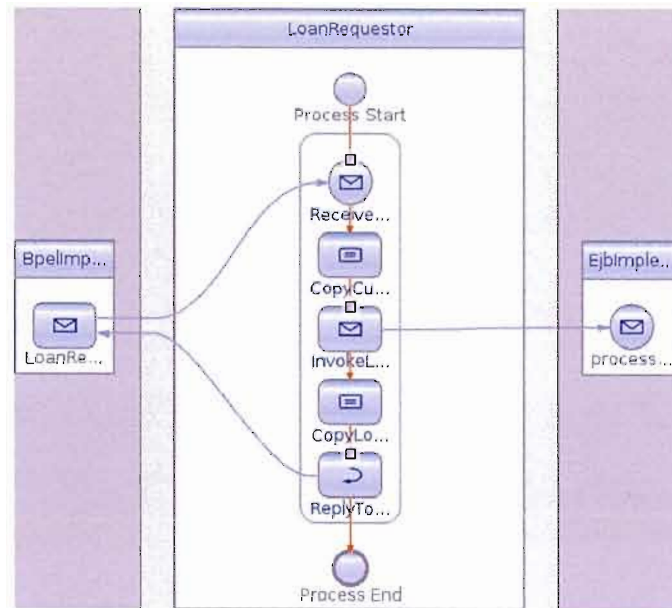

FIGURE 5.2 Service de traitement de demandes de prêt.

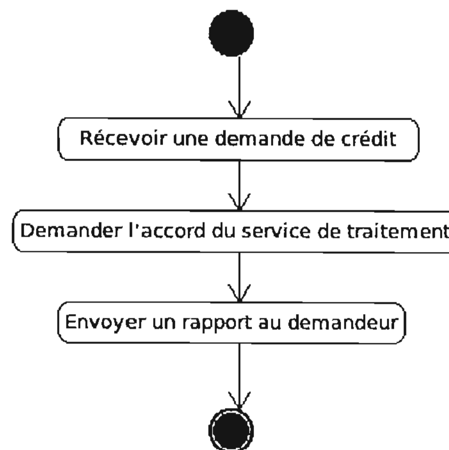

FIGURE 5.3 Service de traitement de demandes de prêt : protocole "Global".

```
<Process>
    <ProcessName>LoanRequestor</ProcessName>
    <Protocol>
        <ProtocolName>Global</ProtocolName>
        <MonitoredActivities>
            <Activity>
                in:LoanRequestorOperation@BpelImplementation
            </Activity>
            <Activity>
                outIn:processApplicOperation@EjbImplementation
            </Activity>
            <Activity>
                out:LoanRequestorOperation@BpelImplementation
            </Activity>
        <MonitoredActivities/>
        <RegExp>
            in:LoanRequestorOperation@BpelImplementation ;
            outIn:processApplicOperation@EjbImplementation ;
            out:LoanRequestorOperation@BpelImplementation
        </RegExp>
    </Protocol>
</Process>
```

Listage 5.1 Service de traitement de demandes de prêt : définition du protocole "Global".

listage 5.2, des messages seront affichés en fonction du respect ou nom du protocole par l'instance en exécution :

- le message de la ligne 3 signale le démarrage du protocole ;
- le message de la ligne 4 signale l'avancement du protocole ;
- le message de la ligne 5 signale l'achèvement du protocole.

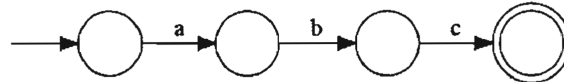

FIGURE 5.4 Service de traitement de demandes de prêt : automate représentant le protocole "Global".

```
========================================================
RegExp
--------------------------------------------------------
abc
========================================================
Identifier | Activity
--------------------------------------------------------
a          | in:LoanRequestorOperation@BpelImplementation
b          | outIn:processApplicOperation@EjbImplementation
c          | out:LoanRequestorOperation@BpelImplementation
========================================================
```

TABLE 5.1 Service de traitement de demandes de prêt : table de correspondance du protocole "Global".

5.3 Service de réservation de voyage

Le service de réservation de voyage (*travel reservation service*) (29) fait lui aussi partie de la documentation en ligne de l'environnement de développement NetBeans. Il est néanmoins plus complexe que l'étude de cas précédente puisque'il se compose de plusieurs traitements conditionnels et interagit avec trois autres services partenaires. Cette étude de cas sera enclenchée par la réception d'une partie de l'itinéraire de voyage d'un client. En fonction de l'état de l'itinéraire, le service fera appel à zéro ou plusieurs autres services :

```
1  LoanRequestor
2      Global
3      --->Started : in:LoanRequestorOperation@BpelImplementation succeed
           at 2010-03-08 17:50:53.851  *
4      --->Stepped : outIn:processApplicOperation@EjbImplementation succeed
            at 2010-03-08 17:50:56.101
5      --->Completed : out:LoanRequestorOperation@BpelImplementation succeed
             at 2010-03-08 17:50:58.24  :)
```

Listage 5.2 Service de traitement de demandes de prêt : messages de surveillance du protocole "Global".

- si l'itinéraire fourni par le client ne comporte pas de réservation de vol, un service partenaire de réservation de vol sera sollicité ;
- si l'itinéraire fourni par le client ne comporte pas de réservation de véhicule, un service partenaire de location de voiture sera sollicité ;
- si l'itinéraire fourni par le client ne comporte pas de réservation de chambre d'hôtel, un autre service partenaire de réservation de chambre d'hôtel sera sollicité.

La figure 5.5 montre la représentation que fait l'éditeur NetBeans du code source BPEL. On y voit que le service interagit avec plusieurs partenaires :

- `Travel`, qui fournit l'itinéraire de voyage partiel du client et qui, à la fin, recevra le nouvel itinéraire complété par les services partenaires ;
- `Airline`, un service de réservation de vol ;
- `Vehicle`, un service de location de voiture ;
- `Hotel`, un service de réservation de chambre d'hôtel.

Le protocole surveillé dans le cadre de cet exemple décrit une situation particulière, à savoir celle d'un client qui n'a aucune réservation au préalable et qui, donc, devrait solliciter les trois services offerts :

- la réservation de vol ;
- la réservation de véhicule ;
- la réservation de chambre d'hôtel.

La figure 5.6 est une illustration de ce protocole ; quant au code source, il est présenté dans le listage 5.3. Au démarrage de l'application, le protocole sera représenté par l'automate de la figure 5.7 et la table de correspondance 5.2.

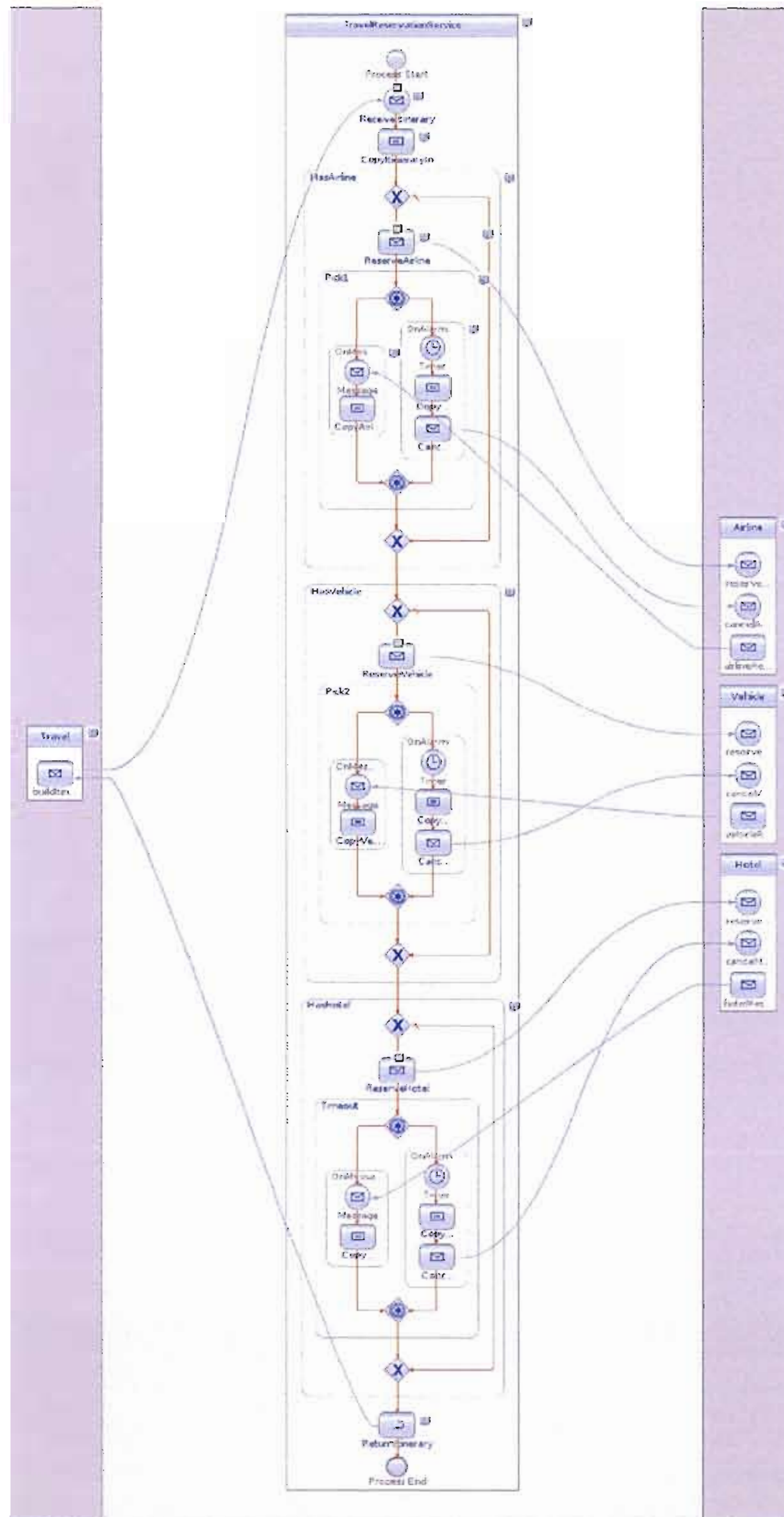

FIGURE 5.5 Service de réservation de voyage.

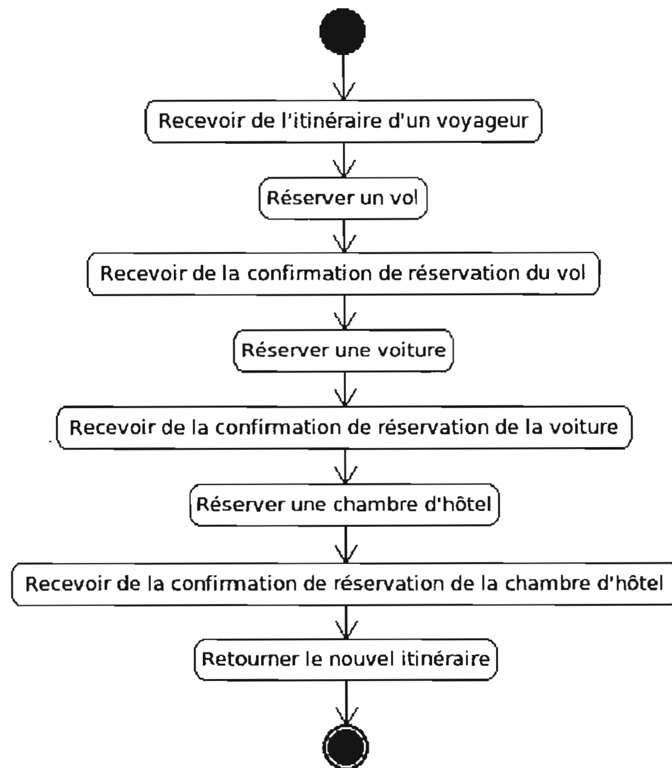

FIGURE 5.6 Service de réservation de voyage : protocole "Has no reservations".

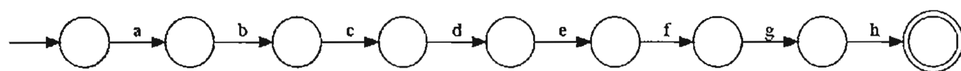

FIGURE 5.7 Service de réservation de voyage : automate représentant le protocole "Has no reservations".

```xml
<Process>
    <ProcessName>TravelReservationService</ProcessName>
    <Protocol>
        <ProtocolName>Has no reservations</ProtocolName>
            <Activity>in:buildItinerary@Travel</Activity>
            <Activity>out:reserveAirline@Airline</Activity>
            <Activity>in:airlineReserved@Airline</Activity>
            <Activity>out:reserveVehicle@Vehicle</Activity>
            <Activity>in:vehicleReserved@Vehicle</Activity>
            <Activity>out:reserveHotel@Hotel</Activity>
            <Activity>in:hotelReserved@Hotel</Activity>
            <Activity>out:buildItinerary@Travel</Activity>
        <RegExp>
            in:buildItinerary@Travel ;
            out:reserveAirline@Airline ;
            in:airlineReserved@Airline ;
            out:reserveVehicle@Vehicle ;
            in:vehicleReserved@Vehicle ;
            out:reserveHotel@Hotel ;
            in:hotelReserved@Hotel ;
            out:buildItinerary@Travel
        </RegExp>
    </Protocol>
</Process>
```

Listage 5.3 Service de réservation de voyage : définition du protocole "Has no reservations".

```
=======================================================
RegExp
-------------------------------------------------------
abcdefgh
=======================================================
Identifier | Activity
-------------------------------------------------------
a          | in:buildItinerary@Travel
b          | out:reserveAirline@Airline
c          | in:airlineReserved@Airline
d          | out:reserveVehicle@Vehicle
e          | in:vehicleReserved@Vehicle
f          | out:reserveHotel@Hotel
g          | in:hotelReserved@Hotel
h          | out:buildItinerary@Travel
=======================================================
```

TABLE 5.2 Service de réservation de voyage : table de correspondance du protocole "Has no reservations".

Les listages 5.4 et 5.5 reproduisent les messages affichés par l'application lors de deux exécutions différentes. La première est enclenchée par la réception d'un itinéraire vierge, c'est-à-dire ne comportant aucune réservation. Dans ce cas le listage des notifications de surveillance 5.4 nous fournit les informations suivantes :

- le message de la ligne 3 signale le démarrage du protocole ;
- les messages des lignes 4 à 9 signalent l'avancement normal du protocole, correspondant aux trois réservations manquant à l'itinéraire reçu ;
- le message de la ligne 10 signale la réalisation du protocole.

Par contre, la deuxième exécution a été enclenchée par la réception d'un itinéraire comportant une réservation de chambre d'hôtel. Ce cas n'est donc pas conforme au protocole surveillé et c'est ce qui transparaît dans les messages du listage 5.5 où on voit que :

- le message de la ligne 3 signale le démarrage du protocole ;
- les messages des lignes 4 à 7 signalent l'avancement normal du protocole, correspondant aux deux réservations manquantes à l'itinéraire reçu ;
- le message de la ligne 8 signale l'échec du protocole, du fait que le comportement attendu était l'appel au service de réservation d'hôtel, qui n'a pas eu lieu parce que l'itinéraire reçu comportait déjà une telle réservation.

5.4 Service de vente aux enchères

Cette étude de cas se base sur un service de vente aux enchères qui se déclenche à la suite de la réception d'une offre d'achat. Cette dernière sera proposée au vendeur, et si elle est refusée il sera demandé à l'acheteur de surenchérir. Tant que la nouvelle offre n'est pas acceptée ou que la vente n'est pas annulée, le processus continuera de proposer aux acheteurs potentiels de surenchérir et de présenter leurs offres au vendeur. À la fin de la transaction, un rapport sera envoyé à la succursale ayant initié le processus.

Le principal intérêt que représente cet exemple dans le cadre des expérimentations de BPEL.RPM est qu'il comporte des traitements itératifs. En effet, et comme le montre la figure 5.8, les appels aux services interagissant avec les acheteurs et les vendeurs se trouvent dans une boucle tant que (`while`). On voit aussi dans cette représentation que fait l'éditeur NetBeans du code source BPEL [2] que le service interagit avec les partenaires suivants :

- `AuctionBranch`, succursale initiatrice de la transaction ;

2. Le code source est présenté dans l'annexe C, page 58.

```
1  TravelReservationService
2      Has no reservations
3      --->Started : in:buildItinerary@Travel succeed  at 2010-03-09
           11:04:48.79  *
4      --->Stepped : out:reserveAirline@Airline succeed  at 2010-03-09
           11:04:51.076
5      --->Stepped : in:airlineReserved@Airline succeed  at 2010-03-09
           11:04:52.437
6      --->Stepped : out:reserveVehicle@Vehicle succeed  at 2010-03-09
           11:04:53.61
7      --->Stepped : in:vehicleReserved@Vehicle succeed  at 2010-03-09
           11:04:53.735
8      --->Stepped : out:reserveHotel@Hotel succeed  at 2010-03-09
           11:04:53.896
9      --->Stepped : in:hotelReserved@Hotel succeed  at 2010-03-09
           11:04:54.076
10     --->Completed : out:buildItinerary@Travel succeed  at 2010-03-09
           11:04:54.236  :)
```

Listage 5.4 Service de réservation de voyage : messages de surveillance du protocole "Has no reservations".

```
1 TravelReservationService
2     Has no reservations
3     --->Started : in:buildItinerary@Travel succeed  at 2010-03-09
          11:35:39.616  *
4     --->Stepped : out:reserveAirline@Airline succeed  at 2010-03-09
          11:35:39.722
5     --->Stepped : in:airlineReserved@Airline succeed  at 2010-03-09
          11:35:39.866
6     --->Stepped : out:reserveVehicle@Vehicle succeed  at 2010-03-09
          11:35:40.024
7     --->Stepped : in:vehicleReserved@Vehicle succeed  at 2010-03-09
          11:35:40.078
8     --->Faulted : out:buildItinerary@Travel succeed  at 2010-03-09
          11:35:40.209  :(
```

Listage 5.5 Service de réservation de voyage : messages de surveillance du protocole "Has no reservations" — cas d'un itinéraire non vierge.

- `SellersRepresentative`, représentant les vendeurs ;
- `BuyersRepresentative`, représentant les acheteurs ;

Ici, et comme pour la première étude de cas, le protocole surveillé donne une vision globale du service. Il est représenté dans la figure 5.9 et son code source est reproduit dans le listage 5.6. Au démarrage de BPEL.RPM le protocole sera représenté par l'automate de la figure 5.10 et la table de correspondance 5.3.

Le listage 5.7 reprend les messages affichés pendant le deroulement d'une instance particulière du processus, nécessitant plusieurs communications entre le service d'achat et celui de vente. On y voit que :

- les messages des lignes 3 et 16 signalent respectivement le démarrage du protocole et son achèvement avec succès ;
- les messages des lignes 4, 6, 8, 10, 12 et 14 signalent l'exécution de plusieurs occurrences de `outIn:biddingSeller@SellersRepresentative`, c'est-à-dire l'appel au service synchrone de proposition d'offre d'achat au vendeur ;
- les messages des lignes 5, 7, 8, 11, 13 et 15 signalent l'exécution de plusieurs occurrences de `outIn:getBuyerOffer@BuyersRepresentative`, c'est-à-dire l'appel au service syn-

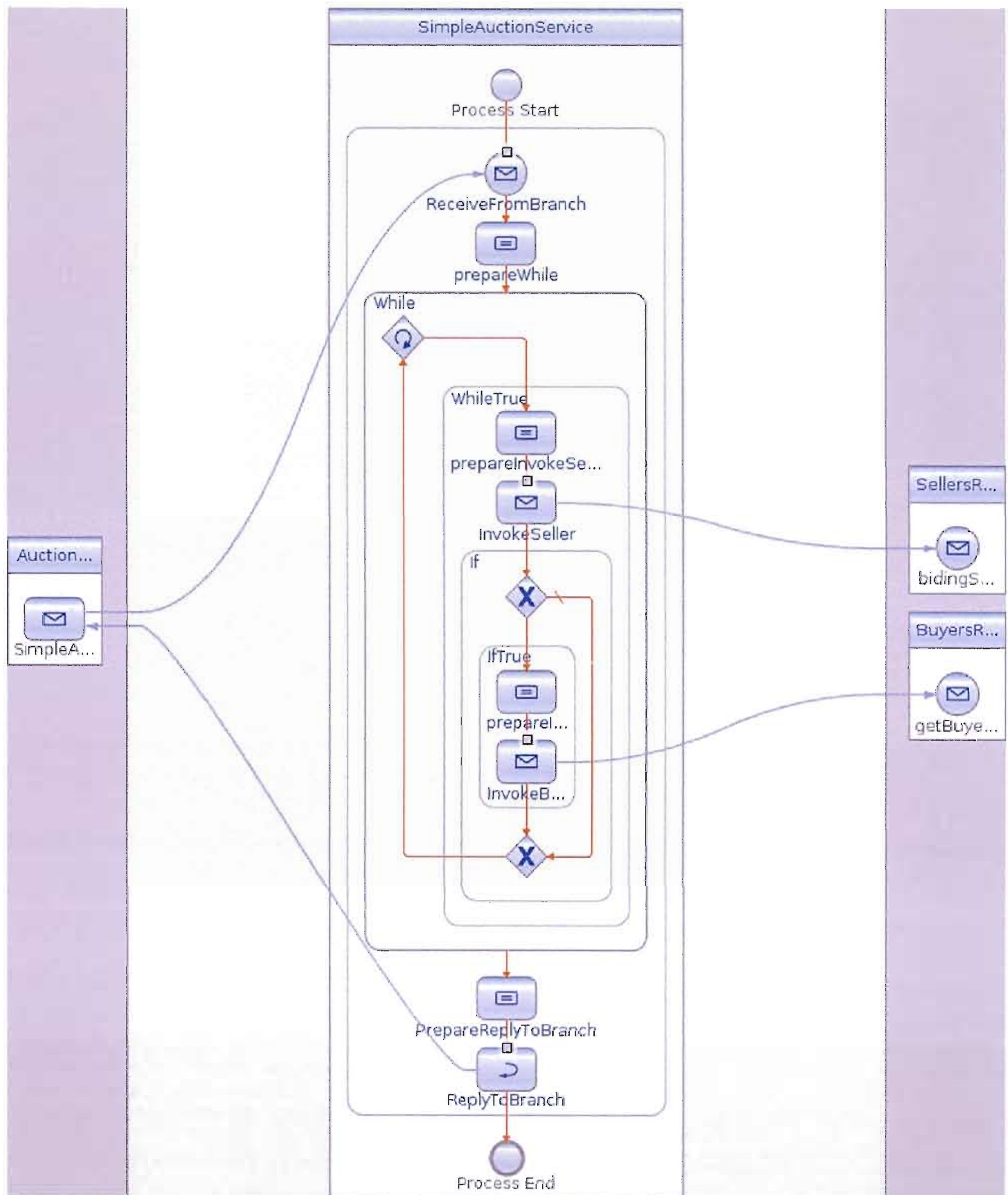

FIGURE 5.8 Service de vente aux enchères.

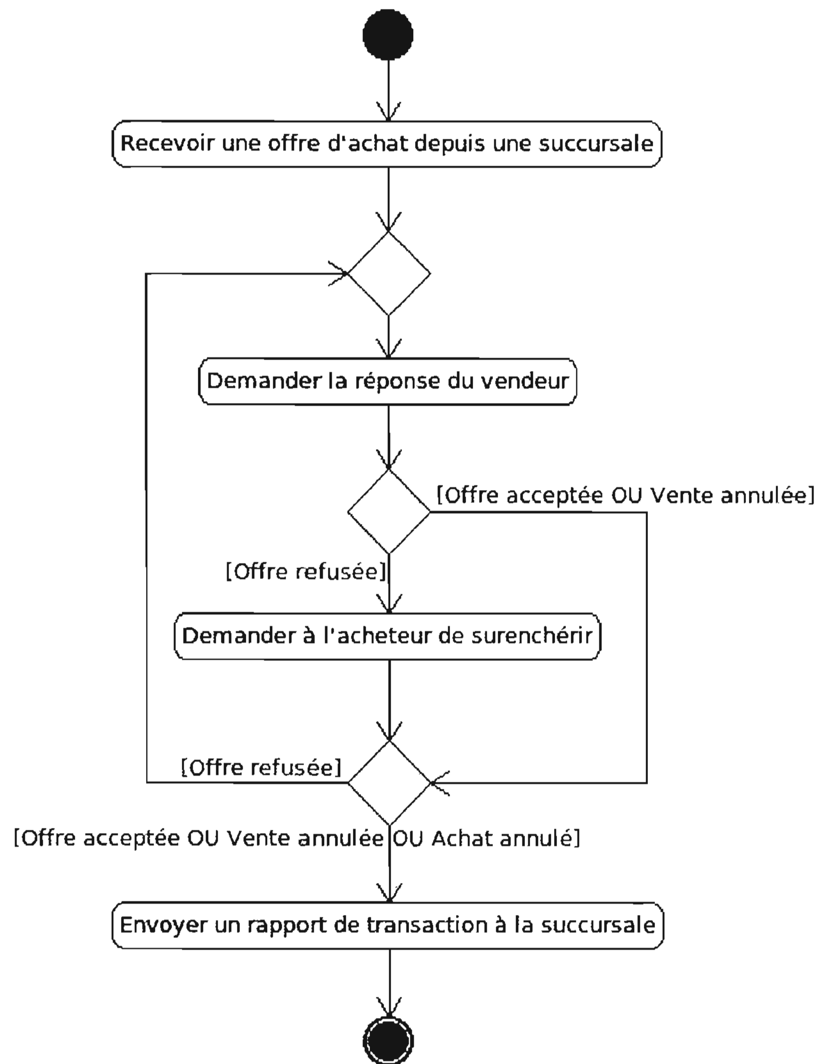

FIGURE 5.9 Service de vente aux enchères : protocole "Global".

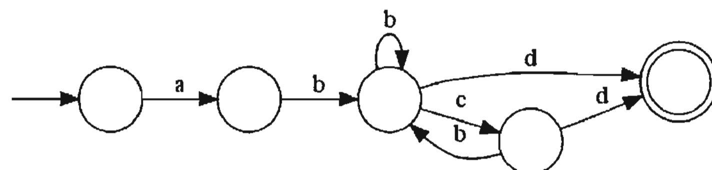

FIGURE 5.10 Service de vente aux enchères : automate représentant le protocole "Global".

```
<Process>
    <ProcessName>SimpleAuctionService</ProcessName>
    <Protocol>
        <ProtocolName>Global</ProtocolName>
        <MonitoredActivities>
            <Activity>
                in:SimpleAuctionServiceOperation@AuctionBranch
            </Activity>
            <Activity>
                outIn:biddingSeller@SellersRepresentative
            </Activity>
            <Activity>
                outIn:getBuyerOffer@BuyersRepresentative
            </Activity>
            <Activity>
                out:SimpleAuctionServiceOperation@AuctionBranch
            </Activity>
        <MonitoredActivities/>
        <RegExp>
            in:SimpleAuctionServiceOperation@AuctionBranch ;
            (outIn:biddingSeller@SellersRepresentative ;
            outIn:getBuyerOffer@BuyersRepresentative ?)+
            out:SimpleAuctionServiceOperation@AuctionBranch
        </RegExp>
    </Protocol>
</Process>
```

Listage 5.6 Service de vente aux enchères : définition du protocole "Global".

```
========================================================
RegExp
--------------------------------------------------------
a(bc?)+d
========================================================
Identifier | Activity
--------------------------------------------------------
a          | in:SimpleAuctionServiceOperation@AuctionBranch
b          | outIn:biddingSeller@SellersRepresentative
c          | outIn:getBuyerOffer@BuyersRepresentative
d          | out:SimpleAuctionServiceOperation@AuctionBranch
========================================================
```

TABLE 5.3 Service de vente aux enchères : table de correspondance du protocole "Global".

chrone de demande de surenchère faite aux clients.

```
1  SimpleAuctionService
2      Global
3      --->Started : in:SimpleAuctionServiceOperation@AuctionBranch succeed
           at 2010-03-10 17:19:11.859  *
4      --->Stepped : outIn:biddingSeller@SellersRepresentative succeed  at
          2010-03-10 17:19:13.234
5      --->Stepped : outIn:getBuyerOffer@BuyersRepresentative succeed  at
          2010-03-10 17:19:14.46
6      --->Stepped : outIn:biddingSeller@SellersRepresentative succeed  at
          2010-03-10 17:19:15.039
7      --->Stepped : outIn:getBuyerOffer@BuyersRepresentative succeed  at
          2010-03-10 17:19:15.419
8      --->Stepped : outIn:biddingSeller@SellersRepresentative succeed  at
          2010-03-10 17:19:15.64
9      --->Stepped : outIn:getBuyerOffer@BuyersRepresentative succeed  at
          2010-03-10 17:19:15.812
10     --->Stepped : outIn:biddingSeller@SellersRepresentative succeed  at
          2010-03-10 17:19:15.971
11     --->Stepped : outIn:getBuyerOffer@BuyersRepresentative succeed  at
          2010-03-10 17:19:16.125
12     --->Stepped : outIn:biddingSeller@SellersRepresentative succeed  at
          2010-03-10 17:19:16.349
13     --->Stepped : outIn:getBuyerOffer@BuyersRepresentative succeed  at
          2010-03-10 17:19:16.516
14     --->Stepped : outIn:biddingSeller@SellersRepresentative succeed  at
          2010-03-10 17:19:16.645
15     --->Stepped : outIn:getBuyerOffer@BuyersRepresentative succeed  at
          2010-03-10 17:19:16.77
16     --->Completed : out:SimpleAuctionServiceOperation@AuctionBranch
          succeed  at 2010-03-10 17:19:16.952  :)
```

Listage 5.7 Service de vente aux enchères : messages de surveillance du protocole "Global".

CONCLUSION

Nous avons proposé dans notre travail un modèle pour un cadre d'applications permettant la définition et la surveillance de protocoles de comportement associés à des compositions de services Web BPEL. Plus précisément notre modèle permet d'assurer une détection dynamique des éventuelles ruptures de contrats, dans le but de déclencher des mécanismes de compensation à chaud, c'est-à-dire, en cours d'exécution des processus supervisés. Le dynamisme du modèle a été réalisé par l'utilisation combinée d'automates finis, déterministes — dont les transitions et les états correspondent respectivement aux activités surveillées et aux états des protocoles associés — et la détection de l'exécution des événements BPEL au niveau de l'engin d'exécution.

Dans notre travail nous avons également élaboré BPEL.RPM, une mise en œuvre de notre modèle, qui est adaptable, dans le sens où elle peut aisément intégrer des modules externes de compensation, mais qui est aussi portable, puisqu'elle fonctionne indépendamment de l'environnement d'exécution des services Web. Afin d'assurer une telle portabilité, il nous a fallu séparer le module de détection d'événements du reste du système. Dès lors, pour pouvoir utiliser notre solution avec un environnement en particulier, il nous a fallu tout d'abord développer un module de détection d'événements approprié à ce dernier en prenant comme champ d'expérimentation la plateforme OpenESB.

Nous estimons néanmoins que plusieurs améliorations peuvent êtres apportées aussi bien au modèle proposé dans ce mémoire qu'à la mise en œuvre qui en a été faite. Nous pensons particulièrement aux point suivants :

- élargir la notion de contrat pour qu'elle puisse englober les stratégies de compensation ;
- modifier et étendre la syntaxe abstraite des expressions régulières utilisées pour définir les protocoles de comportement dans BPEL.RPM vers un langage dédié (*domain-specific language*) plus approprié à BPEL. Une des pistes envisageables est de baser le langage sur la syntaxe des processus abstraits de BPEL (4; 54) ;
- développer des modules de détections d'événements BPEL pour des plateformes d'exécution de compositions de services Web autres qu'OpenESB ;
- développer d'autres modules de compensation qui ne se contenteraient plus d'afficher des

messages à l'utilisateur, mais qui exécuteraient de réelles stratégies de recouvrement. Dans le cadre de la plateforme OpenESB, l'interfaçage avec l'API `BPELManagement Service` (50) pourrait faciliter cela.

Autant de pistes que nous espérons pouvoir explorer dans des travaux futurs.

APPENDICE A

MODÈLE DU DOMAINE DE NOTRE SYSTÈME DE SURVEILLANCE DYNAMIQUE DE PROTOCOLES DE COMPORTEMENT

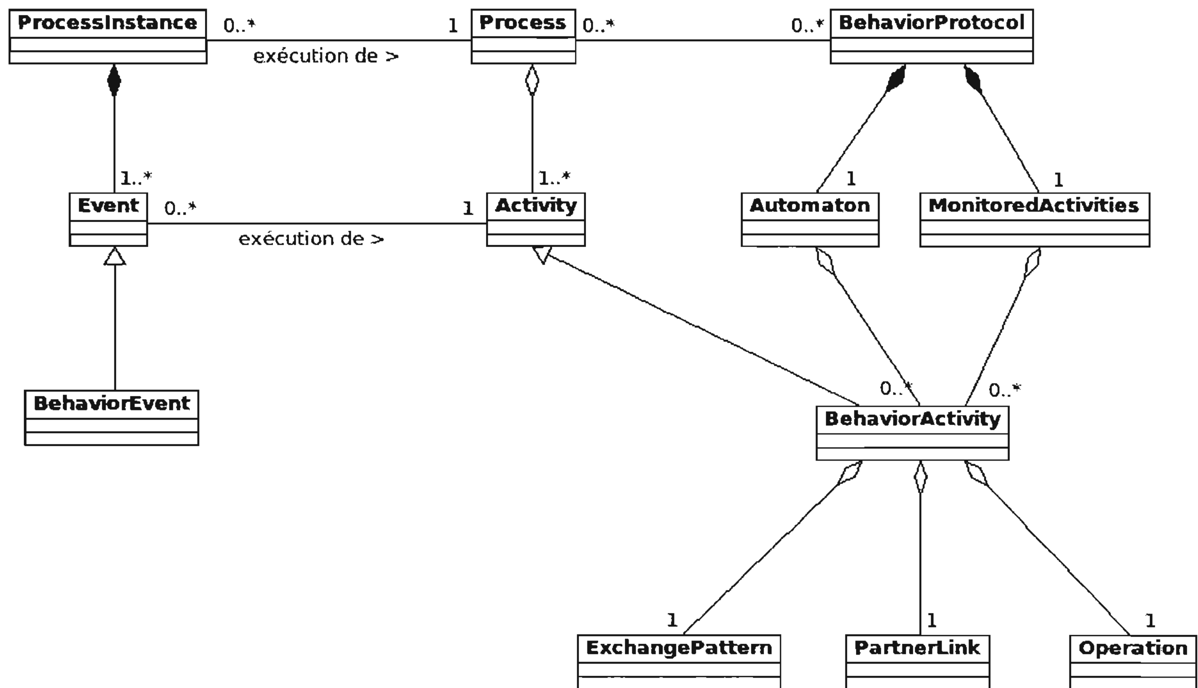

Figure A.1 Modèle du domaine de notre système de surveillance dynamique de protocoles de comportement.

APPENDICE B

DIAGRAMME DE CLASSES DE BPEL.RPM

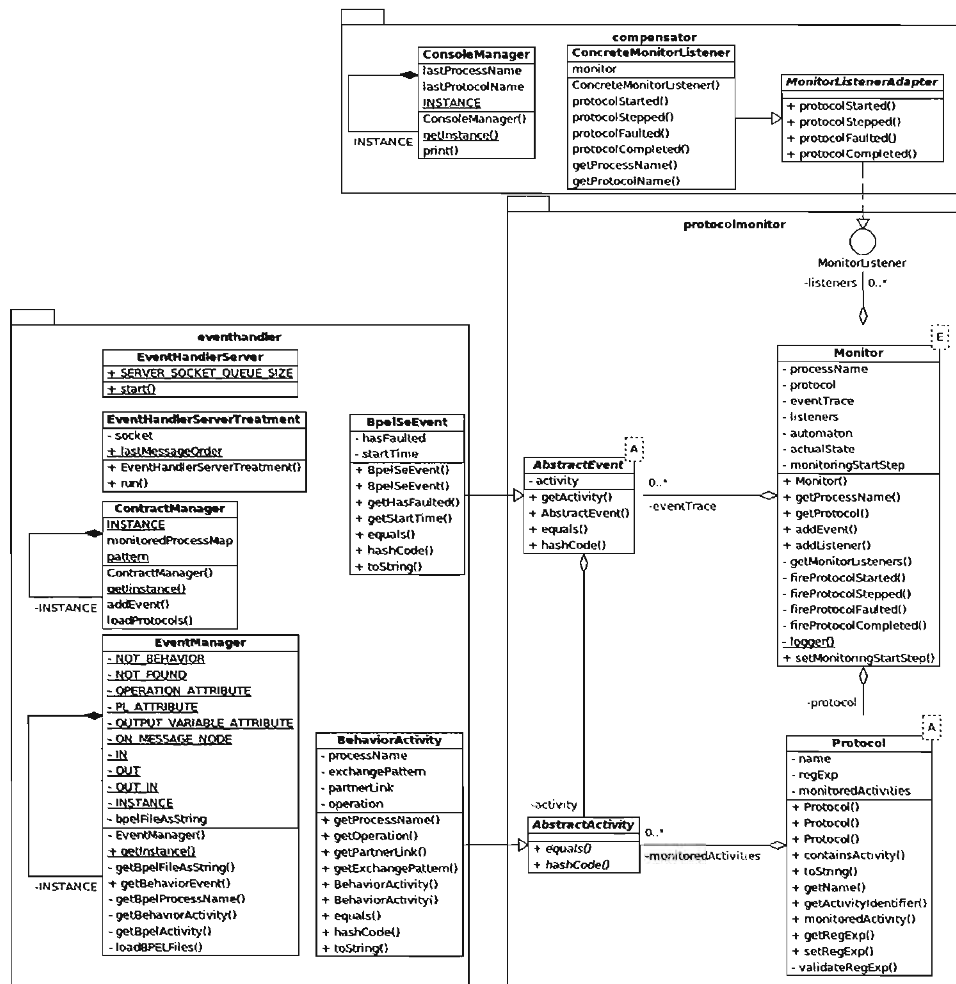

Figure B.1 Diagramme de classes de BPEL.RPM.

APPENDICE C

CODE SOURCE BPEL D'UN SERVICE DE VENTE AUX ENCHÈRES

Nous présentons ici le code source BPEL de cet exemple car, contrairement au service de traitement de demandes de prêt (12) et celui de réservation de voyages (29) qui sont tirés de la littérature, celui-ci est le notre.

```
<?xml version="1.0" encoding="UTF-8"?>
<process
    name="SimpleAuctionService"
    targetNamespace="http://enterprise.netbeans.org/bpel/SimpleAuctionService/
        SimpleAuctionService"
    xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:sxt="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Trace"
    xmlns:sxed="http://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor"
    xmlns:tns="http://enterprise.netbeans.org/bpel/SimpleAuctionService/SimpleAuctionService"
        xmlns:ns0="http://xml.netbeans.org/schema/SimpleAuctionService" xmlns:sxed2="http
        ://www.sun.com/wsbpel/2.0/process/executable/SUNExtension/Editor2">
    <import namespace="http://j2ee.netbeans.org/wsdl/SimpleAuctionService" location="
        SimpleAuctionService.wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/SimpleSellersRepresentativeWrapper
        " location="Partners/SimpleSellersRepresentative/SimpleSellersRepresentativeWrapper.
        wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://simplesellersrepresentative.bpm.com/" location="Partners/
        SimpleSellersRepresentative/SimpleSellersRepresentative.wsdl" importType="http://
        schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://enterprise.netbeans.org/bpel/SimpleBuyersRepresentativeWrapper"
        location="Partners/SimpleBuyersRepresentative/SimpleBuyersRepresentativeWrapper.
        wsdl" importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://simplebuyersrepresentative.sample.bpm.com/" location="Partners/
        SimpleBuyersRepresentative/SimpleBuyersRepresentative.wsdl" importType="http://
        schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink name="SellersRepresentative" xmlns:tns="http://enterprise.netbeans.org/
            bpel/SimpleSellersRepresentativeWrapper" partnerLinkType="tns:
            SimpleSellersRepresentativeLinkType" partnerRole="
```

```
        bpel/SimpleSellersRepresentativeWrapper" partnerLinkType="tns:
        SimpleSellersRepresentativeLinkType" partnerRole="
        SimpleSellersRepresentativeRole"/>
    <partnerLink name="BuyersRepresentative" xmlns:tns="http://enterprise.netbeans.org/
        bpel/SimpleBuyersRepresentativeWrapper" partnerLinkType="tns:
        SimpleBuyersRepresentativeLinkType" partnerRole="SimpleBuyersRepresentativeRole
        "/>
    <partnerLink name="AuctionBranch" xmlns:tns="http://j2ee.netbeans.org/wsdl/
        SimpleAuctionService" partnerLinkType="tns:SimpleAuctionService" myRole="
        SimpleAuctionServicePortTypeRole"/>
</partnerLinks>
<variables>
    <variable name="GetBuyerOfferOut" xmlns:tns="http://simplebuyersrepresentative.sample
        .bpm.com/" messageType="tns:getBuyerOfferResponse"/>
    <variable name="GetBuyerOfferIn" xmlns:tns="http://simplebuyersrepresentative.sample.
        bpm.com/" messageType="tns:getBuyerOffer"/>
    <variable name="BiddingSellerOut" xmlns:tns="http://simplesellersrepresentative.bpm.
        com/" messageType="tns:biddingSellerResponse"/>
    <variable name="BiddingSellerIn" xmlns:tns="http://simplesellersrepresentative.bpm.
        com/" messageType="tns:biddingSeller"/>
    <variable name="SimpleAuctionServiceOperationOut" xmlns:tns="http://j2ee.netbeans.org
        /wsdl/SimpleAuctionService" messageType="tns:
        SimpleAuctionServiceOperationResponse"/>
    <variable name="SimpleAuctionServiceOperationIn" xmlns:tns="http://j2ee.netbeans.org/
        wsdl/SimpleAuctionService" messageType="tns:SimpleAuctionServiceOperationRequest
        "/>
</variables>
<sequence>
    <receive name="ReceiveFromBranch" createInstance="yes" partnerLink="AuctionBranch"
        operation="SimpleAuctionServiceOperation" xmlns:tns="http://j2ee.netbeans.org/
        wsdl/SimpleAuctionService" portType="tns:SimpleAuctionServicePortType" variable
        ="SimpleAuctionServiceOperationIn"/>
    <assign name="prepareWhile">
        <copy>
            <from>'OFFER REJECTED'</from>
            <to>$BiddingSellerOut.parameters/return</to>
        </copy>
        <copy>
            <from>$SimpleAuctionServiceOperationIn.buyersRepresentativeRequestIn/ns0:
                buyerOffer</from>
            <to>$GetBuyerOfferOut.parameters/return</to>
        </copy>
        <copy>
            <from>$SimpleAuctionServiceOperationIn.buyersRepresentativeRequestIn/ns0:
                idProd</from>
            <to>$GetBuyerOfferIn.parameters/idProd</to>
        </copy>
    </assign>
    <while name="While">
        <condition>'SALE CANCELED' != $BiddingSellerOut.parameters/return and 'OFFER
```

```
        ACCEPTED' != $BiddingSellerOut.parameters/return and $GetBuyerOfferOut.
        parameters/return != 0</condition>
    <sequence name="WhileTrue">
        <assign name="prepareInvokeSeller">
            <copy>
                <from>$SimpleAuctionServiceOperationIn.buyersRepresentativeRequestIn/
                    ns0:idProd</from>
                <to>$BiddingSellerIn.parameters/idProd</to>
            </copy>
            <copy>
                <from>$SimpleAuctionServiceOperationIn.buyersRepresentativeRequestIn/
                    ns0:buyerOffer</from>
                <to>$BiddingSellerIn.parameters/buyerOffer</to>
            </copy>
        </assign>
        <invoke name="InvokeSeller" partnerLink="SellersRepresentative" operation="
            biddingSeller" xmlns:tns="http://simplesellersrepresentative.bpm.com/"
            portType="tns:SimpleSellersRepresentative" inputVariable="
            BiddingSellerIn" outputVariable="BiddingSellerOut"/>
        <if name="If">
            <condition>'OFFER REJECTED' = $BiddingSellerOut.parameters/return</
                condition>
            <sequence name="IfTrue">
                <assign name="prepareInvokeBuyer">
                    <copy>
                        <from>$GetBuyerOfferOut.parameters/return</from>
                        <to>$GetBuyerOfferIn.parameters/buyerOffer</to>
                    </copy>
                </assign>
                <invoke name="InvokeBuyer" partnerLink="BuyersRepresentative"
                    operation="getBuyerOffer" xmlns:tns="http://
                    simplebuyersrepresentative.sample.bpm.com/" portType="tns:
                    SimpleBuyersRepresentative" inputVariable="GetBuyerOfferIn"
                    outputVariable="GetBuyerOfferOut"/>
            </sequence>
        </if>
    </sequence>
</while>
<assign name="PrepareReplyToBranch">
    <copy>
        <from>$SimpleAuctionServiceOperationIn.buyersRepresentativeRequestIn/ns0:
            idProd</from>
        <to>$SimpleAuctionServiceOperationOut.buyersRepresentativeRequestOut/ns0:
            idProd</to>
    </copy>
    <copy>
        <from>$GetBuyerOfferOut.parameters/return</from>
        <to>$SimpleAuctionServiceOperationOut.buyersRepresentativeRequestOut/ns0:
            buyerOffer</to>
    </copy>
```

```
            <copy>
                <from>$BiddingSellerOut.parameters/return</from>
                <to>$SimpleAuctionServiceOperationOut.buyersRepresentativeRequestOut/ns0:
                    sellerResponse</to>
            </copy>
        </assign>
        <reply name="ReplyToBranch" partnerLink="AuctionBranch" operation="
            SimpleAuctionServiceOperation" xmlns:tns="http://j2ee.netbeans.org/wsdl/
            SimpleAuctionService" portType="tns:SimpleAuctionServicePortType" variable="
            SimpleAuctionServiceOperationOut"/>
    </sequence>
</process>
```

Listage C.1 Service de vente aux enchères : code source.

Bibliographie

(1) Graphviz — Graph Visualization Software. `http://www.graphviz.org`. (Page consultée le 20 mars 2010).

(2) Web Services Choreography Description Language Version 1.0. `http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/`, novembre 2005.

(3) SOAP Version 1.2 W3C Recommendation (Second Edition). `http://www.w3.org/TR/soap`, mars 2007.

(4) Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`, avril 2007.

(5) Barros A., Dumas M. et Oaks P. : A Critical Overview of the Web Services Choreography Description Language (WS-CDL). `http://bptrends.com/publicationfiles/03-05%20WP%20WS-CDL%20Barros%20et%20al.pdf`, 2005.

(6) Assaf ARKIN, Sid ASKARY, Scott FORDIN, Wolfgang JEKELI, Kohsuke KAWAGUCHI, David ORCHARD, Stefano POGLIANI, Karsten RIEMER, Susan STRUBLE, Pal T. NAGY, Ivana TRICKOVIC et Sinisa ZIMEK : Web Service Choreography Interface (WSCI) 1.0. `http://www.w3.org/TR/wsci/`, 2002.

(7) Siddharth BAJAJ, Don BOX, Dave CHAPPELL, Francisco CURBERA, Glen DANIELS, Phillip H. BAKER, Maryann HONDO, Chris KALER, Dave LANGWORTHY, Anthony NADALIN, Nataraj NAGARATNAM, Hemma PRAFULLCHANDRA, Claus von RIEGEN, Daniel ROTH, Jeffrey SCHLIMMER, Chris SHARP, John SHEWCHUK, Asir VEDAMUTHU, ümit YALÇINALP et David ORCHARD : Web services policy 1.2 – framework (WS-Policy). `http://www.w3.org/Submission/WS-Policy/`, avril 2006.

(8) Fabio BARBON, Paolo TRAVERSO, Marco PISTORE et Michele TRAINOTTI : Run-time monitoring of instances and classes of web service compositions. *In ICWS '06 : Proceedings of the IEEE International Conference on Web Services*, pages 63–71, Washington, DC, USA, 2006. IEEE Computer Society.

(9) Luciano BARESI et Sam GUINEA : Towards Dynamic Monitoring of WS-BPEL Processes. *In International Conference on Service Oriented Computing*, pages 269–282, 2005.

(10) Luciano BARESI et Sam GUINEA : A dynamic and reactive approach to the supervision of BPEL processes. *In ISEC '08 : Proceedings of the 1st India software engineering conference*, pages 39–48, New York, NY, USA, 2008. ACM.

(11) Luciano BARESI, Sam GUINEA et Liliana PASQUALE : Self-healing BPEL processes with Dynamo and the JBoss rule engine. *In ESSPE '07 : International workshop on engineering of software services for pervasive environments*, pages 11–20, New York, NY, USA, 2007. ACM.

(12) Sherry BARKODAR : Creating a Loan Processing Composite Application. `http://netbeans.org/kb/61/soa/loanprocessing.html`, avril 2008.

(13) Catriel BEERI, Anat EYAL, Simon KAMENKOVICH et Tova MILO : Querying business processes with BP-QL. *In VLDB '05 : Proceedings of the 31st international conference on Very large data bases*, pages 1255–1258. VLDB Endowment, 2005.

(14) Catriel BEERI, Anat EYAL, Tova MILO et Alon PILBERG : BP-Mon : query-based monitoring of BPEL business processes. *SIGMOD Rec.*, 37(1):21–24, 2008.

(15) A. BEUGNARD, J.-M. JÉZÉQUEL, N. PLOUZEAU et D. WATKINS : Making components contract aware. *IEEE Computer*, 32(7):38–45, 1999.

(16) Mark BURSTEIN, Jerry HOBBS, Ora LASSILA, Drew MCDERMOTT, Sheila MCILRAITH, Srini NARAYANAN, Massimo PAOLUCCI, Bijan PARSIA, Terry PAYNE, Evren SIRIN, Naveen SRINIVASAN et Katia SYCARA : OWL-S : Semantic Markup for Web Services. `http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/`, novembre 2004.

(17) Meredith Greg CHRISTENSEN ERIK, Curbera Francisco et Weerawarana SANJIVA : Web Service Definition Language (WSDL). `http://www.w3.org/TR/wsdl`, mars 2001.

(18) Lori A. CLARKE et David S. ROSENBLUM : A historical perspective on runtime assertion checking in software development. *SIGSOFT Softw. Eng. Notes*, 31(3):25–37, 2006.

(19) A. DAN, A. R. FRANCK, A. KELLER, R. KING et H. LUDWIG : (IBM) Web Service Level Agreement (WSLA) Language Specification. 2002.

(20) Jos DE BRUIJN, Christoph BUSSLER, John DOMINGUE, Dieter FENSEL, Martin HEPP, Uwe KELLER, Michael KIFER, Birgitta KÖNIG-RIES, Jacek KOPECKY, Rubén LARA, Holger LAUSEN, Eyal OREN, Axel POLLERES, Dumitru ROMAN, James SCICLUNA et Michael STOLLBERG : Web Service Modeling Ontology (WSMO). W3C Member Submission, juin 2005.

(21) Remco DIJKMAN et Marlon DUMAS : Service-oriented Design : A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13:337–368, 2004.

(22) Jean-Jacques DUBRAY : A new model for ebXML BPSS multi-party collaborations and Web services choreography. `http://www.ebpml.org/ebpml.doc`, 2002.

(23) Active ENDPOINTS : BPEL Open Source. `http://www.activevos.com/community-open-source.php`. (Page consultée le 09 janvier 2009).

(24) Roy T. FIELDING : *Architectural Styles and the Design of Network-based Software Architectures.* Thèse de doctorat, University of California, Irvine, 2000.

(25) Erich GAMMA, Richard HELM, Ralph JOHNSON et John VLISSIDES : *Design patterns : elements of reusable object-oriented software.* Addison-Wesley Professional, 1995.

(26) Yuan GAN, Marsha CHECHIK, Shiva NEJATI, Jon BENNETT, Bill O'FARRELL et Julie WATERHOUSE : Runtime monitoring of Web service conversations. *In CASCON '07 : Proceedings of the 2007 conference of the center for advanced studies on collaborative research*, pages 42–57, New York, NY, USA, 2007. ACM.

(27) Inc. GARTNER : Gartner Says the Number of Organizations Planning to Adopt SOA for the First Time Is Falling Dramatically. `http://www.gartner.com/it/page.jsp?id=790717`. (Page consultée le 10 avril 2010).

(28) Object M. GROUP : CORBA Component Model 4.0 Specification. Specification Version 4.0, Object Management Group, avril 2006.

(29) Anastasia KOVAL : Understanding the Travel Reservation Service. `http://netbeans.org/kb/61/soa/understand-trs.html`, novembre 2008.

(30) Damjan KOVAČ et Denis TRČEK : Qualitative trust modeling in SOA. *J. Syst. Archit.*, 55(4):255–263, 2009.

(31) Seung-Hyun LEE et Dong-Ryeol SHIN : Web Service QoS in Multi-Domain. *10th International Conference on Advanced Communication Technology, 2008. ICACT 2008.*, 3(5):1759–1762, 2004.

(32) Wei MA, Vladimir TOSIC, Babak ESFANDIARI et Bernard PAGUREK : Extending Apache Axis for monitoring of Web service offerings. *In BSN '05 : Proceedings of the IEEE international workshop on business services networks*, pages 7–7, Piscataway, NJ, USA, 2005. IEEE Press.

(33) Sir Henry James Sumner MAINE : *Ancient Law : Its Connection With the Early History of Society, and Its Relation to Modern Ideas.* London : John Murray, 1861.

(34) E. Michael MAXIMILIEN et Munindar P. SINGH : A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Computing*, 8(5):84–93, 2004.

(35) MEIWU, Prashant BHAGAT et Andrew HOPKINSON : Monitoring BPEL Instances. `http://wiki.open-esb.java.net/Wiki.jsp?page=MonitoringBPELInstances`. (Page consultée le 25 février 2009).

(36) B. MEYER : Applying "design by contract". *IEEE Computer*, 25(10):40–51, 1992.

(37) MICROSOFT : Distributed Component Object Model (DCOM) Remote Protocol Specification. `http://msdn.microsoft.com/library/cc201989.aspx`, 2008.

(38) Red Hat MIDDLEWARE : JBoss.com — JBoss Rules. `http://www.jboss.com/products/rules`. (Page consultée le 08 janvier 2009).

(39) R. MILNER : *Communication and concurrency.* Prentice-Hall, 1989.

(40) Anders MØLLER : dk.brics.automaton. `http://www.brics.dk/automaton/index.html`. (Page consultée le 25 juillet 2009).

(41) OASIS : Collaboration-Protocol Profile and Agreement Specification Version 2.0. Specification Version 2.0, septembre 2002.

(42) Kruti PATEL, Bernard PAGUREK et Vladimir TOSIC : Improvements in WSOL Grammar and "Premier" WSOL Parser. `http://www.sce.carleton.ca/netmanage/papers/Improvements%20in%20WSOL%20Grammar%20and%20Premier%20WSOL%20Parser.pdf`, 2003.

(43) C. PELTZ : Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, octobre 2003.

(44) Marco PISTORE et Michele TRAINOTTI : ASTRO : Supporting Composition and Execution of Web Services. *In Proceedings of the Third International Conference on Service-Oriented Computing (LNCS 3826*, page 501. Springer, 2005.

(45) Marco PISTORE et Paolo TRAVERSO : ASTRO ~ Supporting the Composition of Distributed Business Processes ~ Research. `http://www.astroproject.org/research.php`. (Page consultée le 15 janvier 2009).

(46) Frantisek PLASIL et Stanislav VISNOVSKY : Behavior protocols for software components. *IEEE Trans. Softw. Eng.*, 28(11):1056–1076, 2002.

(47) Will PROVOST : UML for Web Services. `http://www.capcourse.com/Library/UMLforWebServices/Article.html`, août 2003. (Page consultée le 1 mai 2010).

(48) Office québécois de la langue FRANÇAISE : Le grand dictionnaire terminologique. `http://www.granddictionnaire.com/BTML/FRA/r_Motclef/index1024_1.asp`. (Page consultée le 10 avril 2010).

(49) Xuan SHI : Sharing Service Semantics using SOAP-Based and REST Web Services. *IT Professional*, 8(2):18–24, 2006.

(50) Inc. SUN MICROSYSTEMS : BPEL Monitor. `http://wiki.open-esb.java.net/Wiki.jsp?page=BPELMonitor`. (Page consultée le 10 févier 2009).

(51) Inc. SUN MICROSYSTEMS : JSR 208 : JavaTM Business Integration (JBI). `http://www.jcp.org/en/jsr/detail?id=208`. (Page consultée le 14 juin 2009).

(52) Inc. SUN MICROSYSTEMS : Open ESB. `https://open-esb.dev.java.net/`. (Page consultée le 10 févier 2009).

(53) Inc. SUN MICROSYSTEMS : JDK 5.0 Remote Method Invocation (RMI). `http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html`, 2004.

(54) Satish THATTE : Usage of BPEL Abstract Processes. http://www.oasis-open.org/committees/download.php/6876/UsageofBPELAbstractProcessesStrawman.doc, mai 2004.

(55) Vladimir TOSIC et Bernard PAGUREK : On Comprehensive Contractual Descriptions of Web Services. *In EEE '05 : Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 444–449, Washington, DC, USA, 2005. IEEE Computer Society.

(56) Vladimir TOSIC, Bernard PAGUREK, Kruti PATEL, Babak ESFANDIARI et Wei MA : Management applications of the Web service offerings language (WSOL). *Inf. Syst.*, 30(7):564–586, 2005.

(57) Vladimir TOSIC, Vladimir TOSIC, Bernard PAGUREK, Bernard PAGUREK, Kruti PATEL et Kruti PATEL : WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Rapport technique, 2002.

(58) G. TREMBLAY et J. CHAE : Towards specifying contracts and protocols for Web services. *In* H. MILI et F. KHENDEK, éditeurs : *MCeTech Montreal Conference on eTechnologies*, pages 73–85, Montréal, Canada, janvier 2005.

(59) Yuli VASILIEV : *SOA and WS-BPEL*. Packt Publishing, août 2007.

(60) W3C : World Wide Web Consortium - Web Standards. http://www.w3.org/. (Page consultée le 10 janvier 2009).

(61) World Wide Web Consortium. *Web Services Glossary*, février 2004.

(62) Service Level Agreement ZONE : The Service Level Agreement. http://www.sla-zone.co.uk/. (Page consultée le 10 janvier 2009).