

LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

PLATEFORME ADORE - ASPECTS ET DISTRIBUTED ORCHESTRATIONS

Cédric Joffroy, Sébastien Mosser, Mireille Blay-Fornarino

Projet RAINBOW

Rapport de recherche
ISRN I3S/RR-2007-09-FR

Mars 2007

PLATEFORME ADORE



Aspects & Distributed ORchEstrations

Cédric Joffroy *Sébastien Mosser* **Mireille Blay-Fornarino**

{joffroy,mosser,blay}@polytech.unice.fr

EPU POLYTECH'NICE SOPHIA-ANTIPOLIS
LABORATOIRE I3S, ÉQUIPE RAINBOW
930 Route des Colles – B.P. 145
F-06903 Sophia Antipolis Cedex

PROJET DE FIN D'ÉTUDES (INGÉNIEUR SI)

ANNÉE UNIVERSITAIRE 2006–2007



Résumé

Les Systèmes d'Informations (SI) doivent continuellement être adaptés pour tenir compte de l'évolution des besoins utilisateurs, des modifications de législation et également des changements de supports technologiques. Les architectes logiciels doivent donc mettre au point des systèmes complexes, mêlant les spécificités du code métier et les mécanismes d'évolutions du système.

Les architectures orientées services [MacKenzie et al., 2006] sont particulièrement bien adaptées au développement de logiciels complexes et au support des évolutions rapides des applications en fonction des services attendus et disponibles. Un des paramètres essentiels de cette adaptabilité est le faible couplage entre les services. Pour atteindre cet objectif, les compositions entre les services sont explicitées en utilisant des langages adaptés qui offrent différents mécanismes pour composer et contrôler les invocations entre services.

Dans le cadre des Services Web, nous nous intéressons aux langages d'orchestrations. Les orchestrations sont définies par le consortium W3C (W3C Glossary) comme *"le modèle des interactions que doit respecter un agent Service Web pour atteindre son but"*.

Mais, lorsqu'un système d'information utilisant une architecture à base de services doit évoluer de manière non anticipée sans interruption de l'application. A notre connaissance, il n'existe actuellement aucune plateforme capable de supporter une telle évolution. Lorsque ces évolutions interviennent suite à des préoccupations différentes, il est nécessaire de gérer la cohérence de ces adaptations.

Notre contribution se propose de combler ces vides, en considérant les orchestrations comme des aspects explicitant les interactions sur une application constituée des services de base. Ces aspects permettent une expression séparée des différents contrôles, tandis que nous prenons en charge la détection des conflits et le calcul de l'adaptation résultante.

Ce rapport se décompose en trois parties. La première partie explicite la problématique et la solution générale que nous proposons. La deuxième partie décrit plus précisément la démarche et les différents éléments de mise en oeuvre de cette solution. La dernière partie décrit brièvement l'application qui tient lieu pour l'instant de validation de ce travail et exhibe quelques unes des perspectives envisagées de ce travail.

Mots Clés : *Orchestrations, Aspects, Web Services, Composition, Interactions, SOA*

Sommaire

I	Contexte	2
1	Introduction	3
2	Contrôler un Service Web	5
3	Une plateforme multi-tâches	13
II	Un développement guidé par les modèles	22
4	Une plateforme multi-modèles	23
5	ODML : Des aspects pour contrôler	26
6	OMSM : De la logique pour fusionner	31
7	GOD : Des objets pour afficher	37
8	DEMON : Des <i>threads</i> pour exécuter	40
III	Conclusion	43
9	Démonstration : « <i>Proof Of Concept</i> »	44
10	Conclusion & Évolutions futures	50
IV	Annexes	52
A	Une orchestration BPEL	53
B	ODML : Syntaxe du langage	56
C	Orchestration GOD : Schéma XSD	61
D	Configuration des services ADORE	63
E	Lancement de la plateforme	66

Première partie

Contexte

Chapitre 1

Introduction

Le projet ADORE est né au sein de l'équipe RAINBOW du laboratoire I3S, comme une suite logique des travaux de David EMSELLEM, ex-ingénieur (2002–2006) de recherche de l'équipe, et fondateur de VULOG.

Il s'inscrit dans le thème «*Modélisation, Composition et Transformation de Modèles*». L'équipe s'efforce dans cette thématique de mettre en place des mécanismes permettant la fusion comportementale de composants métiers. Après avoir appliqué ces mécanismes aux composants JAVA [Blay-Fornarino et al., 2004, Berger, 2001], elle s'efforce aujourd'hui de mettre en place des mécanismes similaires dans le monde des Services Web. Ce projet exploratoire rentre dans le cadre du RNTL FAROS, et vise à mettre en place une plateforme permettant la composition dynamique de Services Web.

1.1 Un monde de Service ...

Les architectures orientées services (SOA) sont utilisées pour créer des systèmes réactifs, grâce au faible couplage présent entre les différents services mis en jeu au sein des systèmes.

Les composants techniques métiers sont exposés sous la forme de services (dans notre cas, des Services Web). Leurs interfaces sont clairement définies au sein d'un contrat, utilisé par des générateurs de code pour créer des *stubs* à destination des services exposés.

En l'état, ces services découplés les uns des autres n'ont que peu d'intérêt. La création de processus métiers répondant aux besoins des utilisateurs passe par la définition d'Orchestrations (*ie* assemblages) de services. Ces orchestrations, définies de manière séparée du code source des services (*séparation des préoccupations*), permettent d'ordonner les invocations des services métiers, de composer différents services en un seul, ...

1.2 Un monde de «normes» ...

Le monde des Services Web est composé d'un ensemble de normes et autres protocoles standards. On peut citer parmi les principaux protocoles :

- Transport des données : HTTP
- Encodage des données : SOAP, XML-RPC (dialectes XML)
- Interface des services : WSDL (dialecte XML)
- Description des structures de données : XSD (dialecte XML)
- Définition de processus métiers par orchestration : BPEL (dialecte XML)

A ces protocoles principaux se greffent un grand nombre de protocoles divers et variés : UDDI, WS-SECURITY, WS-POLICY, SAML, XACML, ... On compte parmi les rangs des organismes de normalisation des protocoles précédemment cités le W3C, l'OASIS, l'OSI, ...

1.3 Deux approches complémentaires

Les orchestrations se présentent comme un ensemble de mécanismes pour la construction d'un nouveau service dit composite dans une application composée de l'ensemble des services atteignables. Au niveau des orchestrations sont définies des variables qui permettent de partager les informations entre les différentes invocations. Sans les orchestrations, la prise en charge de ces nouvelles applications imposerait de définir un nouveau service faisant référence aux précédents de manière cachée, voir dans certains cas de redéfinir les services eux-même pour exprimer les interactions avec les services requis. Les nouvelles notations [White, 2006a, White, 2006b], associées aux orchestrations visent à favoriser le maintien d'un faible couplage, une lecture plus facile des codes et la réutilisation des services composites ou non [Bartoli et al., 2005].

Cependant, même si les orchestrations sont un support à une programmation incrémentale pour réagir à l'introduction d'une nouvelle levée d'événement ou la coordination avec un nouveau service (approche composite, [Chandran and Poduval, 2005, Bartoli et al., 2005]), elles n'offrent pas de support à leur propre composition. Ce point est d'autant plus critique que les applications supportées évoluent très rapidement et que le développement collaboratif est reconnu comme indispensable à l'élaboration de gros projets.

Dans le cadre de son MASTER RECHERCHE, Clémentine NEMO positionne ses travaux sur la composition d'orchestrations de Services Web [Nemo, 2006, Nemo et al., 2006] comme la recherche d'appels multiples à des services communs. La détection de tels "points de fusion" donne alors lieu à un processus de fusion où le développeur est fortement impliqué.

A l'inverse des travaux de Clémentine NEMO, nous nous intéressons aux orchestrations comme un moyen d'exprimer de nouveaux comportements, et ainsi permettre l'utilisation de ce formalisme à des fins de modifications du comportement des services.

Il faut cependant remarquer que les deux travaux ne sont pas antinomiques, et partagent une base commune de développement.

1.4 Positionnement d'ADORE

La plateforme ADORE mise en place dans le cadre du projet RAINBOW vise à permettre une évolution rapide des applications à base de services.

Ce projet explore différents paradigmes de programmation (logique, objet, aspects, ...) afin d'en extraire leurs avantages respectifs et produire une plateforme permettant la modification comportementale d'opérations de Services Web.

Chapitre 2

Contrôler un Service Web

Dans un premier temps, nous précisons au travers d'exemples les besoins en matière d'introduction dynamique de contrôle dans les assemblages de Services Web. Nous présentons ensuite le langage sur lequel nous avons travaillé pour répondre à ce besoin.

2.1 Exemple : Architecture Bancaire

Dans cette partie, nous nous positionnons au sein d'un système d'information bancaire. Nous disposons, à l'intérieur du système, des cinq Services Web suivants :

- Services Métiers :
 - AccountManager : permet la gestion des comptes bancaires des clients de l'établissement ;
 - BankingFraud : fournit une liste de comptes soupçonnés de fraude par la justice française ;
 - AccountChecker : fournit une liste de comptes certifiés pour les transactions électroniques.
- Services Techniques :
 - TcpLog : permet d'envoyer une information sur une socket TCP à des fins de *debug* ;
 - Historic : permet la journalisation des opérations effectuées au sein du système d'information.

Afin d'améliorer la compréhension de ces exemples, nous décrivons les différentes orchestrations mises en jeu, dans les exemples suivants, en utilisant le formalisme des Diagrammes d'Activité UML.

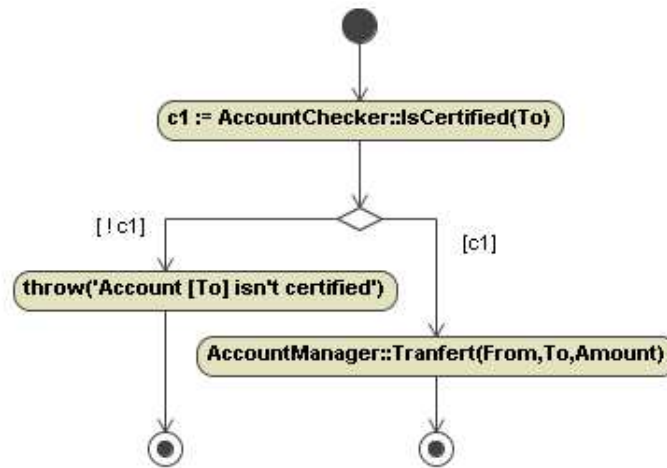
2.1.1 Composition de processus métiers

Certification des comptes bancaires obligatoire

Hypothèse : Selon la Loi, toute opération bancaire effectuée de manière électronique ne peut s'effectuer que si le compte destinataire est certifié pour ce type d'opérations.

Afin de répondre à cette nouvelle préoccupation, un programmeur P_1 va utiliser une orchestration afin de combiner les services AccountManager et AccountChecker. Il écrit donc l'orchestration O_1 , décrite en Figure 2.1.

L'orchestration ainsi définie est ensuite chargée dans un moteur d'exécution BPEL, et est publiée comme un nouveau service web AccountManagerWithCertification. Il faut mainte-

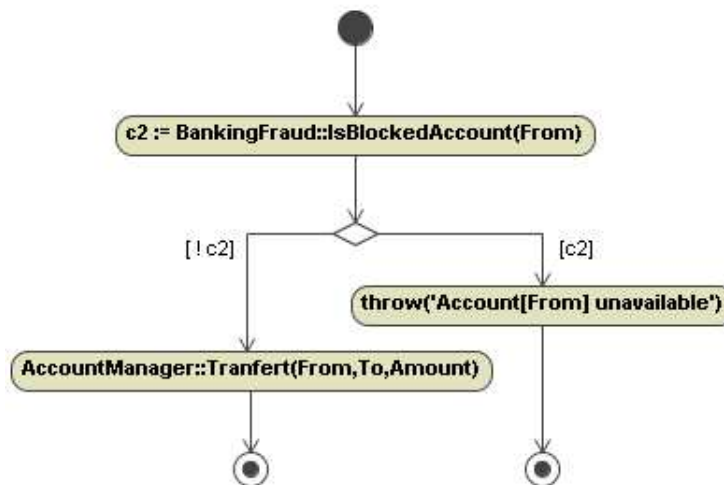
FIG. 2.1 – O_1 : Certification électronique

nant informer tous les services développant des logiciels invoquant l'ancien service pour qu'ils adressent maintenant le nouveau service publié.

Gestion des fraudes bancaires

Hypothèse : Selon la Loi, aucune opération bancaire ne peut être effectuée si le compte source est soupçonné de fraude bancaire.

Un second programmeur P_2 va décrire une orchestration combinant les services `AccountManager` et `BankingFraud`. L'orchestration O_2 ainsi produite (Figure 2.2) est publiée dans le moteur d'exécution BPEL sous le nom `AccountManagerWithFraud`, et les consommateurs du service `AccountManager` doivent être informés de la publication de ce nouveau service.

FIG. 2.2 – O_2 : Fraude Bancaire

Le problème

Dans le scénario décrit précédemment, les développeurs P_1 et P_2 ont travaillé de manière non coordonnée. Le système d'information se retrouve donc constitué de trois services pour gérer le même but, chacun doté de sa sémantique :

- AccountManager : Service historiquement présent dans le SI ;
- AccountManagerWithCertification : Gestion de la certification électronique ;
- AccountManagerWithFraud : Gestion des fraudes bancaires.

Afin d'uniformiser le système d'information, le chef de produit P_L , qui détient la gouvernance sur le système d'information, va fusionner les deux orchestrations décrites par ses développeurs, et produire l'orchestration O_3 dessinée en Figure 2.3.

Il doit détecter l'existence d'un conflit lorsque les conditions $!c_1 \& \& c_2$ sont réunies, et choisir quelle exception lever dans un tel cas. Étant données les circonstances, il paraît plus important à P_L de lever l'exception *Fraude Bancaire* dans ces conditions. La nouvelle orchestration est publiée dans le moteur d'exécution BPEL, sous le nom AccountManagerLawCompliant.

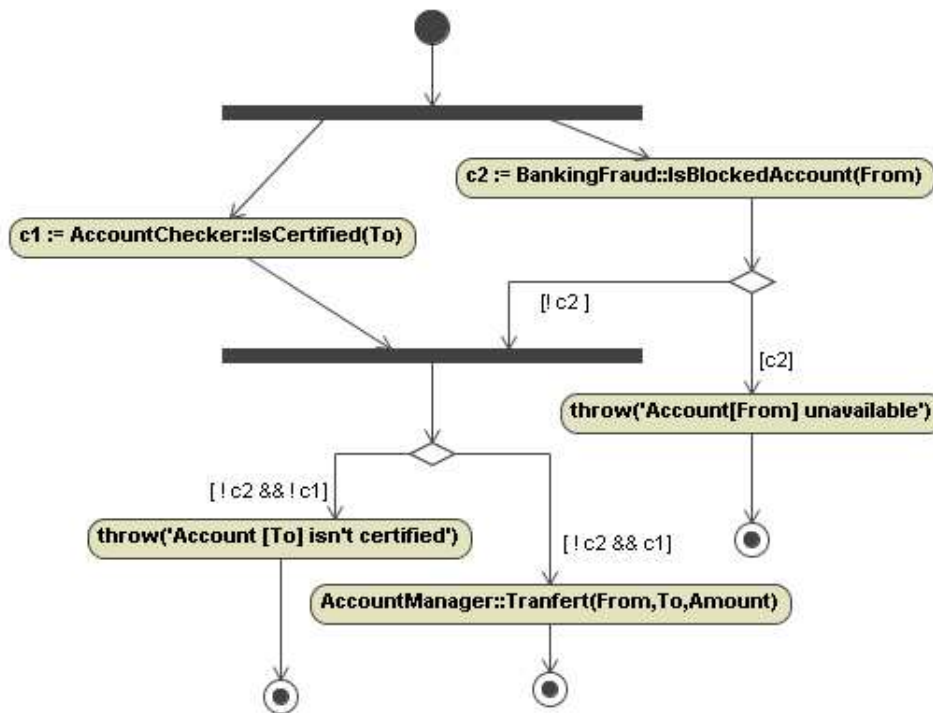


FIG. 2.3 – $O_3 \equiv \text{merge}(O_1, O_2)$

2.1.2 Préoccupations orthogonales : Journalisation & Debug

Journalisation des opérations effectuées

Hypothèse : Afin d'obtenir des statistiques sur l'utilisation du système d'information, P_L demande à son équipe de développement de journaliser durant 30 jours toutes les opérations effectuées sur le système.

Le développeur P_1 crée une nouvelle orchestration couplant `AccountManagerLawCompliant` et `Historic`, comme décrit dans la figure 2.4. Nous nous focalisons sur cette seule orchestration pour l'instant, mais le développeur fait de même pour tous les services critiques mis en jeux dans le système d'information.

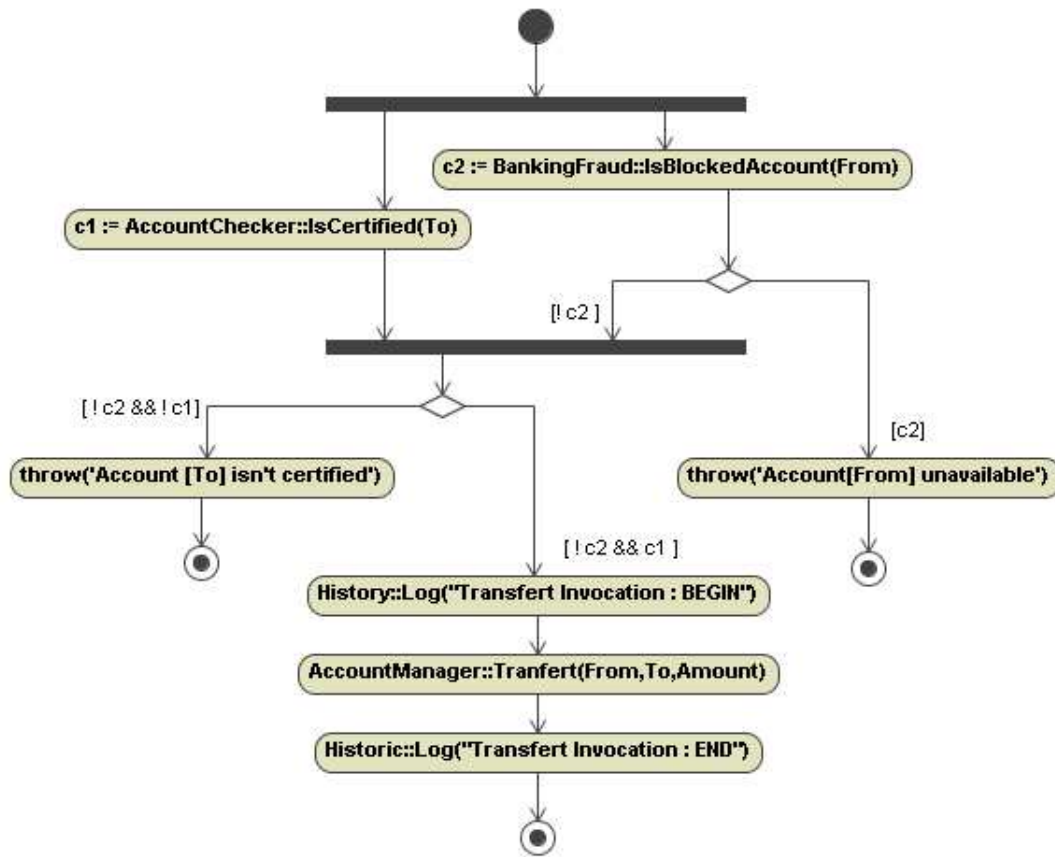


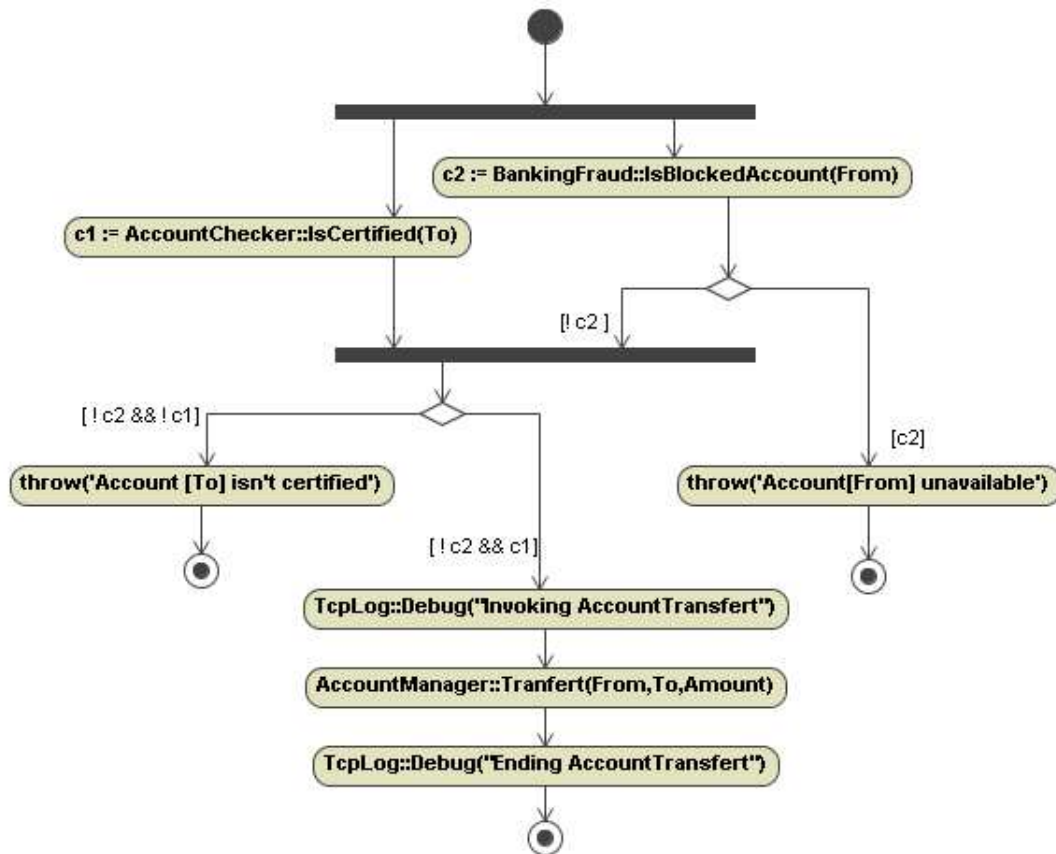
FIG. 2.4 – O_4 : Journalisation des Opérations

Cette orchestration est publiée en lieu et place de l'ancienne, ce qui implique un arrêt du moteur d'exécution BPEL le temps d'effectuer le remplacement. Une fois la durée de 30 jours dépassée, un nouvel arrêt du système sera nécessaire pour remplacer l'orchestration O_4 par O_3 .

Debugging du système d'information

Hypothèse : Dans le cadre du développement d'un nouveau service à publier sur le SI, un développeur a besoin de vérifier que les invocations à l'opération de transfert de fond suivent une politique définie par les nouvelles règles internes de l'établissement (validation de flots de messages).

Le développeur P_2 décrit donc une orchestration O_5 permettant de coupler temporairement le service `AccountManagerLawCompliant` et `TcpLog`, de manière similaire à la journalisation de la section précédente. Une fois le *debugging* effectué, un nouvel arrêt du moteur d'exécution sera nécessaire pour remettre le système dans l'état précédent.

FIG. 2.5 – O_5 :Debug du transfert de fond

2.1.3 The all together ...

Comme nous venons de le voir, les combinaisons d'orchestrations sont pour l'instant faites de manière manuelle par un *gouverneur*. La production d' O_3 reste un exercice complexe si l'on se place dans un contexte industriel. Même si les nouvelles notations telles que BPMN [White, 2006a, White, 2006b] permettent une modélisation graphique des processus métiers décrits en BPEL, un processus métier complexe reste complexe à décrire.

Si on pose l'hypothèse qu'à un instant t donné, les orchestrations O_1 , O_2 , O_4 et O_5 doivent s'appliquer, on obtient comme résultat l'orchestration décrite en figure 2.6.

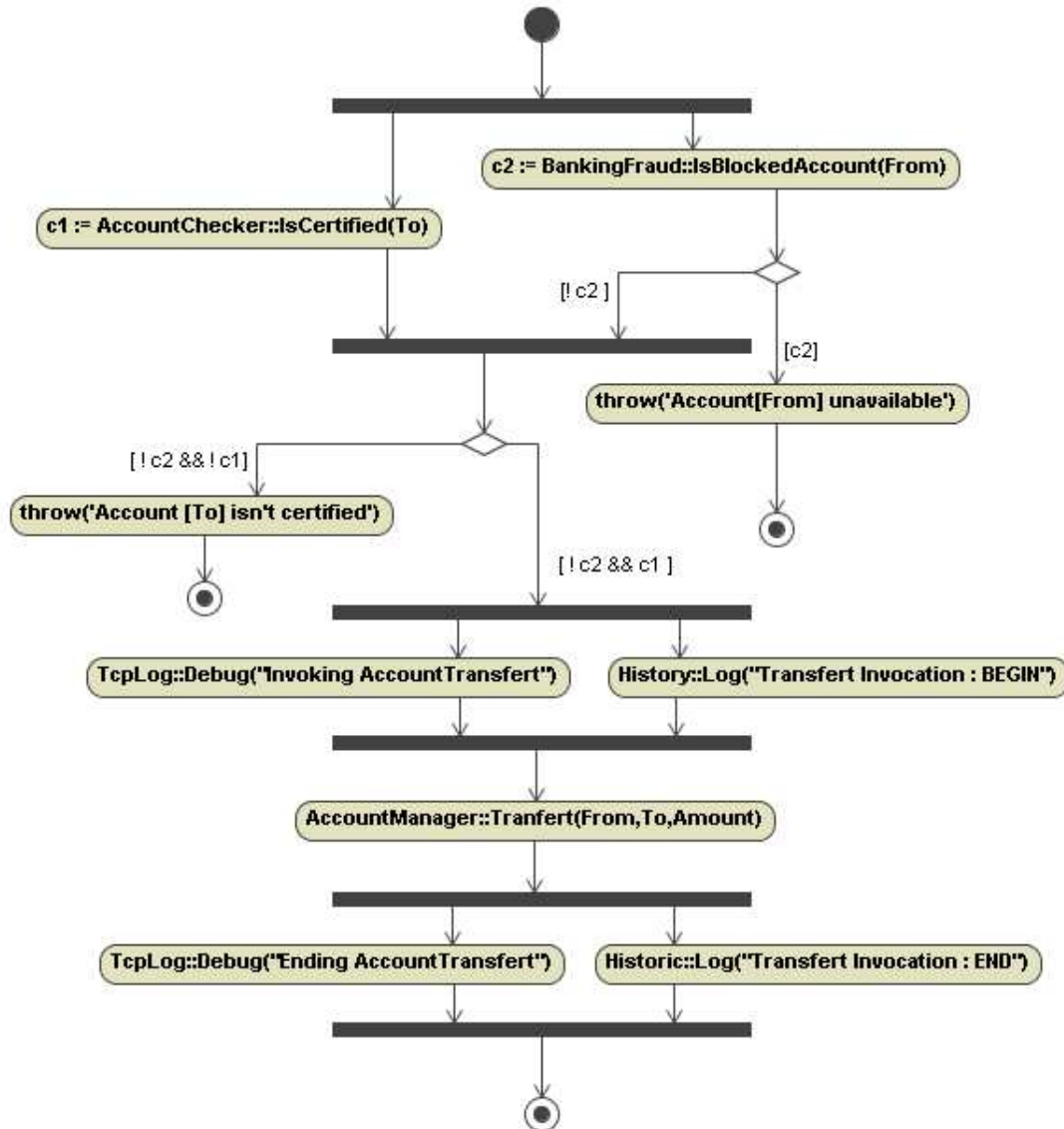


FIG. 2.6 – $O_{all} \equiv merge(O_1, O_2, O_4, O_5)$

2.2 Du contrôle sur les opérations de Services Web

2.2.1 Analyse des exemples précédents

Au vu des exemples évoqués précédemment, on se rend compte que le formalisme des orchestrations, pris dans son utilisation standard, répond mal aux problèmes de l'évolution dynamique des assemblages. En effet :

- la composition des assemblages est une opération humaine complexe ;
- les compositions obtenues à partir de processus métiers simples sont rapidement complexes ;
- le remplacement d'un processus métier implique un arrêt du système.

2.2.2 Des Orchestrations de contrôle

Les exemples décrits précédemment se positionnent en fait comme un sous-ensemble $Orch_{ctrl}$ de l'ensemble $Orch$ des orchestrations ($Orch_{ctrl} \subset Orch$) qui définissent le système d'information. Nous appelons *Orchestrations de contrôles* ce type d'orchestrations.

Soit $o \in Orch$. Une telle orchestration vise à combiner différents Services Web *basiques* (BS) en un Service *Composite* (CS), en décrivant un processus métier. Une telle orchestration est naturellement publiée comme un nouveau service dans le moteur d'exécution, et les programmes nécessitant son invocation devront être adaptés pour tenir compte de cette publication

Un des exemples classique dans la littérature est l'*Agence de Voyage Virtuelle*. A partir d'un service de réservation d'hôtels et d'un service de réservation d'avions, on cherche à mettre en place par une orchestration un service de réservation de *pack de vacances*, incluant au sein d'un seul processus métier la *Réservation d'un avion* et la *Réservation d'un hotel* sur le lieu d'arrivée.

Soit $o' \in Orch_{ctrl}$. L'orchestration o' vise, comme dans les exemples précédents, à contrôler l'invocation d'une opération d'un service, selon certaines conditions. Elle ne se prête donc pas au jeu de la publication d'un nouveau service, puisque son but est de redéfinir le comportement d'un service existant.

2.3 Langage de description des Orchestrations

2.3.1 BPEL pour orchestrer

Dans le formalisme BPEL de description des orchestrations, on ne peut *que* définir un nouveau service par assemblage de services existants.

Le chapitre A décrit une orchestration BPEL visant à publier sur un système bancaire un processus métier permettant l'approbation d'un prêt contracté auprès de l'établissement bancaire.

2.3.2 ODML pour contrôler

Dans notre cas de figure, nous nous intéressons uniquement à des orchestrations redéfinissant un comportement. Nous décidons donc d'étendre le langage BPEL pour lui intégrer la notion d'activités faisant référence au *comportement original* et l'expression des sélecteurs (point d'intercession). Ce langage se nomme ODML (*Orchestration Description Markup Language*).

Cette nouvelle activité est appelée *proceed* et est conforme au *proceed* des langages d'aspects tels qu'ASPECTJ. De plus nous introduisons l'activité *delegate*, qui permet de déléguer le comportement initial à un autre service, à la manière de [Klein et al., 2007] (surcharge de la méthode *proceed*).

Des spécifications BPEL, nous retenons dans ODML les constructions suivantes :

- L'affectation d'une valeur à une variable (`assign`)
- L'invocation d'un Service Web (`invoke`)
- Le retour de résultat (`reply`)
- La levée d'exception (`throw`)
- L'ordonnancement séquentiel des activités (`sequence`)
- L'ordonnancement parallèle des activités (`flow`)

Déviations : La spécification BPEL définit de nombreux autres mécanismes qui complexifie le langage. Volontairement, nous ne les retenons pas, afin de produire un micro-langage simple. Il est toutefois possible de passer d'ODML à BPEL par une transformation de modèle, comme expliqué au chapitre 4, page 23.

Conditionnelle : La spécification BPEL décrit une activité conditionnelle `switch`. Afin de simplifier l'implémentation de l'évaluateur d'Orchestration, nous avons introduit un classique `if/then/else`. Il est cependant aisé de projeter un `switch` en `if/then/else`, et inversement.

Syntaxe du langage : Un descriptif plus complet du langage ODML est disponible en annexe B.

Chapitre 3

Une plateforme multi-tâches

3.1 Objectifs de la plateforme

La plateforme ADORE permet la définition, la manipulation ainsi que l'évaluation d'Orchestrations de contrôle. Nous avons plus particulièrement approfondi les idées suivantes :

- La décentralisation des orchestrations
- La fusion automatique d'orchestrations de contrôle
- La modification dynamique des contrôles

3.1.1 Distribuer des orchestrations ?

Dans la définition actuelle des orchestrations, celles-ci sont chargées dans un moteur d'exécution BPEL centralisé, qui a la charge de les exposer aux consommateurs. Cette approche n'est à notre sens pas adéquate pour les orchestrations de contrôles, puisque celles-ci rédéfinissent un comportement de service. Il est donc plus logique de les charger sur le service devant être modifié.

3.1.2 Fusionner des orchestrations ?

Comme vu au chapitre précédent, la composition d'orchestration est une opération longue, fastidieuse et effectuée humainement. Ce processus est source d'erreur pour le *gouverneur*. Or, l'utilisation de mécanismes issus de la programmation orientée aspects permet (grâce à la définition de pivots de compositions au sein du langage de description des orchestrations) d'automatiser ce processus.

3.1.3 Modifier dynamiquement des orchestrations ?

En utilisant des moteurs BPEL usuels, la modification d'une orchestration implique un arrêt du système. Dans le cadre d'un travail collaboratif porté par les mécanismes de fusion automatique des orchestrations que nous mettons en place au sein de la plateforme, il est utile de pouvoir modifier un comportement sans rupture de service pour les utilisateurs.

3.2 Choix technologiques & Utilisateurs

3.2.1 Technologies

Notre implémentation utilise le framework .NET, dans sa version libre MONO, afin de s'affranchir de la contrainte mono-plateforme imposée par le framework officiel publié par MICROSOFT. Le langage choisi est le langage C#.

Notre choix s'est basé sur les critères suivants :

- Intégration avancée des Services Web au sein du framework.
- Simplicité de traitement des documents XML.
- Présence d'un existant .NET au sein de l'équipe.
- Grande facilité d'intégration d'autres langages.

La gestion de la persistance des données utilise une base de données objet DB4O. Ce système de gestion de base de données à l'avantage d'être embarqué au sein d'une DLL C#, et de ne nécessiter aucune installation annexe sur les machines cibles.

Communautés : Le framework libre MONO, tout comme la base de données objet DB4O sont portés par une grande communauté utilisateurs, dans la philosophie du logiciel libre.

3.2.2 Rôles utilisateurs

Au sein de la plateforme, nous distinguons 3 types d'acteurs différents.

Consommateurs : il s'agit des clients des Services Web contrôlés par la plateforme. Il peut s'agir d'applications finales à destination d'utilisateurs réels (*End User Software*), ou d'applications *business to business*.

Compositeurs : Les compositeurs utilisent la plateforme pour orchestrer des services contrôlés par ADORE. Ils utilisent le langage ODML pour définir de nouveaux comportements, et les charger dans la plateforme.

Administrateurs : Les administrateurs ont un contrôle complet de la plateforme. Ils peuvent autoriser des compositeurs sur la plateforme, introduire de nouveaux services contrôlés, ... L'administrateur peut aussi jouer le rôle de compositeur (*Administrateur* \subset *Compositeur*)

3.3 Une architecture à proxies

La figure 3.1 donne un aperçu général de la plateforme. Pour chaque service contrôlé, nous définissons deux Services Web permettant leur intégration au sein de la plateforme :

- ADORE Proxy : permet l'administration des contrôles sur le service
- User Proxy : permet l'invocation des contrôles par les *consommateurs*.

Chaque tuple (*UserProxy*, *AdoreProxy*) partage une base de données qui stocke les différents contrôles.

3.3.1 Proxies ADORE

Ces proxies permettent aux compositeurs de positionner des contrôles sur le service. Les orchestrations de contrôle définies sont chargées dans la base de données partagée.

Lorsque plusieurs contrôles portent sur la même opération du service contrôlé, le processus de fusion est déclenché. Le système de chargement des orchestrations de contrôle au sein des proxies ADORE est explicité en section 3.5

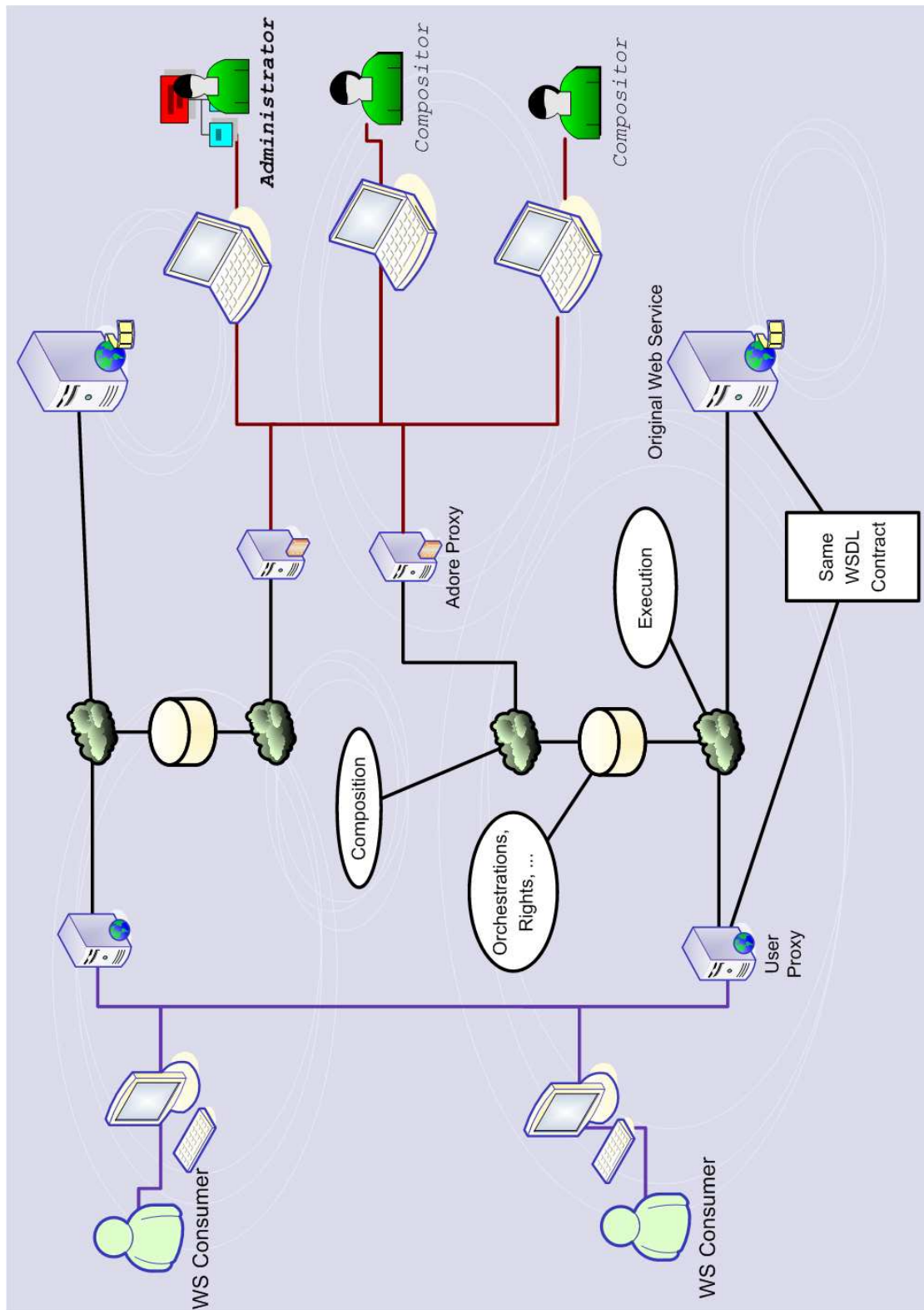


FIG. 3.1 – Architecture Générale de la plateforme ADORE

3.3.2 Proxies Utilisateurs

Afin d'autoriser le contrôle de services pré-existants, et pour faciliter le développement de nouveaux services visant à être contrôlés, nous définissons un proxy utilisateur présent en façade des services contrôlés.

Ce proxy publie le même contrat WSDL que le service contrôlé. Du point de vue des *Consommateurs*, il s'agit donc du même service, qu'ils référencent en lieu et place du service originel.

Au sein du proxy se trouve un évaluateur ODML. Lors de l'invocation d'une opération, on utilise l'algorithme suivant pour déterminer le comportement de l'opération :

début

si Un contrôle est défini **alors**
 l'évaluer

sinon
 invoquer l'opération sur le service contrôlé

finsi

fin

Les *consommateurs* invoquent uniquement les proxies et non les services contrôlés. Les modifications faites au sein des proxies ADORE sont directement prises en compte lors des invocations.

3.3.3 Ajout d'un service sur la plateforme

Pour faire en sorte qu'un service soit contrôlé par la plateforme ADORE, il faut générer :

1. Un fichier de stockage DB4O
2. Un proxy ADORE pour l'administration
3. Un proxy utilisateur pour l'évaluation des contrôles

Ces opérations sont totalement automatisable, et peuvent être faites par un générateur de code.

3.4 Services techniques

La plateforme ADORE est composée de trois services techniques uniques permettant son administration (*Annuaire*, *Gestion Utilisateurs* et *Mise à jour*). La figure 3.2 montre une architecture plus détaillée de la plateforme les mettant en évidence.

Remarque : Les proxies ADORE sont eux aussi considérés comme des services techniques d'administration, mais, contrairement aux trois services précédents, ne sont pas uniques sur le réseau.

3.4.1 Proxies Annuaire

Ce service sert d'annuaire pour la plateforme. Il référence tous les proxies déclarés par les administrateurs de la plateforme. Par convention, nous postfixons d'une (*) les opérations uniquement accessibles pour les administrateurs.

Opérations fournies :

- Login : identifie un utilisateur sur le service
- GetAll : retourne tous les proxies connus par la plateforme
- Register (*) : ajoute un couple de proxy dans l'annuaire
- UnRegister (*) : opération réciproque

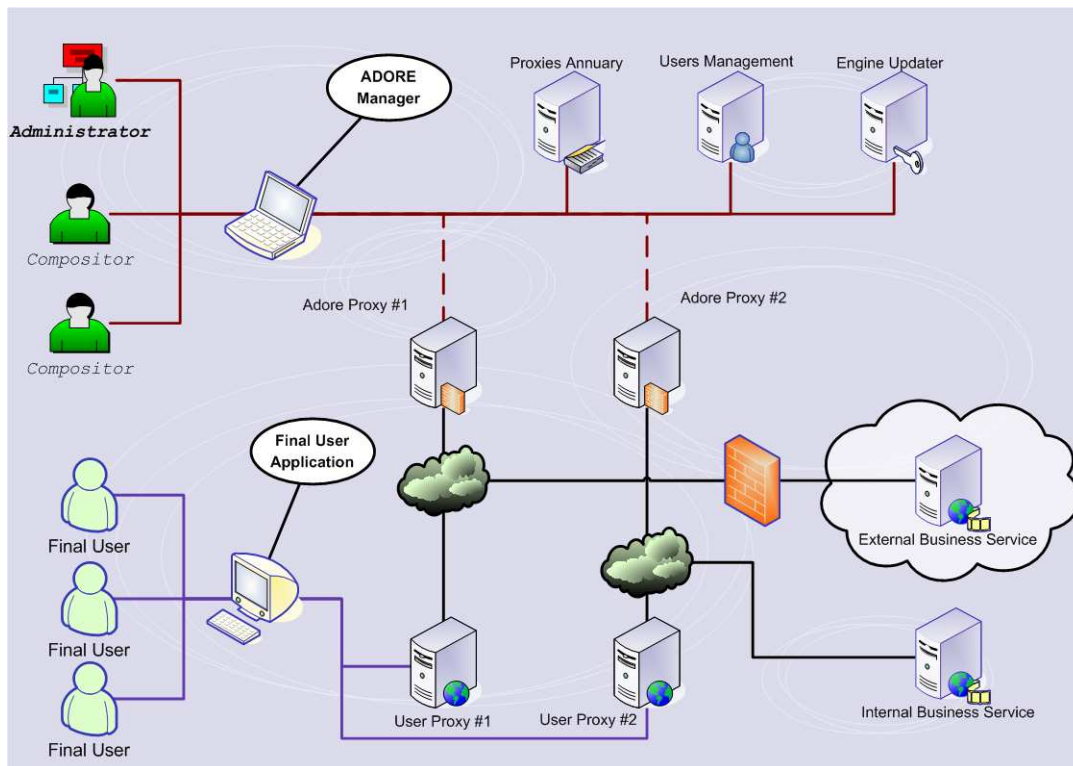


FIG. 3.2 – Services techniques de la plateforme ADORE

3.4.2 Users Management

Ce service permet la gestion des utilisateurs de la plateforme. Il permet de définir les comptes des différents administrateurs et/ou compositeurs sur la plateforme. Dans une démarche SOA, ces informations ne sont pas centralisées au sein du service, mais distribuées auprès des services en ayant besoin.

Opérations fournies :

- Identify : identifie un utilisateur sur le service
- Passwd : change le mot de passe de l'utilisateur courant
- GetAllUsers : retourne le nom et le statut de tous les utilisateurs de la plateforme
- Add (★) : Ajoute un utilisateur sur la plateforme
- Del (★) : Opération réciproque

3.4.3 Engine Updater

Ce service permet la mise à jour aisée des proxies publiés sur la plateforme. En effet, chaque proxy embarque son propre moteur de fusion, ainsi que son propre moteur d'exécution. En cas de modification, il est utile de pouvoir modifier d'un seul coup tous les moteurs. Ce service a donc la charge de mettre à jour tous les proxies référencés dans l'annuaire lorsqu'on lui demande de publier une mise à jour.

Opération fournies :

- Login : identifie un utilisateur sur le service
- Publish (★) : publie un moteur sur le service

- Propagate (★) : propage un moteur publié
- GetAll (★) : liste toutes les mises à jour effectuées.

3.4.4 ADORE proxies

Ces services (au nombre de n) permettent l'administration des contrôles définis sur les Services Web. Les opérations fournies se divisent en quatre catégories :

Opérations administratives :

- Login : identifie un utilisateur sur le service
- GetOperations : renvoie la liste des opérations publiées par le service contrôlé.

Opérations sur les utilisateurs :

- Grant (★) : autorise un utilisateur à contrôler une opération du service
- Deny (★) : opération réciproque
- GetUsers : renvoie le nom des utilisateurs pouvant contrôler une certaine opération du service

Opérations de gestion des orchestrations :

- LoadOrchestration : permet à un utilisateur autorisé de charger une orchestration de contrôle sur une opération
- UnLoadOrchestration : opération réciproque
- GetOrchestrations : retourne tous les documents ODML chargés pour une opération donnée.

Opération de gestion des connaissances : Le processus de fusion nécessite l'utilisation d'une base de connaissances pour résoudre les conflits de fusion, comme décrit en section 3.5. Cette base de connaissances est maintenue par les utilisateurs à l'aide d'opérations similaires à celles de gestion des opérations (LoadKnowledge, UnLoadKnowledge, GetKnowledges).

3.4.5 Sauvegarde et restauration des services

Afin de permettre une restauration rapide du système, tous les services d'administration décrits précédemment exposent deux opérations permettant de sauvegarder (DumpDatabase) et restaurer (ReloadDatabase) leur base de données interne.

3.4.6 Communication inter-service

Les services techniques ont besoin de communiquer entre eux, par exemple lorsqu'un utilisateur change son mot de passe. En effet, le service de gestion des utilisateurs doit alors récupérer la liste de tous les proxies auprès de l'annuaire, et propager la modification auprès des services qu'il vient d'obtenir.

La «sécurisation» des communications inter-service est pour l'instant gérée par un token commun à tous les services. Toutes les opérations de ce type (préfixée par convention par un `__`) prennent en paramètre une chaîne de caractère correspondant au token de l'émetteur.

début

```

si TokenRecu == MonToken alors
  exécuter l'opération
sinon
  lever une exception (accès invalide)

```

```

finsi
fin

```

3.4.7 Configuration des services techniques

Les services sont paramétrables à l'aide d'un fichier `web.config` qui définit certains paramètres des applications web. On trouve en annexe D une description précises des possibilité de configuration des services.

3.5 Cycle de vie des Orchestrations de Contrôles

3.5.1 Chargement d'une orchestration

1. Un *Compositeur* décrit son contrôle en ODML
2. Il invoque l'opération `LoadOrchestration` du proxy visé :
 - (a) Si aucun contrôle n'est présent, l'orchestration est mémorisée pour interprétation.
 - (b) Si un contrôle interprétable est déjà présent, le processus de fusion intervient :
 - i. Si la fusion avec ce contrôle est possible, le contrôle fusionné est calculé et remplace le précédent
 - ii. Si la fusion échoue, le *Compositeur* est informé de l'échec et est invité à ajouter les connaissances manquantes dans la base de connaissances.

3.5.2 Déchargement d'une orchestration

1. Un *Compositeur* choisit l'orchestration à decharger en invoquant l'opération `UnLoadOrchestration` du proxy visé.
2. Le processus de fusion est lancé avec les contrôles encore présents
 - (a) Si la fusion est possible, le nouveau contrôle est mémorisé pour interprétation.
 - (b) Si la fusion échoue, le déchargement est annulé, et le *Compositeur* est invité à ajouter les connaissances manquantes.

3.5.3 Exemple pas à pas

Date	Action	Orchestration Interprétée	Ensemble d'Orchestrations
$t = 0$	–	–	\emptyset
$t = 1$	<code>load(O_1)</code>	O_1	$\{O_1\}$
$t = 2$	<code>load(O_2)</code>	$O_{12} \equiv \text{merge}(O_1, O_2)$	$\{O_1, O_2\}$
$t = 3$	<code>load(O_3)</code>	$O_{123} \equiv \text{merge}(O_{12}, O_3)$	$\{O_1, O_2, O_3\}$
$t = 4$	<code>load(O_4)</code>	$O_{1234} \equiv \text{merge}(O_{123}, O_4)$	$\{O_1, O_2, O_3, O_4\}$
$t = 5$	<code>unload(O_2)</code>	$O_{134} \equiv \text{merge}(\text{merge}(O_1, O_3), O_4)$	$\{O_1, O_3, O_4\}$
$t = 6$	<code>unload(O_1)</code>	$O_{34} \equiv \text{merge}(O_3, O_4)$	$\{O_3, O_4\}$
$t = 7$	<code>unload(O_4)</code>	O_3	$\{O_3\}$
$t = 8$	<code>unload(O_3)</code>	–	\emptyset

3.6 Administration & Utilisation

Afin de simplifier le référencement des services, un script ASP permet de lister sur une page d'accueil tous les Services Web présents sur un serveur. La figure 3.5 montre le résultat de l'invoation de ce script dans un butineur.

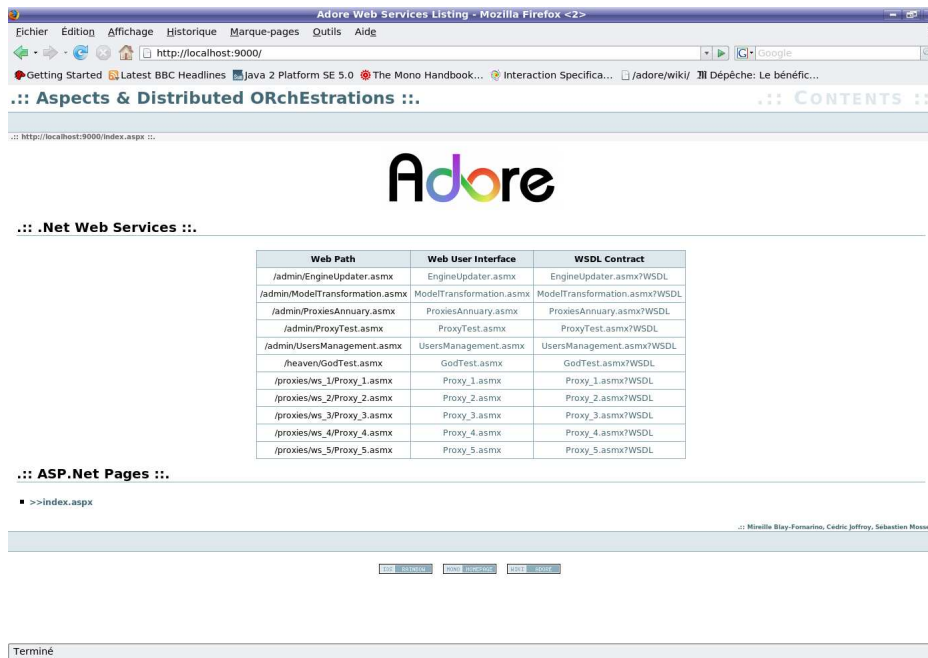


FIG. 3.3 – Interface Web d'ADORE

Il est ensuite possible de manipuler chacun des services de manière indépendante à l'aide de son interface Web générée par MONO (figure 3.4)

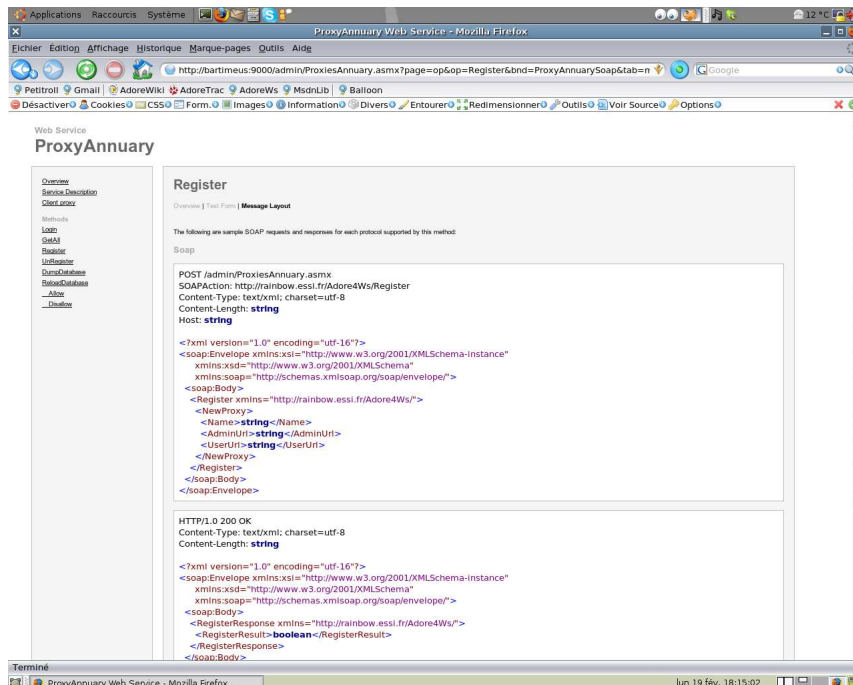


FIG. 3.4 – Interface MONO d'un Service Web

Deuxième partie

Un développement guidé par les modèles

Chapitre 4

Une plateforme multi-modèles

4.1 Les Objectifs

La plateforme ADORE est une réponse aux problèmes définis dans la partie précédente. Notre contribution se focalise sur les points suivants :

- La mise en place d'un mécanisme de description *orienté aspects* des orchestrations
- La fusion automatique des aspects de Services Web ainsi définis
- L'affichage graphique des orchestrations
- La persistance des orchestrations de contrôle au sein de la plateforme
- L'interprétation des orchestrations sur les Services Web.

4.2 Différents paradigmes ...

Aucun des paradigmes de programmation usuels ne permet aisément la mise en place de telles fonctionnalités. Le paradigme du *document* est adapté à la description des orchestrations de contrôle, mais leurs manipulations restent plus puissantes dans le paradigme des *objets*. Le paradigme *logique* est particulièrement adapté à la fusion d'orchestrations, mais utilise des mécanismes intrinsèquement différents des mécanismes *objets*. L'utilisation de *langages spécifique aux domaines* (DSL, *Domain Specific language*) reste une réponse puissante aux problèmes spécifiques de visualisation des orchestrations.

Notre démarche se veut ainsi une démarche multi-paradigme, tentant d'exploiter le meilleur de chaque paradigme précédemment cité. Afin de mettre en place une telle démarche, nous avons définis différents modèles, et des mécanismes de transformation inter-modèles permettant de passer de l'un à l'autre.

4.3 Différents modèles ...

A partir des objectifs cités au début de ce chapitre, nous définissons 4 modèles principaux au sein de la plateforme :

- Un modèle de description des orchestrations (ODML)
- Un modèle de fusion (OMSM)
- Un modèle interopérable .NET \Leftrightarrow JAVA (GOD)
- Un modèle d'exécution (DEMON)

La figure 4.1 montre les différentes relations entre les modèles précédemment cités. La figure 4.2 nomme explicitement chaque modèle utilisé dans ADORE.

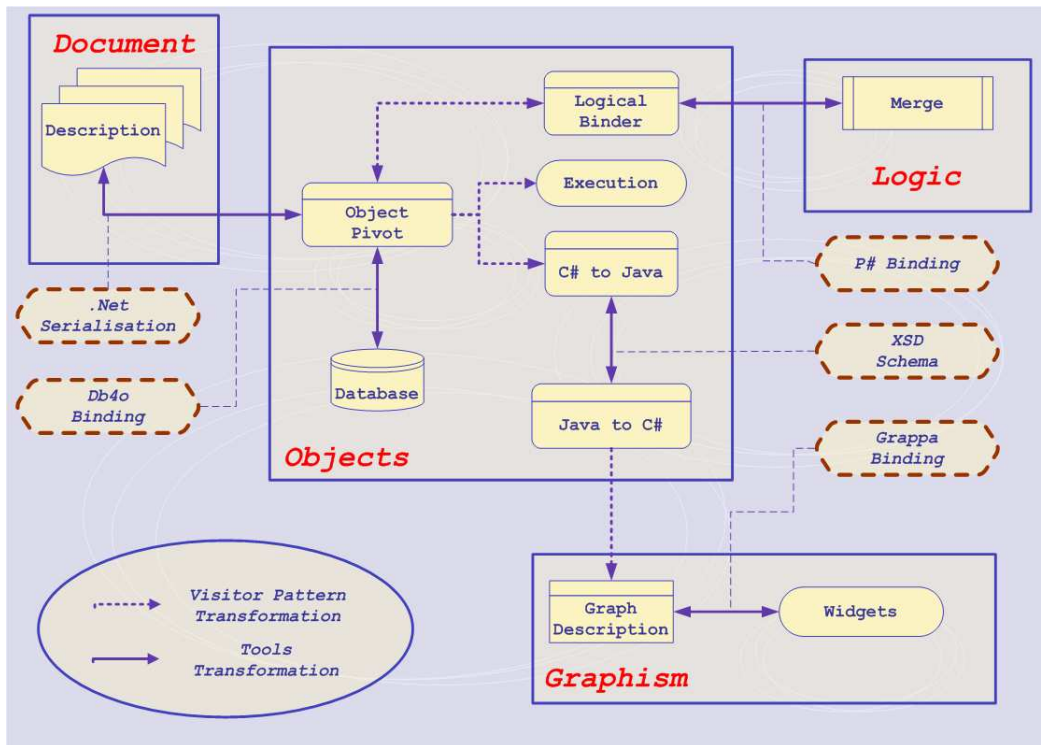
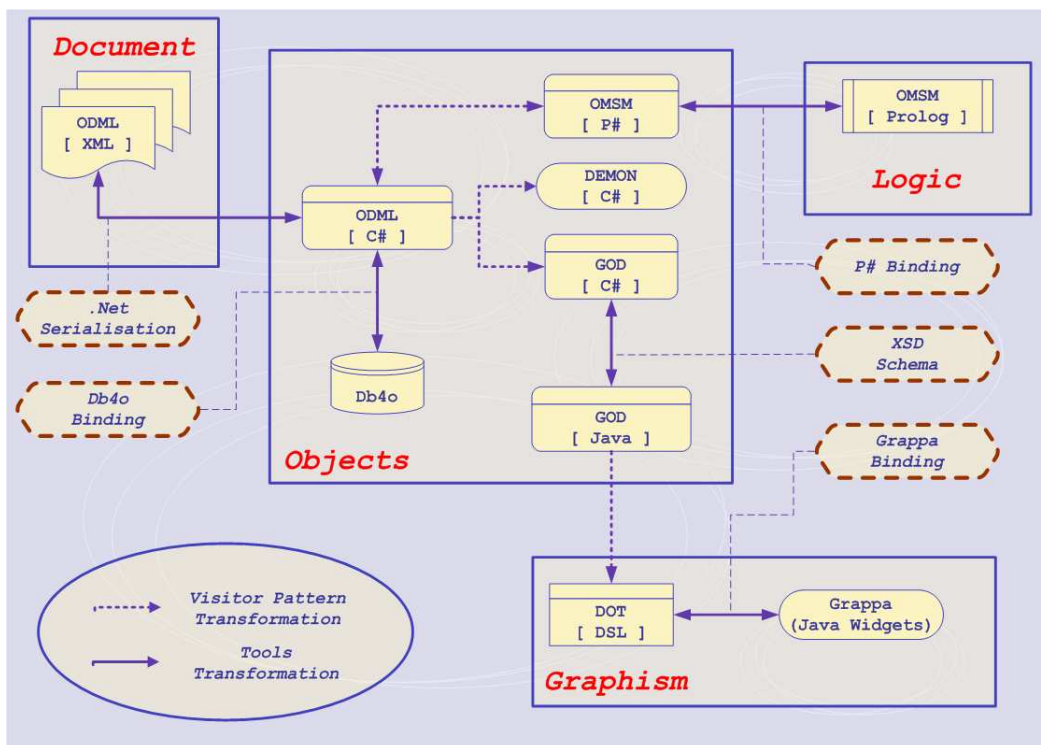
FIG. 4.1 – ADORE : Une plateforme *multi-modèles*

FIG. 4.2 – Modèles mis en jeu au sein d'ADORE

Les autres modèles sont laissés à la charge d'outils existants, tels P# pour la liaison *logique* \Leftrightarrow *objet*, DB4O pour la gestion de la persistance, ...

Nous choisissons comme pivot de transformation le modèle *objet* ODML, qui est au cœur des processus de transformations de modèles mis en jeu dans ADORE.

4.4 Différentes transformations

Notre approche du problème utilise fortement des outils existants. Nous sommes donc confrontés à des problèmes de transformations de modèles, et d'interopérabilité des outils utilisés.

Nous distinguons différents types de transformations de modèles. L'un des points clés du processus de développement utilisé est la nécessité de définir des modèles intermédiaires *passerelles* entre les divers outils utilisés, afin de les rendre interopérables.

Ce constat a donné lieu à la soumission d'un poster pour la conférence IDM07 (*Ingénierie des Modèles*), qui, partant des différents modèles mis en jeu au sein de la plateforme, expose les divers problèmes rencontrés et soulève la question de la maintenance et de l'évolution des transformations de modèles écrites manuellement.

Chapitre 5

ODML : Des aspects pour contrôler

5.1 Pivots de composition : une approche AOP

Rémi DOUENCE définit, dans [Douence, 2004], la programmation orientée aspects (AOP, *Aspects Oriented Programming*) de la manière suivante :

AOP is a set of language mechanisms which enable the introduction of non anticipated functionalities in a base application. Without these mechanisms, the code of the base application should be modified in several locations. AOP enables to modularize these functionalities.

On constate donc une analogie forte entre Orchestration de Contrôle et programmation par aspects.

5.1.1 Des aspects sur les Services Web ?

Les travaux actuels mêlant aspects et Services Web [Charfi and Mezini, 2006, Courbis and Finkelstein, 2005] visent à intégrer les propriétés non fonctionnelles au niveau des orchestrations et des moteurs d'exécution par des langages dédiés. Ce faisant, le programmeur se retrouve confronté à une architecture à trois niveaux (Services, Orchestrations et Aspects), chacun défini dans son propre formalisme.

Nous proposons donc d'utiliser le formalisme des orchestrations comme un moyen d'exprimer les aspects, puisqu'il nous paraît adapté au dynamisme requis par les utilisateurs des Services Web. Nous mettons ainsi en place des aspects sur les Services Web et non sur les orchestrations.

5.1.2 Définition des aspects sur les Services Web

Un aspect de Service Web se définit par un sélecteur S et un greffon (*advice*). Le sélecteur identifie la ou les opération(s) contrôlée(s). Un greffon se définit par un tuple $(A, P, Cond, Pred)$ où A est l'ensemble des activités, P l'ensemble des activités pivots ($P \subseteq A$), $Pred$ est la relation de précédence entre les activités et $Cond$ l'ensemble des conditions sur les activités définies dans A .

Nous avons pour l'instant pris en charge dans A , outre les activités pivots (*proceed*, *delegate*), l'affectation de variable, le retour de résultat, l'invocation d'une opération distante et la levée d'exception, qui correspondent à des instructions BPEL (*assign*, *reply*, *invoke*, *throw*) retranscrites dans ODML.

La manière de composer autour du point d'interception respecte les règles de fusion ISL [Blay-Fornarino et al., 2004] en utilisant des "flows" pour exprimer l'absence d'ordre de la même manière que dans [Douence et al., 2004] et en utilisant les conjonctions de conditions pour gérer les différentes branches de compositions.

5.1.3 Conflits de tissage

Une telle composition autour des activités pivots conduit à des conflits lorsqu'il existe des interactions entre les aspects définis. A ce jour, les points de conflits identifiés sont (i) la présence de différentes instructions `reply` au sein d'une même branche, (ii) la levée d'exceptions de manière concurrente, (iii) les accès concurrents en lecture et écriture à des variables et (iv) la présence de deux activités de délégation.

5.1.4 Processus de tissage

L'un des objectifs de la plateforme ADORE est d'automatiser le processus de fusion. Ce mécanisme reste complexe à mettre en place dans les paradigmes usuels de programmation. C'est pourquoi nous déportons cette problématique dans le paradigme de la *programmation logique*, en définissant en PROLOG le modèle de fusion des orchestrations de contrôles (OMSM, *Orchestration Model Supporting Merging*).

Ce modèle est décrit dans le chapitre 6.

5.2 Du document à l'objet

5.2.1 ODML, un langage d'aspects sur Services Web

ODML est le langage que nous définissons pour répondre à notre problématique. Contrairement à BPEL, il intègre :

- La notion de **selecteur** (attributs `target` et `overload`)
- la notion d'**activités pivots** (`proceed`, `delegate`)

Les activités composites `flow` et `sequence` permettent l'ordonnancement des activités atomiques présentes dans le langage. Nous défendons donc la même capacité d'expression que les approches usuelles du monde des aspects basées sur *before*, *around* et *after*.

Expression d'un *before* : Soit Asp_1 un aspect de Service Web, S_1 son sélecteur (l'opération contrôlée) et G_1 le greffon associé. On souhaite effectuer *avant* l'invocation de S_1 une opération O . On obtient la définition suivante pour Asp_1 :

$$Asp_1 = (S_1, G_1) \quad (5.1)$$

$$S_1 = \text{"Opération_Contrôlée"} \quad (5.2)$$

$$G_1 = (A, P, Cond, Pred) \quad (5.3)$$

$$G_1 = (\{(i_1, O), (i_2, \mathbf{proceed})\}, \{(i_2, \mathbf{proceed})\}, \emptyset, \{(i_1 < i_2)\}) \quad (5.4)$$

Ce qui s'exprime par une *sequence* en ODML :

```

1 <orchestration overload="Opération_Contrôlée">
  <sequence>
3   <invoke operation="O" />
   <proceed />
5 </sequence>
</orchestration>
```

Listing 5.1 – Définition d'un *before* en ODML

Expression d'un *around* : Soit Asp_2 un aspect de Service Web, S_2 son sélecteur et G_2 le greffon associé. On souhaite effectuer *autour* de l'invocation de S_2 une opération O . On obtient la définition suivante pour Asp_2 :

$$Asp_2 = (S_2, G_2) \quad (5.5)$$

$$S_2 = \text{"Opération_Contrôlée"} \quad (5.6)$$

$$G_2 = (A, P, Cond, Pred) \quad (5.7)$$

$$G_2 = (\{(i_1, O), (i_2, \mathbf{proceed}), (i_3, O)\}, \{(i_2, \mathbf{proceed})\}, \emptyset, \{(i_1 < i_2), (i_2 < i_3)\}) \quad (5.8)$$

Ce qui s'exprime par une *sequence* en ODML :

```

1 <orchestration overload="Opération_Contrôlée">
2   <flow>
3     <invoke operation="O" />
4     <proceed />
5     <invoke operation="O" />
6   </flow>
</orchestration>

```

Listing 5.2 – Définition d'un *around* en ODML

Expression d'un *after* : Soit Asp_3 un aspect de Service Web, S_3 son sélecteur et G_3 le greffon associé. On souhaite effectuer *après* l'invocation de S_3 une opération O . On obtient la définition suivante pour Asp_3 :

$$Asp_3 = (S_3, G_3) \quad (5.9)$$

$$S_3 = \text{"Opération_Contrôlée"} \quad (5.10)$$

$$G_3 = (A, P, Cond, Pred) \quad (5.11)$$

$$G_3 = (\{(i_1, O), (i_2, \mathbf{proceed})\}, \{(i_2, \mathbf{proceed})\}, \emptyset, \{(i_2 < i_i)\}) \quad (5.12)$$

Ce qui s'exprime par une *sequence* en ODML :

```

1 <orchestration overload="Opération_Contrôlée">
2   <sequence>
3     <proceed />
4     <invoke operation="O" />
5   </sequence>
</orchestration>

```

Listing 5.3 – Définition d'un *after* en ODML

On retrouve ici l'idée maître d'utiliser les langages d'orchestrations comme un moyen de définir des aspects sur les services, puisque les approches usuelles *before*, *around* et *after* sont intrinsèquement prises en charge par la définition même du langage.

On remarque aussi que la seule différence entre les greffons G_1 , G_2 et G_3 est la relation d'ordre $Pred$ qui leur est associés.

5.2.2 ODML, un modèle objet

Un document ODML peut être instancié dans un modèle objet permettant sa manipulation par la plateforme. On trouve en figure 5.1 un diagramme de classe UML général, et en figure 5.2 le détail des activités présentes au sein du modèle.

On obtient une *bijection* entre le langage (document) et son instantiation (objets) grâce aux mécanismes de *sérialisation/désérialisation* fournis par le framework .NET.

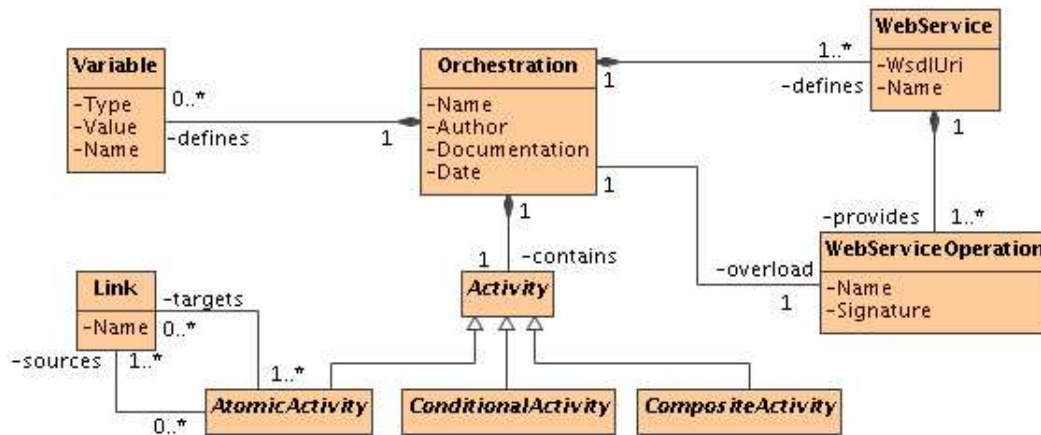


FIG. 5.1 – ODML : Diagramme de classes général

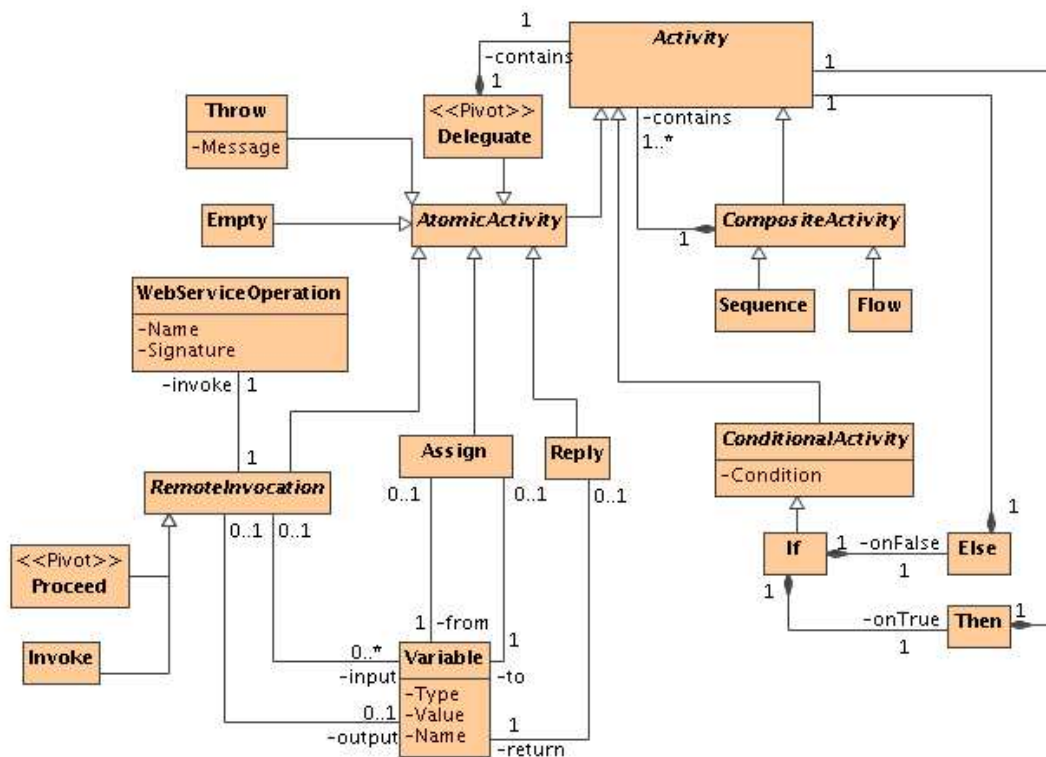


FIG. 5.2 – ODML : Diagramme de classe des Activités

Soit O_d une Orchestration de contrôle ODML et O_i son instantiation dans le modèle objet.

$$O_d = \text{serialize}(O_i)$$

$$O_i = \text{deserialize}(O_d)$$

5.3 Des Objets au cœur de la plateforme

La plupart des applications SOA utilisent le format XML comme articulation générale. Il s'agit en effet de l'un des buts de ce format (W3C, 1998) :

XML has been designed for ease of implementation and for interoperability

Or, au sein de la plateforme ADORE, l'articulation de la plateforme est le modèle objet .NET ODML. Notre choix a été porté par les faits suivants :

- Passer du document à l'objet est transparent au sein de .NET
- La plateforme à de toutes façons besoin de .NET pour s'exécuter
- Les langages de traitement de documents XML sont limités
- La manipulation d'un arbre d'objet est simple

Chapitre 6

OMSM : De la logique pour fusionner

6.1 Programmation Logique : un paradigme adéquat

6.1.1 Indépendant de la description des Orchestrations

Le modèle OMSM fait partie de la base commune à l'approche de Clémentine NEMO. Or, ses travaux utilisent comme formalisme de description des orchestrations le langage BPEL, à l'inverse d'ADORE, qui utilise ODML.

6.1.2 Puissance de l'interpréteur logique

Au contraire des langages *impératifs* usuels, la programmation logique est basée sur la déclaration de faits et de règles (clauses de Horn), puis sur le calcul d'une solution à un problème par application du théorème de résolution en logique du premier ordre.

A partir d'une base de règles décrivant le processus de fusion, il suffit ensuite de déclarer des faits au sein de l'interpréteur, puis de demander le calcul de l'adaptation résultante.

L'interpréteur PROLOG, utilisant l'unification, la récursivité et le *backtracking*, établit un arbre de dérivation et produit automatiquement l'orchestration fusionnée, si celle-ci existe. Déléguer une grande partie du travail de résolution à l'interpréteur permet de se focaliser sur la description des règles de fusion, laissant l'interpréteur maître de la partie calculatoire.

6.2 OMSM en détail

6.2.1 Définition du modèle

Orchestration : Une orchestration O est définie par un tuple (O_{id}, I, Ord, C) , où O_{id} est un identifiant unique, I l'ensemble des instructions utilisées dans O , Ord la relation d'ordre partielle établie entre les éléments de I et C l'ensemble des conditions utilisées dans O .

Instruction : Une instruction i est définie par le tuple $(i_{id}, type, in, out, cond)$, où i_{id} est un identifiant unique, $type$ le type de l'activité modélisée, in l'ensemble des variables d'entrées, out l'ensemble des variables de sorties et $cond$ l'ensemble des conditions gardant cette instruction.

Remarque : Dans les exemples suivants, nous nommons directement les entités, ce qui explique l'absence d'identifiant unique au sein de leur définition :

$$(i_p, \mathbf{proceed}, \{v_1, v_2\}, \{v_3\}, \{c_1\}) \iff i_p = (\mathbf{proceed}, \{v_1, v_2\}, \{v_3\}, \{c_1\})$$

Formalisme OMSM :

$$O = (O_{id}, I, Ord, C) \quad (6.1)$$

$$i \in I \Rightarrow i = (i_{id}, type, in, out, cond) \quad (6.2)$$

$$t \in type \Rightarrow t \in \{assign, reply, invoke, throw, proceed, delegate\} \quad (6.3)$$

6.2.2 D'ODML à OMSM, et réciproquement ...**Du PROLOG en .NET ?**

ADORE utilise le framework P# [Cook, 2004] pour établir la liaison entre le modèle objet ADORE et OMSM. Cet outil fournit un générateur de code capable de transformer des règles PROLOG en classes C#, et implémente un interpréteur capable de les évaluer.

Transformation ODML → OMSM

Il n'est pas possible de contrôler la génération des classes produites par l'outil de génération de code P#. Afin de finaliser la liaison entre le modèle objet ADORE et le modèle objet OMSM généré par P#, il est nécessaire d'effectuer une nouvelle transformation de modèle, modélisé par le signe « \rightsquigarrow ».

Cette transformation est effectuée à l'aide d'un schéma de conception *Visiteur* traversant le graphe d'objet ODML. Les transformations modélisées par le signe « \rightarrow » sont quant à elles effectuées automatiquement par les outils utilisés au sein de la plateforme.

$$\text{Document ODML} \rightarrow \text{Objets ODML} \rightsquigarrow \text{Modèle P\#} \rightarrow \text{OMSM}$$

Transformation OMSM → ODML

Cette transformation est la réciproque de la précédente :

$$\text{OMSM} \rightarrow \text{Modèle P\#} \rightsquigarrow \text{Objets ODML} \rightarrow \text{Document ODML}$$

6.2.3 Processus de fusion

Le processus de fusion suit le scénario suivant :

1. Transformation des orchestration ODML en des instances OMSM
2. Invocation de la règle de fusion OMSM
 - (a) si un conflit de fusion est détecté :
 - i. Enrichissement de la base de connaissance
 - ii. aller en 2
 - (b) s'il n'y a pas de conflit, aller en 3
3. Transformer l'orchestration OMSM résultante en ODML

6.3 Une fusion par l'exemple**6.3.1 Orchestration O_1 : Certification électronique**

On s'intéresse ici à l'orchestration O_1 définie page 6.

Description ODML :

```

2 <orchestration target="AccountManager" overload="Transfert">
  <sequence>
4     <invoke webservice="AccountChecker" operation="isCertified" output="c1">
      <parameters>
6         <parameter variable="To" />
      </parameters>
    </invoke>
8     <if condition="c1">
      <then>
10        <proceed>
          <parameters>
12            <parameter variable="From" />
13            <parameter variable="To" />
14            <parameter variable="Amount" />
          </parameters>
15        </proceed>
      </then>
16    <else>
      <throw message="Account [To] isn't certified" />
17    </else>
18    </if>
19 </sequence>
20 </orchestration>

```

Listing 6.1 – ODML : description de O_1 **Instanciation OMSM :**

$$O_1 = (\{i_1, i_2, i_3\}, \{(i_1 < i_2), (i_1 < i_3)\}, \{c_1\}) \quad (6.4)$$

$$i_1 = (\text{invoke}, \{To\}, \{c_1\}, \emptyset) \quad (6.5)$$

$$i_2 = (\text{throw}, \{\text{"Account [To] isn't certified"}\}, \emptyset, \{-c_1\}) \quad (6.6)$$

$$i_3 = (\text{proceed}, \{From, To, Amount\}, \emptyset, \{c_1\}) \quad (6.7)$$

6.3.2 Orchestration O_2 : Fraude Bancaire

On s'intéresse ici à l'orchestration O_2 définie page 6.

Description ODML :

```

1 <orchestration target="AccountManager" overload="Transfert">
  <sequence>
3     <invoke webservice="BankingFraud" operation="isBlockedAccount" output="c2">
      <parameters>
5         <parameter variable="From" />
      </parameters>
    </invoke>
7     <if condition="c2">
      <then>
9         <throw message="Account [From] isn't available" />
10        </then>
11    <else>
12    <proceed>
      <parameters>
13        <parameter variable="From" />
14        <parameter variable="To" />
15        <parameter variable="Amount" />
      </parameters>
16    </proceed>
17 </sequence>
18 </orchestration>

```

```

21     </else>
22     </if>
23 </sequence>
</orchestration>

```

Listing 6.2 – ODML : description de O_2 **Instanciation OMSM :**

$$O_2 = (\{i_{11}, i_{12}, i_{13}\}, \{(i_{11} < i_{12}), (i_{11} < i_{13})\}, \{c_2\}) \quad (6.8)$$

$$i_{11} = (\mathit{invoke}, \{From\}, \{c_2\}, \emptyset) \quad (6.9)$$

$$i_{12} = (\mathit{throw}, \{"Account [From] isn't available"\}, \emptyset, \{c_2\}) \quad (6.10)$$

$$i_{13} = (\mathit{proceed}, \{From, To, Amount\}, \emptyset, \{\neg c_2\}) \quad (6.11)$$

6.3.3 Calcul de $O_3 \equiv \mathit{merge}(O_1, O_2)$

Cette section décrit le processus de fusion tel qu'il s'exécute lorsqu'il tente de calculer l'orchestration résultante de la fusion de O_1 et O_2 .

Unification des activités autour des pivots :

Le processus de fusion va construire une nouvelle orchestration en utilisant les pivots ($\mathit{proceed}$) comme point de jointure afin de construire la nouvelle relation d'ordre partielle entre les différentes instructions.

$$O_3 = (\{i_1, i_2, i_{11}, i_{12}, i_p\}, \{(i_1 < i_2), (i_1 < i_p), (i_{11} < i_{12}), (i_{11} < i_p)\}, \{c_1, c_2\}) \quad (6.12)$$

$$i_1 = (\mathit{invoke}, \{To\}, \{c_1\}, \emptyset) \quad (6.13)$$

$$i_2 = (\mathit{throw}, \{"Account [To] isn't certified"\}, \emptyset, \{\neg c_1\}) \quad (6.14)$$

$$i_{11} = (\mathit{invoke}, \{From\}, \{c_2\}, \emptyset) \quad (6.15)$$

$$i_{12} = (\mathit{throw}, \{"Account [From] isn't available"\}, \emptyset, \{c_2\}) \quad (6.16)$$

$$i_p = (\mathit{proceed}, \{From, To, Amount\}, \emptyset, \{c_1, \neg c_2\}) \quad (6.17)$$

Recherche de conflits lors des conjonctions de conditions :

Le processus de fusion va calculer les conjonctions de toutes les conditions définies au sein de l'orchestration, et décider des instructions à insérer dans l'orchestration résultante :

$$c_1 \wedge c_2 \Rightarrow \{i_{12}\} \quad (6.18)$$

$$c_1 \wedge \neg c_2 \Rightarrow \{i_p\} \quad (6.19)$$

$$\neg c_1 \wedge c_2 \Rightarrow \{i_2, i_{12}\} \equiv \langle\langle \text{Conflit throw/throw} \rangle\rangle \quad (6.20)$$

$$\neg c_1 \wedge \neg c_2 \Rightarrow \{i_2\} \quad (6.21)$$

Enrichissement de la base de connaissance :

Le développeur est informé du conflit détecté en 6.20, et décide de lever l'exception contenu dans l'instruction i_{12} . Il rajoute donc l'information adéquate dans la base de connaissances, et le processus de fusion reprend. La gestion des conflits est explicité plus précisément en section 6.4.

$$\neg c_1 \wedge c_2 \Rightarrow \{i_{12}\} \quad (6.22)$$

Propagation des conjonctions :

Il faut maintenant propager au sein de l'orchestration résultante les conjonctions calculées. On remarque que la garde portant sur i_{12} peut être aisément simplifiée :

$$(c_1 \wedge c_2) \wedge (\neg c_1 \wedge c_2) \equiv c_2 \wedge (c_1 \wedge \neg c_1) \equiv c_2$$

On obtient comme résultat de cette propagation l'orchestration OMSM suivante :

$$O_3 = (\{i_1, i_2, i_{11}, i_{12}, i_p\}, \{(i_1 < i_2), (i_1 < i_p), (i_{11} < i_{12}), (i_{11} < i_p)\}, \{c_1, c_2\}) \quad (6.23)$$

$$i_1 = (\text{invoke}, \{To\}, \{c_1\}, \emptyset) \quad (6.24)$$

$$i_2 = (\text{throw}, \{\text{"Account [To] isn't certified"}\}, \emptyset, \{\neg c_1, \neg c_2\}) \quad (6.25)$$

$$i_{11} = (\text{invoke}, \{From\}, \{c_2\}, \emptyset) \quad (6.26)$$

$$i_{12} = (\text{throw}, \{\text{"Account [From] isn't available"}\}, \emptyset, \{c_2\}) \quad (6.27)$$

$$i_p = (\text{proceed}, \{From, To, Amount\}, \emptyset, \{c_1, \neg c_2\}) \quad (6.28)$$

Transformation de modèle :

L'orchestration résultante est maintenant transformée en une instance ODML par la transformation de modèle OMSM \rightarrow ODML.

6.4 Gestion des conflits

6.4.1 Définition

Un conflit est identifié par un tuple (c, g, i, I) où c est le contexte, g désigne le greffon en cours de calcul, i identifie le conflit, et I est l'ensemble des informations relatives au conflit. Le contexte comprend les conditions dans lesquelles le conflit est apparu. L'identifiant du conflit caractérise la liste des informations attendues.

A ce jour, les identifiants de conflits sont :

- *concurrentAccessTo*,
- *multipleThrowActivities*,
- *multipleReplyActivities*,
- *multipleDelegate*.

6.4.2 Résolution des conflits

Lorsqu'un conflit est détecté, la résolution du conflit est recherchée dans la base de connaissances. La base de connaissances est alimentée par les programmeurs soit en réponse à une détection de conflits, soit par des connaissances générales sur l'application. La base de connaissance est constituée d'un ensemble de règles. Une règle dans la base de connaissance comprend le contexte identifiant les conditions d'application de la règle et les activités en conflits. Les conséquences de l'application de la règle sont une modification du greffon en cours de calcul.

6.4.3 Dans l'exemple précédent ...

Dans notre exemple, on détecte en 6.20 un conflit C de la forme suivante :

$$C = (c, g, i, I)$$

$$c = \{\neg c_1 \wedge c_2\}$$

$$g = O_3$$

$i = \mathit{multipleThrowActivities}$

$I = \{throw("Account [to] isn't certified"), throw("Account [From] isn't available")\}$

Un langage dédié est défini qui permet dans les cas récurrents de simplement décrire la résolution d'un conflit. Dans notre exemple, le *gouverneur* précisera :

```

1 absorbent(_,
2     throw('Account_[From]_isn\'t_available'),
3     throw('Account_[To]_isn\'t_certified')).

```

L'absence de conditions stipule que la première levée d'exception est absorbante vis-à-vis de la seconde quelles que soient les conditions.

Cette déclaration est équivalente à ajouter dans la base de connaissance la règle suivante :

```

1 conflictSolve(Cond,
2     G,
3     multipleThrowActivities,
4     [ throw ('Account_[To]_isn\'t_certified'),
5       throw ('Account_[From]_isn\'t_available') ],
6     NewG) :-
7     removeActivity(Cond, throw('Account_[To]_isn\'t_certified'), G, NewG).

```

Chapitre 7

GOD : Des objets pour afficher

7.1 Visualisation des Orchestrations

L'un des plus grands inconvénients des dialectes XML est leur lourdeur. En effet, une orchestration simple représente rapidement de nombreuses lignes de XML lorsqu'elle est exprimée en ODML.

Intuitivement, nous définissons des modèles graphiques de descriptions des orchestrations. Dans ce document, il s'agit du formalisme des diagrammes d'activités UML, qui est utilisé pour proposer au lecteur une visualisation claire des orchestrations. Il existe cependant d'autres formalismes de descriptions des processus métiers, comme BPMN [White, 2006a, White, 2006b].

La définition d'un outil d'administration de la plateforme ADORE a été posé en sujet de projet de seconde année en janvier 2007 au département informatique de l'ÉPU POLYTECH'NICE SOPHIA-ANTIPOLIS. Parmi les objectifs de ce projet était présent l'écriture d'un module de visualisation graphique des orchestrations de contrôle chargées dans la plateforme.

7.2 Un nouveau modèle ?

Afin de fournir une visualisation des orchestrations sous forme de graphe, il était plus simple pour les étudiants en charge de la réalisation de ce projet (Alix MERCIER, Denis TEA, Rémy CHECK et Abdelhak LAKNIN) d'utiliser des outils disponibles dans le monde JAVA.

7.2.1 Hétérogénéité des langages ...

Les Services Web pronant l'hétérogénéité des langages, il paraissait aisé de renvoyer depuis les services concernés des objets ODML. Une simple visite des graphes d'objets aurait permis la visualisation des orchestrations.

Or, si le modèle objet ODML est exprimable dans un schéma XSD (formalisme utilisé par la technologie Service Web afin de décrire les structures de données échangées entre les services), il est mal interprété par le générateur de code AXIS utilisé par la communauté JAVA.

Au final, on se rend compte à l'usage que l'hétérogénéité des langages proné par les Services Web n'est pas encore assez aboutie pour arriver à faire transiter correctement des objets définis dans un modèle reflexif et utilisant un fort héritage (ce qui est le cas d'ODML).

7.2.2 Vers un modèle supportant l'hétérogénéité ...

Il restait aux étudiants les choix suivants :

1. *Patcher* manuellement les classes générées par AXIS
2. Ne pas utiliser AXIS et manipuler directement les documents SOAP échangés
3. Ecrire un *parseur* JAVA capable d'interpréter un document ODML

ADORE étant un projet exploratoire, aucune des solutions proposées ci-dessus n'était perenne pour la plateforme. En effet, le moindre changement du modèle objet ADORE aurait impliqué un redéploiement des applicatifs d'administration de la plateforme ...

La solution retenue est d'utiliser un nouveau modèle simplifié, dont les instances peuvent transiter entre .NET et JAVA. Ce modèle est appelé GOD, pour *Graphical Orchestration Description*.

7.3 GOD, un raffinement d'ODML

7.3.1 Aplatissement de l'héritage

Afin de ne pas retomber dans un modèle utilisant un trop fort héritage, le modèle objet ADORE est simplifié à l'extrême. On perd notamment la notion de typage des activités. Le type des activités instanciées est transformée en un attribut «*chaîne de caractères*» dans le modèle GOD.

Remarque : Cet héritage *a plat* n'a plus de sens en terme de modélisation objet. Il est en effet plus efficace d'utiliser les concepts de *polymorphisme* et de *surcharge* sur les objets pivots de la plateforme. Or, ce modèle servant uniquement à afficher les orchestrations sous forme graphique, il est aisé d'analyser l'attribut *Odm1Type* d'une instance d'*Activity* et d'en déterminer la forme du nœud à afficher.

7.3.2 Disparition des activités composites

Les activités composites présentes dans ODML sont elles-mêmes composées d'activités (0 ... *n*). Or, un tel modèle reste hasardeux lorsque l'on tente d'en faire transiter les instances entre JAVA et .NET.

Ces activités doivent donc disparaître de GOD. Nous les remplaçons par une relation d'ordre partiel sur les activités, de la même manière qu'au sein d'OMSM. Chaque activité possède un identifiant unique, et l'orchestration dispose d'une table de *liens* entre ces identifiants.

$$\begin{array}{ccccc} \text{ODML} & \rightsquigarrow & \text{OMSM} & \rightsquigarrow & \text{GOD} \\ \text{sequence}(i_1, i_2) & \iff & i_1 < i_2 & \iff & \text{link}(\text{source} := i_1, \text{dest} := i_2) \end{array}$$

7.3.3 Transformation de Modèle

La transformation passant du modèle objet ODML au modèle objet GOD est effectué à l'aide d'un schéma de conception *Visiteur* qui parcourt l'orchestration ODML.

Au sein de la plateforme, cette transformation est exposée par un service nommé *ModelTransformation*. L'opération *Odm12God* de ce service prend en entrée un document ODML et retourne son instantiation GOD en effectuant la transformation suivante :

$$\text{Document ODML} \rightarrow \text{Objets ODML} \rightsquigarrow \text{Objets GOD}$$

7.3.4 Modélisation

Diagramme de classes UML

La figure 7.1 donne le diagramme de classe du modèle GOD défini sur la plateforme ADORE.

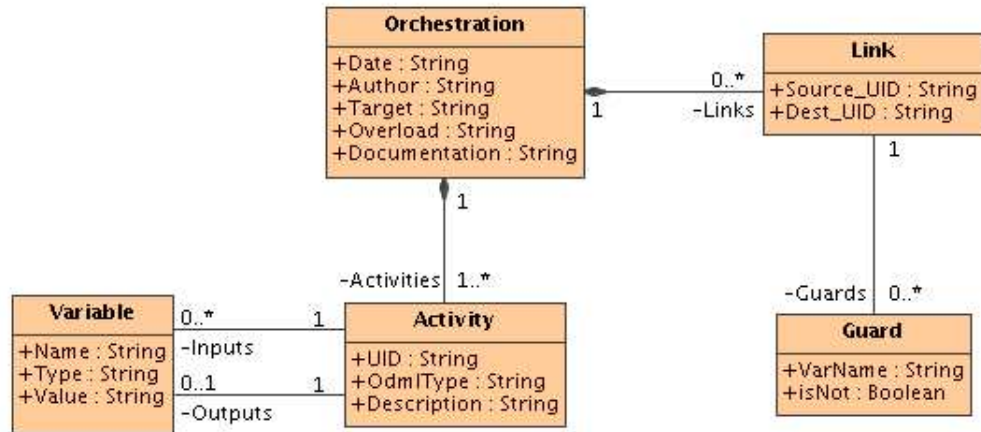


FIG. 7.1 – GOD : Diagramme de classes UML

Schéma XSD

On trouve en annexe C le schéma XSD décrivant le modèle GOD.

7.4 Affichage des Orchestrations

Une fois l'instance GOD reçue dans le monde JAVA, l'outil d'administration va effectuer une nouvelle transformation de modèle afin d'obtenir une description DOT (*Domain Specific Language* permettant la description de graphes) de l'orchestration.

La description DOT obtenue est ensuite enrichie par un nouvel outil (GRAPHVIZ) afin d'obtenir un *layout* agréable pour le graphe. On note DOT* ce modèle augmenté.

Cette orchestration est ensuite passée à un interpréteur DOT nommé GRAPPA, qui va effectuer une transformation de DOT* vers des *widgets* JAVA.

Cycle de transformation :

Afin d'être affiché, un document ODML suit le processus de transformation ci-dessous :

Document ODML → Objets ODML ⇔ Objets GOD ⇔ DOT → DOT* → Widgets JAVA

Chapitre 8

DEMON : Des *threads* pour exécuter

8.1 Une évaluation distribuée

Dans le cadre de la plateforme ADORE, les orchestrations de contrôle sont réparties auprès des proxies via l'utilisation des *proxies* ADORE et leur inscription dans une base de données persistantes.

Lors de l'invocation des opérations sur les *proxies* utilisateurs, si une orchestration de contrôle a été définie, elle doit alors être évaluée en lieu et place de l'invocation de l'opération contrôlée.

Chaque *proxy* utilisateur embarque donc un évaluateur ODML, afin de permettre ce comportement au sein de la plateforme.

Avantages : Contrairement aux approches BPEL, les orchestrations de contrôle ADORE ne sont pas centralisées dans un moteur d'exécution. Nous évitons donc les problèmes de ruptures de services qui peuvent apparaître avec un serveur centralisé surchargé ou injoignable.

8.2 *Distributed Execution Model for Orchestration*

8.2.1 Un nouveau modèle ?

Le modèle objet ODML permet la description d'Orchestrations de contrôle. Il est intrinsèquement lié aux documents qui permettent d'en créer des instances.

L'essence même du projet ADORE étant la fusion d'orchestrations de contrôles, et la mise en place de mécanismes permettant leur distribution au sein d'un système d'information, la mise en place de mécanismes d'exécutions efficace n'a jamais été sa priorité.

C'est pourquoi nous déléguons l'exécution des orchestrations aux mécanismes de programmation concurrente présents sur les plateformes .NET.

8.2.2 Ordonancement événementiel .NET

La plateforme .NET définit des mécanismes d'ordonancement événementiel des *threads*. Il est possible d'associer à chaque *thread* des *sources*. Lors de son lancement, un *thread* peut se mettre en attente de la fin de l'exécution de ses sources, et, une fois son exécution terminée, notifier qu'il a terminé son exécution, selon l'algorithme suivant :

début*Attendre la terminaison de mes sources***Exécuter mon comportement***Notifier ma terminaison***fin**

Il suffit donc d'opérer une transformation de modèle $ODML \rightsquigarrow DEMON$, puis de lancer de manière concurrente toutes les instructions présentes au sein de l'orchestration de contrôle transformée. L'ordonnancement se fait naturellement au sein de la plateforme, par notification interne. La transformation de modèle est effectuée à l'aide d'un schéma de conception *Visiteur* qui parcourt le graphe d'objets ODML pour produire une instance de DEMON conforme à la sémantique de l'orchestration donnée en entrée.

8.2.3 DEMON en détail

Orchestration : Une orchestration DEMON est définie par le tuple (O_{id}, V, C, I) , où O_{id} est un identifiant unique pour l'orchestration, V l'ensemble des variables utilisées dans l'orchestration, C l'ensemble des conditions et I l'ensemble des instructions utilisées.

Instruction : Une instruction est définie par le tuple $(i_{id}, S, Cond, in, out, type)$, où i_{id} est un identifiant unique, S l'ensemble des sources de l'instruction, $Cond$ l'ensemble des conditions gardant son exécution, in ses variables d'entrées, out sa variable de sortie, et $type$ son type.

Variable : Une variable est définie par le triplet $(n, vtype, val)$, où n est son nom, $vtype$ son type et val sa valeur.

Condition : Une condition est définie par le couple (n, val) , où n est son nom et val sa valeur.

Modélisation formelle :

$$O_{id} = (V, C, I) \quad (8.1)$$

$$i_{id} = (S, Cond, in, out, type) \quad (8.2)$$

$$s \in S \Rightarrow \exists i_{id} \in I, i_{id} = s \quad (8.3)$$

$$(n, val) \in Cond \Rightarrow \exists (n', val') \in C, n = n' \quad (8.4)$$

$$\forall i \in I, execute(i) \Rightarrow \forall i' \in S, \forall c \in Cond, end(i') \wedge valid(c) \wedge run(i) \wedge end(i) \quad (8.5)$$

$$c = (n, val) \in Cond, \Rightarrow valid(c) := \{\exists (n', val') \in C, n = n' \wedge val = val'\} \quad (8.6)$$

8.3 Validation par l'exemple

8.3.1 Une Orchestration ODML

On définit ci-après une orchestration ODML simple.

```

1 <orchestration>
  <flow>
3   <proceed />
  <sequence>
5     <invoke webservice="ws1" />
     <invoke webservice="ws2" />
7   </sequence>
  </flow>
9 </orchestration>

```

Listing 8.1 – Orchestration ODML pour exécution DEMON

8.3.2 Instanciation DEMON

$$O = (\emptyset, \emptyset, \{i_{proceed}, i_{ws_1}, i_{ws_2}\}) \quad (8.7)$$

$$i_{proceed} = (\emptyset, \emptyset, \emptyset, \emptyset, \mathbf{proceed}) \quad (8.8)$$

$$i_{ws_1} = (\emptyset, \emptyset, \emptyset, \emptyset, \mathbf{invoke}) \quad (8.9)$$

$$i_{ws_2} = (\{i_{ws_1}\}, \emptyset, \emptyset, \emptyset, \mathbf{invoke}) \quad (8.10)$$

$$(8.11)$$

8.3.3 Exécution pas à pas

Date	Action	$end(i_{proceed})$	$end(i_{ws_1})$	$end(i_{ws_2})$
$t = 0$	-	<i>false</i>	<i>false</i>	<i>false</i>
$t = 1$	$i_{proceed}, i_{ws_1}$	<i>true</i>	<i>true</i>	<i>false</i>
$t = 2$	i_{ws_2}	<i>true</i>	<i>true</i>	<i>true</i>

$t = 0$: Aucune instruction n'est lancée, et aucune n'a terminée.

$t = 1$: Les instructions $i_{proceed}$ et i_{ws_1} ne possèdent aucune sources, et sont donc exécutées. L'instruction i_{ws_2} attend la fin de sa source.

$t = 2$: La source de i_{ws_2} s'étant terminée, elle peut s'exécuter.

Troisième partie

Conclusion

Chapitre 9

Démonstration : «*Proof Of Concept*»

9.1 Objectifs de ce chapitre

Ce chapitre décrit l'application qui sert de base à la validation du travail effectué dans le cadre du projet ADORE.

On s'intéresse ainsi à la modification de l'application SÉDUITE, afin d'utiliser des orchestrations de contrôle et la plateforme ADORE pour améliorer l'évolutivité de l'applicatif.

9.2 SÉDUITE – EPUB

9.2.1 Implantation de l'application

L'application SÉDUITE est développée dans le cadre du RNTL FAROS, et déployée sur le site des TEMPLIERS de l'ÉCOLE POLYTECHNIQUE UNIVERSITAIRE DE NICE–SOPHIA ANTIPOLIS ainsi qu'à l'INSTITUT CLÉMENT ADER de Nice. On s'intéresse uniquement ici à la partie *affichage et récupération des informations* de la plateforme, qui est beaucoup plus vaste.

Le cœur du système est une plateforme à base de Services Web destinée à la diffusion d'informations au sein d'établissements scolaires. La plateforme est reliée à deux écrans sur le site de l'EPU :

- Un écran plasma géant (hall d'entrée, tout public)
- Un écran 17" (*coin café* du 4ème étage, enseignants)

Ces écrans diffusent les informations présentes dans la plateforme sur les écrans, selon le *profil* du public à qui elles sont destinées.

9.2.2 Vocabulaire & Fonctionnement de la plateforme SEDUITE

Services partenaires : SEDUITE intègre la notion de services partenaires. Ces services sont, pour le système, des sources d'informations, produisant des données affichables dans le formalisme interne de la plateforme SEDUITE.

Fournisseur d'information : Le fournisseur d'information (aussi appelé `InfoProvider`) est un service central de la plateforme. Adressé par les écrans de diffusion, il interroge ses partenaires et agrège les informations reçues en fonctions de filtres définis dans le *profil* du public visé.

Écran : Les écrans sont des écrans physiques exécutant un client SEDUITE capable d'adresser le fournisseur d'information. Les informations récupérées sont alors affichées séquentiellement sur le moniteur d'affichage.

La figure 9.1 décrit l'architecture de la plateforme SEDUITE déployée au sein de POLYTECH'NICE.

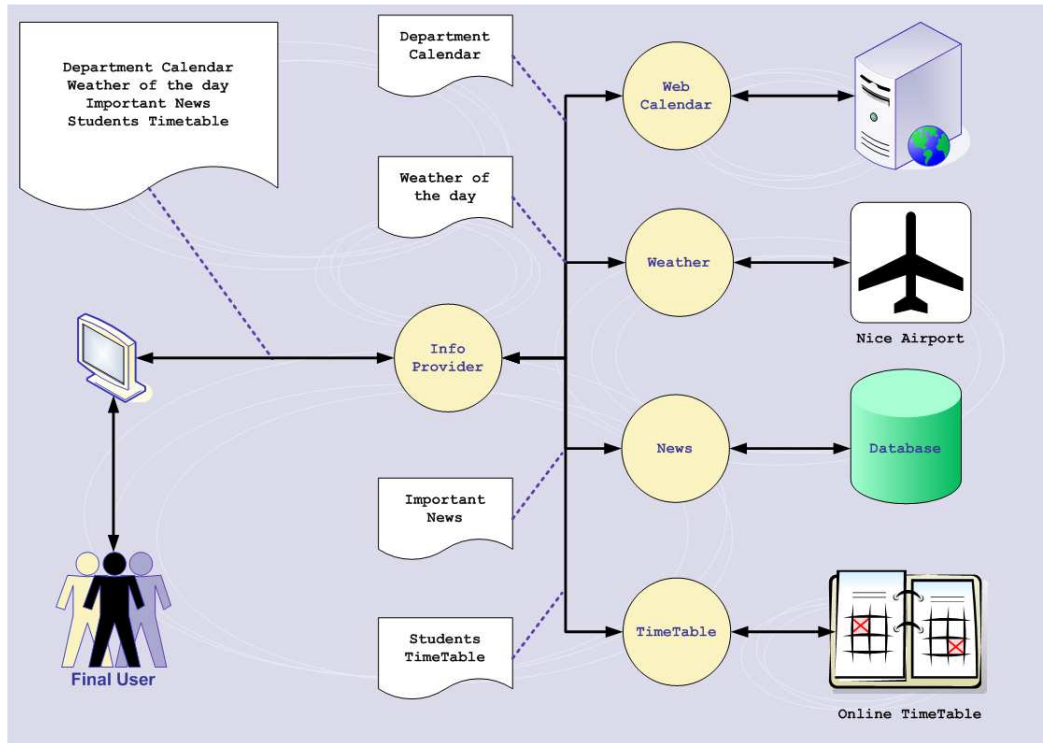


FIG. 9.1 – Architecture SÉDUITE déployée sur POLYTECH'NICE

9.2.3 Ajout d'un partenaire dans le fournisseur

En l'état, l'ajout d'un service partenaire implique sur la plateforme :

1. Un arrêt de l'application
2. La génération d'un *stub* pour le nouveau service
3. La déclaration d'un nouveau partenaire sur le service `InfoProvider`
4. La modification du code source de ce service, puis sa recompilation
5. Un redémarrage de la plateforme

Même si ces actions sont soutenues par (i) une forte documentation de la plateforme et (ii) l'utilisation d'un schéma de conception *Constructeur Virtuel*, on se rend compte intuitivement qu'une telle architecture ne supporte pas les évolutions rapides qui peuvent être demandée par les utilisateurs du système. Notre contribution à la plateforme est donc de combler ce vide !

9.3 Étude de cas & Exemple d'utilisation

9.3.1 Scénario

Le réseau de services présents sur l'intranet du site a évolué. On dispose maintenant de deux nouveaux services partenaires respectant le formalisme imposé par SÉDUITE :

- Un service fournissant les anniversaires du jour (Birthday).
- Un service fournissant les derniers projets publiés en vitrine (Showcase)

L'administrateur de la plateforme SÉDUITE souhaite donc intégrer facilement ces deux nouveaux services sur la plateforme. Il va donc utiliser des orchestrations de contrôle ADORE pour modifier le comportement de l'opération `GetInformations` du service `InfoProvider`.

9.3.2 Orchestration O_b : Ajout des Anniversaires du Jour

L'orchestration permettant d'ajouter la récupération des anniversaires du jour est simple : il s'agit d'invoquer le comportement habituel de l'opération en parallèle à l'invocation du service d'anniversaire, puis de renvoyer l'union des deux listes d'informations récupérées. La figure 9.2 décrit le diagramme d'activité d'une telle orchestration.

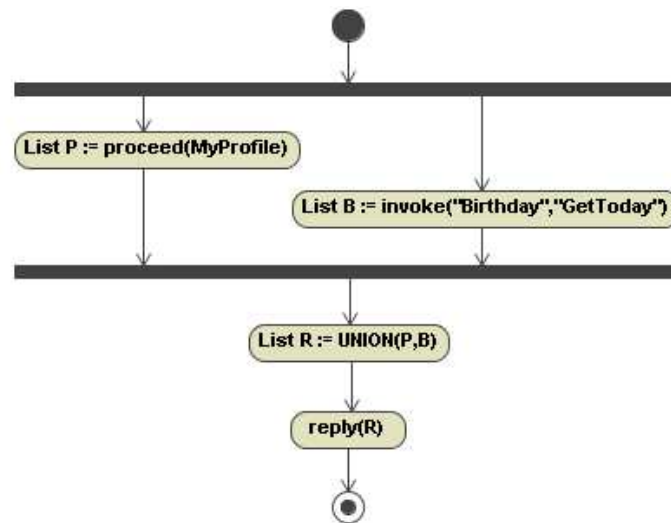


FIG. 9.2 – O_b : Ajout du service d'anniversaire au sein de SÉDUITE

Description ODML

```

1 <orchestration target="InfoProvider" overload="GetInformations" >
2   <!-- ... Suppression des details techniques ... -->
3
4   <!-- Nouveau comportement de l'operation -->
5   <sequence>
6     <flow>
7       <!-- Comportement Habituel -->
8       <proceed output="P">
9         <parameters>
10          <parameter variable="MyProfile" />
11        </parameters>
12      </proceed>

```

```

13     <!-- Invocation du nouveau partenaire -->
14     <invoke webservice="Birthday" operation="GetBirthday" output="B" />
15 </flow>
17 <!-- Composition par union des listes obtenues -->
18 <assign>
19     <from> P </from>
20     <to> R </to>
21 </assign>
22 <assign>
23     <behavior> UNION </behavior>
24     <from> B </from>
25     <to> R </to>
26 </assign>
27
28 <!-- Retour de la liste obtenue -->
29 <reply variable="R" />
30 </sequence>
31 </orchestration>

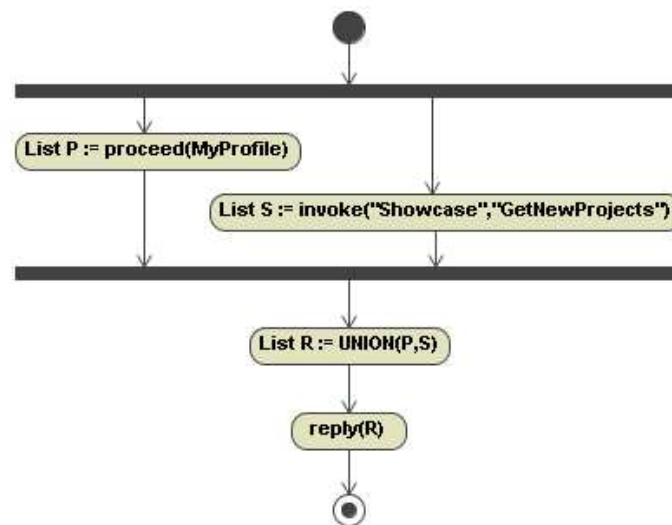
```

Listing 9.1 – O_b : Ajout des anniversaires dans le système SÉDUITE

Cette orchestration de contrôle est maintenant chargée sur le proxy généré en façade du service InfoProvider. Lors de l'invocation de l'opération modifiée, cette orchestration est exécutée en lieu et place du comportement habituel. L'administrateur n'a pas eu besoin de redémarrer le système, et la modification est immédiatement prise en compte par le système.

9.3.3 Orchestration O_s : Ajout de la Vitrine des Projets

De la même manière, il va ajouter une orchestration permettant la récupération de la liste des projets récemment ajoutés dans la vitrine des projets de POLYTECH'SOPHIA. La figure 9.3 montre le diagramme d'activité de l'orchestration de contrôle ainsi définie.

FIG. 9.3 – O_s : Ajout du service vitrine au sein de SÉDUITE

Description ODML

```

1 <orchestration target="InfoProvider" overload="GetInformations" >
  <!-- ... Suppression des details techniques ... -->
3
  <!-- Nouveau comportement de l'operation -->
5 <sequence>
  <flow>
7   <!-- Comportement Habituel -->
   <proceed output="P">
9     <parameters>
      <parameter variable="MyProfile" />
11    </parameters>
   </proceed>
13   <!-- Invocation du nouveau partenaire -->
   <invoke webservice="Showcase" operation="GetNewProjects" output="S" />
15 </flow>

17 <!-- Composition par union des listes obtenues -->
  <assign>
19   <from> P </from>
   <to> R </to>
21 </assign>
  <assign>
23   <behavior> UNION </behavior>
   <from> S </from>
25   <to> R </to>
  </assign>
27
  <!-- Retour de la liste obtenue -->
29 <reply variable="R" />
  </sequence>
31 </orchestration>

```

Listing 9.2 – O_s : Ajout de la vitrine des projets dans le système SÉDUITE

Comme une orchestration est déjà présente pour le sélecteur (*InfoProvider*, *GetInformations*), le processus de fusion se déclenche, et calcule O_{bs} , résultat de la composition des deux orchestrations précédentes.

Remarque : Chaque Orchestration de contrôle décrite est indépendante de l'autre. Elles peuvent avoir été écrites par des administrateurs différents.

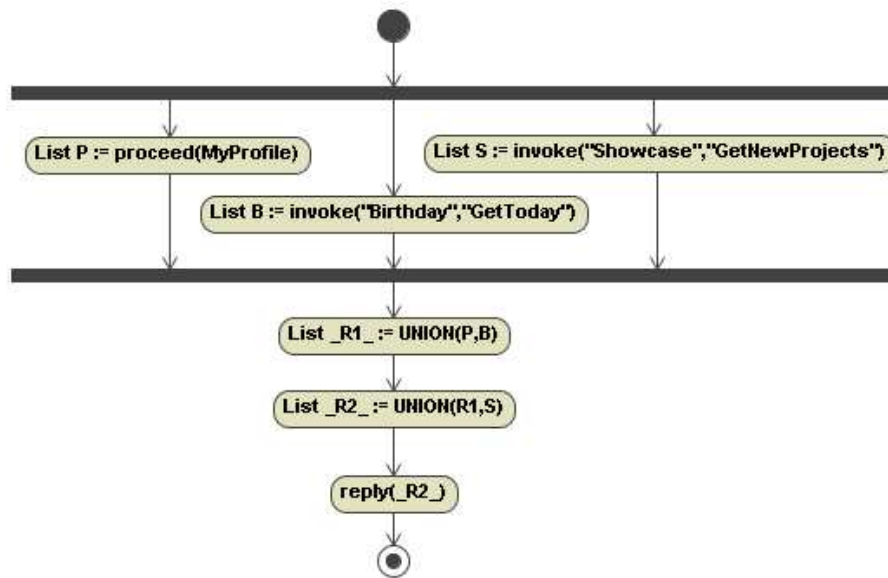
9.3.4 Orchestration $O_{bs} \equiv merge(O_b, O_s)$

Il n'y a aucun conflit de fusion pour calculer O_{bs} . Le processus de fusion effectue donc le travail suivant :

1. Lancer les trois invocations de service en parallèle
2. Composer par union les résultats obtenus
3. Renvoyer le résultat des unions

L'union de deux listes étant une opération commutative et associative, le résultat obtenu est le même, quel que soit l'ordre dans lequel les contrôles ont été posés.

La figure 9.4 montre le résultat de l'orchestration obtenue par fusion.

FIG. 9.4 – $O_{bs} \equiv merge(O_b, O_s)$

Description ODML

```

1 <orchestration target="InfoProvider" overload="GetInformations" >
  <!-- ... -->
3 <sequence>
  <flow>
5 <proceed output="P">
  <parameters> <parameter variable="MyProfile" /> </parameters>
7 </proceed>
  <invoke webservice="Showcase" operation="GetNewProjects" output="S" />
9 <invoke webservice="Birthday" operation="GetBirthday" output="B" />
  </flow>
11 <assign> <from> P </from> <to> _R1_ </to> </assign>
  <assign> <behavior> UNION </behavior> <from> S </from> <to> _R1_ </to> </assign>
13 <assign> <from> _R1_ </from> <to> _R2_ </to> </assign>
  <assign> <behavior> UNION </behavior> <from> B </from> <to> _R2_ </to> </assign>
15 <reply variable="_R2_" />
  </sequence>
17 </orchestration>

```

Listing 9.3 – $O_{bs} \equiv merge(O_b, O_s)$

Chapitre 10

Conclusion & Évolutions futures

10.1 Un projet exploratoire

La mise en place de la plateforme ADORE est un projet exploratoire, qui vise à utiliser de nombreux outils, paradigmes et formalismes.

Le déroulement de ce projet a donné lieu à la soumission de deux articles (*3ème Journée Francophone sur le Développement de Logiciels Par Aspects, 7th IFIP International Conference on Distributed Applications and Interoperable Systems*) et d'un poster (*3ème journées sur l'Ingénierie Dirigée par les Modèles*).

A la date d'impression de ce document, l'article JFDLPA est accepté sur cette conférence qui aura lieu le 26 mars 2007 à Toulouse (France), précédant les conférences LMO et IDM. Cet article concourt à une publication dans la revue L'OBJET de l'éditeur HERMÈS.

Dans le cadre de la mise en place des interfaces graphiques nécessaires à la manipulation des orchestrations, le projet a donné lieu à l'encadrement d'un projet de seconde année du département SCIENCES INFORMATIQUES de POLYTECH'NICE.

10.2 Une plateforme *quasi*-opérationnelle

L'objectif de ce projet restait avant tout la mise en place des mécanismes nécessaires à son implémentation. Dans le cadre du projet de dernière année ingénieur, il était matériellement impossible de produire (i) une base théorique solide sur laquelle s'appuyer et (ii) une implémentation globale et finalisée de la plateforme.

Nous avons ainsi développé chacune des fonctionnalités présentées dans ce document, et implémenté chacun des modèles utilisés par la plateforme. Chaque implémentation est opérationnelle mais pour l'instant isolée des autres.

L'assemblage final des modèles, et l'écriture de certaines transformations sont délégués à un stage de fin d'étude effectué par Sébastien MOSSER au sein de l'équipe RAINBOW de mars à novembre 2007.

Site du projet : Un site WIKI complétant ce document est disponible à l'adresse <http://rainbow.polytech.unice.fr/adore>

10.3 Évolutions envisagées

Disposant maintenant d'une solide base théorique, la plateforme ADORE peut évoluer afin de pérenniser sa valeur ajoutée.

10.3.1 Refonte de la plateforme SÉDUITE

Une fois l'implémentation de la plateforme ADORE finalisée, l'application SÉDUITE sera utilisée afin de valider l'approche, comme décrit au chapitre précédent. Une réécriture d'une partie de la plateforme sera nécessaire (génération des *proxies* ADORE, modification des clients écrans pour qu'ils adressent les *proxies* et non directement les services SÉDUITE), ainsi que la mise en place des orchestrations de contrôles la gouvernant.

10.3.2 Intégration du projet POLLUX

Dans le cadre de leur projet de fin d'étude, Vincent MAURIN et Férédrick MATHIOT travaille à la mise en place de mécanismes de gestions de contrats au sein d'une architecture basée sur des Services Web. Ce projet, dirigé par Philippe COLLET s'inclut aussi dans la plateforme SÉDUITE. On peut donc envisager à terme une fusion des deux projets, qui utilisent une architecture «à *proxies*» similaire.

10.3.3 Commutativité et Associativité des contrôles ADORE

De par son approche, différente des travaux précédents [Berger, 2001], la plateforme ADORE ne possède pas un mécanisme de fusion respectant la commutativité et l'associativité dans la pose des contrôles. Cette absence est néanmoins voulue dans le cadre du projet, puisqu'elle nous permet de mettre en place une plateforme très large et d'adresser de nombreuses problématiques.

Or, les orchestrations fusionnées par la plateforme sont dorénavant dépendantes de l'ordre dans lequel elles ont été posées. Nous défendons l'idée qu'une restriction des contrôles ADORE à un sous ensemble précis permettra de garantir l'associativité et la commutativité du processus de fusion ADORE.

10.3.4 Adaptation à la manipulation de composants d'IHM

Dans le cadre de son stage de fin d'étude, Cédric JOFFROY travaillera lui aussi sur la plateforme SÉDUITE, dans le but de manipuler des composants d'*Interfaces Homme Machines* et de faciliter leur intégration par composition dans la plateforme.

Une réutilisation des mécanismes de fusion mis en jeu au sein d'ADORE permettrait un développement rapide d'un prototype de fusion de composants d'IHM.

Quatrième partie

Annexes

Annexe A

Une orchestration BPEL

On peut trouver à l'adresse <http://www.activebpel.org/docs/tutorial.html> un exemple type des possibilités de BPEL. On trouve en figure A.1 une représentation graphique (dans le formalisme *Active EndPoint Design*) de l'orchestration suivante :

```
1 String processLoanRequest(message) {  
2     if (message.amount < 10000 && "low".equals(assessor.risk(message)))  
3         return "approved";  
4     else  
5         return approver.approve(message);  
6 }
```

Listing A.1 – *ActiveBpel* Loan Approval pseudo-code

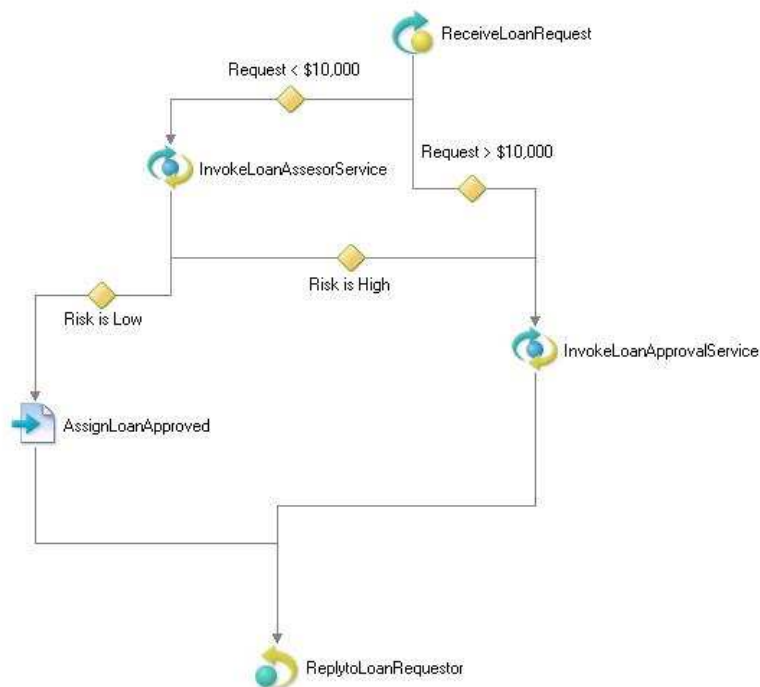


FIG. A.1 – *ActiveBpel* : Loan Approval Process

Le listing suivant montre le code source BPEL d'une telle orchestration :

```

2 <?xml version="1.0" encoding="UTF-8"?>
3 <process name="loanApprovalProcess"
4     suppressJoinFailure="yes"
5     targetNamespace="http://acme.com/loanprocessing"
6     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
7     xmlns:apns="http://tempuri.org/services/loanapprover"
8     xmlns:asns="http://tempuri.org/services/loanassessor"
9     xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
10    xmlns:lns="http://loans.org/wsd/loan-approval"
11    xmlns:loandef="http://tempuri.org/services/loandefinitions"
12    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
13  <partnerLinks>
14    <partnerLink myRole="approver" name="customer"
15        partnerLinkType="lns:loanApprovalLinkType"/>
16    <partnerLink name="approver" partnerLinkType="lns:loanApprovalLinkType"
17        partnerRole="approver"/>
18    <partnerLink name="assessor" partnerLinkType="lns:riskAssessmentLinkType"
19        partnerRole="assessor"/>
20  </partnerLinks>
21  <variables>
22    <variable messageType="loandef:creditInformationMessage" name="request"/>
23    <variable messageType="asns:riskAssessmentMessage" name="riskAssessment"/>
24    <variable messageType="apns:approvalMessage" name="approvalInfo"/>
25    <variable messageType="loandef:loanRequestErrorMessage" name="error"/>
26  </variables>
27  <faultHandlers>
28    <catch faultName="apns:loanProcessFault" faultVariable="error">
29      <reply faultName="apns:loanProcessFault" operation="approve"
30          partnerLink="customer" portType="apns:loanApprovalPT"
31          variable="error"/>
32    </catch>
33  </faultHandlers>
34  <flow>
35    <links>
36      <link name="receive-to-approval"/>
37      <link name="receive-to-assess"/>
38      <link name="approval-to-reply"/>
39      <link name="assess-to-setMessage"/>
40      <link name="assess-to-approval"/>
41      <link name="setMessage-to-reply"/>
42    </links>
43    <receive createInstance="yes" name="receive1" operation="approve"
44        partnerLink="customer" portType="apns:loanApprovalPT"
45        variable="request">
46      <source linkName="receive-to-approval"
47          transitionCondition="bpws:getVariableData('request', 'amount')>=10000"/>
48      <source linkName="receive-to-assess"
49          transitionCondition="bpws:getVariableData('request', 'amount')<10000"/>
50    </receive>
51    <invoke inputVariable="request" name="invokeApprover" operation="approve"
52        outputVariable="approvalInfo" partnerLink="approver"
53        portType="apns:loanApprovalPT">
54      <target linkName="receive-to-approval"/>
55      <target linkName="assess-to-approval"/>
56      <source linkName="approval-to-reply"/>
57    </invoke>
58    <invoke inputVariable="request" name="invokeAssessor" operation="check"
59        outputVariable="riskAssessment" partnerLink="assessor"
60        portType="asns:riskAssessmentPT">
61      <target linkName="receive-to-assess"/>
62      <source linkName="assess-to-setMessage"
63          transitionCondition="bpws:getVariableData('riskAssessment', 'risk')='low'"/>
64      <source linkName="assess-to-approval"
65          transitionCondition="bpws:getVariableData('riskAssessment', 'risk')!='low'"/>
66    </invoke>
67    <reply name="reply" operation="approve" partnerLink="customer"

```

```
68     portType="apns:loanApprovalPT" variable="approvalInfo">
    <target linkName="approval-to-reply"/>
    <target linkName="setMessage-to-reply"/>
70 </reply>
    <assign name="assign">
72     <target linkName="assess-to-setMessage"/>
    <source linkName="setMessage-to-reply"/>
74     <copy>
    <from expression="'approved'"/>
76     <to part="accept" variable="approvalInfo"/>
    </copy>
78     </assign>
    </flow>
80 </process>
```

Listing A.2 – *ActiveBpel* Loan Approval BPEL code

Annexe B

ODML : Syntaxe du langage

Le langage ODML permet la description d'orchestrations de contrôle sur la plateforme ADORE.

B.1 Orchestration de contrôle

Une orchestration de contrôle se définit de la manière suivante en ODML :

```
2 <?xml version="1.0" encoding="utf-16"?>
3 <orchestration xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xmlns="http://rainbow.essi.fr/Adore/odml"
6   target="WSName" overload="OperationName" >
7
8   <author> Author Name </author> <!-- multiplicity = 1 -->
9   <name> Orchestration Name </name> <!-- multiplicity = 1 -->
10  <date> Date of Creation </date> <!-- multiplicity = 1 -->
11
12  <documentation> <!-- multiplicity = 1 -->
13  Orchestration semantinc, in few lines
14 </documentation>
15
16  <webservices> <!-- multiplicity = 1 -->
17    <webservice name="Common_Name"
18      wsdl="WSDL URL" /> <!-- multiplicity = 1..* -->
19  </webservices>
20
21  <variables> <!-- multiplicity = 1 -->
22    <variable name="Variable Name"
23      type="Variable Type" /> <!-- multiplicity = 0..* -->
24  </variables>
25
26  <activity /> <!-- multiplicity = 1 -->
</orchestration>
```

Listing B.1 – Définition d'une Orchestration ODML

Une orchestration ODML redéfinit le comportement de l'opération `OperationName` sur le service `WSName` en le remplaçant par `activity`.

activity: L'activité décrivant le nouveau comportement peut être choisie parmi les activités décrites dans les sections suivantes.

B.2 Activités de contrôle

Les activités de contrôles permettent de contrôler l’invocation du comportement habituel du service contrôlé.

B.2.1 proceed

Sémantique : Cette activité représente le comportement initial de l’opération contrôlée.

Syntaxe :

```

1 <proceed
  output="Output Variable"                                <!-- multiplicity = 0..1 -->
3 <parameters>                                           <!-- multiplicity = 0..1 -->
  <parameter variable="Input Variable Name" /> <!-- multiplicity = 1..* -->
5 </parameters>
</proceed>

```

Listing B.2 – ODML : Syntaxe du proceed

Remarques :

- L’absence du champ `output` représente une opération dont la signature est void.
- Les paramètres d’entrées sont évalués en séquence.

B.2.2 delegate

Sémantique : Cette activité redéfinit le comportement originel de l’opération contrôlée en le remplaçant par une nouvelle activité ODML.

Syntaxe :

```

2 <delegate>
  <activity /> <!-- multiplicity = 1 -->
</delegate>

```

Listing B.3 – ODML : Syntaxe du delegate

B.3 Activité Atomiques

Les activités atomiques sont les éléments de base du langage ODML.

B.3.1 assign

Sémantique : Cette activité permet l’affectation d’une valeur à une variable.

Syntaxe :

```

1 <assign>
  <behavior> Assignment Behavior </behavior> <!-- multiplicity = 0..1 -->
3 <from> Litteral, Variable, Field </from> <!-- multiplicity = 1 -->
  <to> Variable, Field </to> <!-- multiplicity = 1 -->
5 </assign>

```

Listing B.4 – ODML : Syntaxe du assign

Remarques :

- Les comportements d'affectation supportés sont à l'heure actuelle :
 - append : concatène 2 listes
 - union : effectue l'union de 2 ensembles
 - inter : effectue l'intersection de deux ensembles
- L'affectation n'est valide que pour des types compatibles!

B.3.2 empty

Sémantique : Cette activité correspond au nop des langages assembleurs usuels.

Syntaxe :

```
1 <empty />
```

Listing B.5 – ODML : Syntaxe du empty

B.3.3 invoke

Sémantique : Cette activité permet l'invocation d'une opération sur un Service Web déclaré dans l'orchestration.

Syntaxe :

```
1 <invoke
2   webservice="Common Name"           <!-- multiplicity = 1 -->
3   operation="Operation Name"        <!-- multiplicity = 1 -->
4   output="Output Variable Name" >   <!-- multiplicity = 0..1 -->
5   <parameters>                       <!-- multiplicity = 0..1 -->
6     <parameter variable="aVariable" /> <!-- multiplicity = 1..* -->
7   </parameters>
8 </invoke>
```

Listing B.6 – ODML : Syntaxe du invoke

Remarque

- Les mêmes remarques que pour le proceed s'appliquent à cette activité.

B.3.4 reply

Sémantique : Cette activité permet à l'orchestration de positionner une valeur de retour.

Syntaxe :

```
2 <reply
   variable="Output" /> <!-- multiplicity = 1 -->
```

Listing B.7 – ODML : Syntaxe du reply

Remarque

- Il ne s'agit pas d'un return. L'évaluation de l'orchestration n'est pas stoppée par cette activité.

B.3.5 throw

Sémantique : Cette activité interrompt l'évaluation d'une orchestration en levant une exception au sein du processus de contrôle.

Syntaxe :

```
1 <throw>
2   <message>Explanations about the exception </message> <!-- multiplicity = 1 -->
3 </throw>
```

Listing B.8 – ODML : Syntaxe du throw

B.4 Activités Composites

Les activités composites permettent d'exécuter plusieurs activités atomiques au sein d'une même orchestration.

B.4.1 sequence

Sémantique : Les activités présentes dans une `sequence` sont évaluées selon l'ordre dans lequel elles ont été ordonnées dans l'orchestration.

Syntaxe :

```
1 <sequence>
2   <activity /> <!-- multiplicity = 1..* -->
3 </sequence>
```

Listing B.9 – ODML : Syntaxe de la sequence

B.4.2 flow

Sémantique : Les activités présentes dans un `flow` sont évaluées en parallèle.

Syntaxe :

```
1 <flow>
2   <activity /> <!-- multiplicity = 1..* -->
3 </flow>
```

Listing B.10 – ODML : Syntaxe du flow

B.5 Activités Conditionnelles

Les activités conditionnelles permettent de définir des branches au sein d'une orchestration et de garder l'évaluation de certaines activités sur une ou plusieurs conditions.

B.5.1 Définition des conditions

- Les conditions sont définies en utilisant une notation XML les représentant sous forme d'arbre.
- On dispose des opérateurs OR, AND et NOT.
 - Les opérateurs OR et AND sont d'arité n -aire.
 - L'opérateur NOT est unaire.
- Les variables mises en jeu dans les conditions doivent être des booléens.

B.5.2 if/then/else

Sémantique : Cette activité représente la conditionnelle usuelle des langages de programmation.

Syntaxe :

```
1 <if>
  <condition> Xml Contition </condition>      <!-- multiplicity = 1 -->
3  <then>
  <activity />                                <!-- multiplicity = 1 -->
5  </then>
  <else>
7  <activity />                                <!-- multiplicity = 1 -->
  </else>
9 </if>
```

Listing B.11 – ODML : Syntaxe du is/then/else

Annexe C

Orchestration GOD : Schéma XSD

```
1 <?xml version="1.0" encoding="utf-16">
2 <xsd:schema>
3   <xs:complexType name="Orchestration">
4     <xs:sequence>
5       <xs:element minOccurs="0" maxOccurs="1" name="Date" type="xs:string"/>
6       <xs:element minOccurs="0" maxOccurs="1" name="Name" type="xs:string"/>
7       <xs:element minOccurs="0" maxOccurs="1" name="Author" type="xs:string"/>
8       <xs:element minOccurs="0" maxOccurs="1" name="Target" type="xs:string"/>
9       <xs:element minOccurs="0" maxOccurs="1" name="Overload" type="xs:string"/>
10      <xs:element minOccurs="0" maxOccurs="1" name="Documentation" type="xs:string"/>
11      <xs:element minOccurs="0" maxOccurs="1" name="Activities"
12        type="s0:ArrayOfActivity"/>
13      <xs:element minOccurs="0" maxOccurs="1" name="Links" type="s0:ArrayOfLink"/>
14      <xs:element minOccurs="0" maxOccurs="1" name="WebServices"
15        type="s0:ArrayOfWebService"/>
16    </xs:sequence>
17  </xs:complexType>
18
19  <xs:complexType name="ArrayOfActivity">
20    <xs:sequence>
21      <xs:element minOccurs="0" maxOccurs="unbounded" name="Activity" nillable="true"
22        type="s0:Activity"/>
23    </xs:sequence>
24  </xs:complexType>
25
26  <xs:complexType name="Activity">
27    <xs:sequence>
28      <xs:element minOccurs="0" maxOccurs="1" name="UID" type="xs:string"/>
29      <xs:element minOccurs="0" maxOccurs="1" name="OdmlType" type="xs:string"/>
30      <xs:element minOccurs="0" maxOccurs="1" name="Description" type="xs:string"/>
31      <xs:element minOccurs="0" maxOccurs="1" name="Inputs" type="s0:ArrayOfVariable"/>
32      <xs:element minOccurs="0" maxOccurs="1" name="Outputs" type="s0:ArrayOfVariable"/>
33    </xs:sequence>
34  </xs:complexType>
35
36  <xs:complexType name="ArrayOfVariable">
37    <xs:sequence>
38      <xs:element minOccurs="0" maxOccurs="unbounded" name="Variable" nillable="true"
39        type="s0:Variable"/>
40    </xs:sequence>
41  </xs:complexType>
42
43  <xs:complexType name="Variable">
44    <xs:sequence>
45      <xs:element minOccurs="0" maxOccurs="1" name="Name" type="xs:string"/>
46      <xs:element minOccurs="0" maxOccurs="1" name="Type" type="xs:string"/>
47      <xs:element minOccurs="0" maxOccurs="1" name="Value" type="xs:string"/>
48    </xs:sequence>
49  </xs:complexType>
50 </xsd:schema>
```

```
49 </xs:complexType>
51 <xs:complexType name="ArrayOfLink">
52   <xs:sequence>
53     <xs:element minOccurs="0" maxOccurs="unbounded" name="Link" nillable="true"
54       type="s0:Link" />
55   </xs:sequence>
56 </xs:complexType>
57
58 <xs:complexType name="Link">
59   <xs:sequence>
60     <xs:element minOccurs="0" maxOccurs="1" name="SourceUID" type="xs:string" />
61     <xs:element minOccurs="0" maxOccurs="1" name="DestUID" type="xs:string" />
62     <xs:element minOccurs="0" maxOccurs="1" name="Guards" type="s0:ArrayOfGuard" />
63   </xs:sequence>
64 </xs:complexType>
65
66 <xs:complexType name="ArrayOfGuard">
67   <xs:sequence>
68     <xs:element minOccurs="0" maxOccurs="unbounded" name="Guard" nillable="true"
69       type="s0:Guard" />
70   </xs:sequence>
71 </xs:complexType>
72
73 <xs:complexType name="Guard">
74   <xs:sequence>
75     <xs:element minOccurs="0" maxOccurs="1" name="VarName" type="xs:string" />
76     <xs:element minOccurs="1" maxOccurs="1" name="IsNot" type="xs:boolean" />
77   </xs:sequence>
78 </xs:complexType>
79
80 <xs:complexType name="ArrayOfWebService">
81   <xs:sequence>
82     <xs:element minOccurs="0" maxOccurs="unbounded" name="WebService" nillable="true"
83       type="s0:WebService" />
84   </xs:sequence>
85 </xs:complexType>
86
87 <xs:complexType name="WebService">
88   <xs:sequence>
89     <xs:element minOccurs="0" maxOccurs="1" name="Name" type="xs:string" />
90     <xs:element minOccurs="0" maxOccurs="1" name="Uri" type="xs:string" />
91   </xs:sequence>
92 </xs:complexType>
93 </xsd:schema>
```

Listing C.1 – GOD : Schéma XSD

Annexe D

Configuration des services ADORE

D.1 Configuration Générale

Les informations de configuration des services ADORE sont définies dans un fichier `web.config` présent à la racine de l'application web à paramétrer

```
1 <?xml version="1.0" encoding="utf-8" ?>
  <configuration>
3   <appSettings>
5     <!-- General Setting -->
     <add key="trace" value="on/off" />
7
     <!-- Specific settings goes here -->
9
   </appSettings>
11 </configuration>
```

Listing D.1 – Fichier `web.config` usuel

trace : lorsque cette valeur est à `on`, des informations de *debug* sont affichées dans la console du serveur web lors de l'exécution :

```
[02/19/2007 18:37:31] Building new AdoreProxy Instance
[02/19/2007 18:37:31] Invoking AdoreProxy::Login
[02/19/2007 18:37:31] Invoking Specific AdoreProxy Builder
[02/19/2007 18:37:50] Building new AdoreProxy Instance
[02/19/2007 18:37:50] Invoking AdoreProxy::GetUsers
[02/19/2007 18:37:50] Invoking Specific AdoreProxy Builder
[02/19/2007 18:38:01] Building EngineUpdater Instance
[02/19/2007 18:38:01] Invoking EngineUpdater::GetAll
[02/19/2007 18:38:12] Building UsersManagement Instance
[02/19/2007 18:38:12] Invoking UsersManagement::GetAllUsers
```

D.2 ProxyAnnuary

```
1 <add key="proxy_annuary_token" value="Adore_token_value" />
  <add key="proxy_annuary_db" value="file_path" />
```

Listing D.2 – Paramétrage du service ProxyAnnuary

proxy_annuary_token: token du service.

proxy_annuary_db: chemin d'accès au fichier stockant la base de données du service.

D.3 EngineUpdater

```

2 <add key="engine_updater_token" value="Adore_token_value" />
  <add key="engine_updater_db" value="file_path" />
4 <add key="engine_updater_annuary_url"
  value="proxies_annuary_service_url" />

```

Listing D.3 – Paramétrage du service EngineUpdater

engine_updater_token: token du service.

engine_updater_db: chemin d'accès au fichier stockant la base de données du service.

engine_updater_annuary_url: url permettant d'accéder au service d'annuaire.

D.4 UsersManagement

```

2 <add key="users_management_emergency_mode" value="on/off" />
  <add key="users_management_token" value="Adore_token_value" />
  <add key="users_management_db" value="file_path" />
4 <add key="users_management_annuary_url"
  value="proxies_annuary_service_url" />
6 <add key="users_management_engine_updater_url"
  value="engine_updater_service_url" />

```

Listing D.4 – Paramétrage du service UsersManagement

users_management_emergency_mode: Lorsque cette valeur est à on, il est possible d'invoquer l'opération `__EmergencyAdd`, qui permet de créer un compte administrateur (login : admin, password : admin).

users_management_token: token du service.

users_management_db: chemin d'accès au fichier stockant la base de données du service.

users_management_annuary_url: url permettant d'accéder au service d'annuaire.

users_management_engine_updater_url: url permettant d'accéder au service de gestion des mise à jour.

D.5 ADORE Proxy

```

1 <add key="proxy_token" value="Adore_token_value" />
  <add key="adore_proxy_database" value="file_path" />

```

Listing D.5 – Paramétrage d'un proxy ADORE

proxy_token: token du service.

adore_proxy_database: chemin d'accès au fichier stockant la base de données du service.

D.6 Configuration type

Le listing suivant donne la configuration type de la plateforme ADORE.

```
2 <?xml version="1.0" encoding="utf-8" ?>
  <configuration>
    <appSettings>
4
      <!-- General Configuration -->
6      <add key="trace" value="on" />
8
      <!-- Proxies Annuary settings -->
10     <add key="proxy_annuary_token" value="let_us_go_!" />
12     <add key="proxy_annuary_db" value="databases/proxy_annuary.yap" />
14
      <!-- Engine Updater settings -->
16     <add key="engine_updater_token" value="let_us_go_!" />
18     <add key="engine_updater_db" value="databases/engine_updater.yap" />
20     <add key="engine_updater_annuary_url"
22     value="http://localhost:9000/admin/ProxiesAnnuary.asmx" />
24
      <!-- Users Management settings -->
26     <add key="users_management_emergency_mode" value="on" />
28     <add key="users_management_token" value="let_us_go_!" />
30     <add key="users_management_db" value="databases/users_management.yap" />
32     <add key="users_management_annuary_url"
    value="http://localhost:9000/admin/ProxiesAnnuary.asmx" />
    <add key="users_management_engine_updater_url"
    value="http://localhost:9000/admin/EngineUpdater.asmx" />
    <!-- Proxy sample settings -->
    <add key="proxy_token" value="let_us_go_!" />
    <add key="adore_proxy_database" value="databases/proxy_test.yap" />
  </appSettings>
</configuration>
```

Listing D.6 – Fichier /Adore/webservices/admin/web.config

Annexe E

Lancement de la plateforme

E.1 Système de fichiers

```
mosser@bartimeus:~$ cd Adore
mosser@bartimeus:~/Adore$ tree -L 1 -d -n
.
|-- bin          <-- Binaires Adore compilés
|-- doc         <-- Documentation (LATEX)
|-- java        <-- Programmes Java (Adore Manager)
|-- kml-samples <-- Exemples de documents Kml
|-- libs        <-- Librairies tierces .Net
|-- odml-samples <-- Exemples de documents Odml
|-- prolog      <-- Interpreteur pour la fusion d'orchestration OMSM
|-- src         <-- Sources .Net/C# du projet
\-- webservices <-- Services Web disponibles sur la plateforme
```

E.2 Démarrage d'ADORE

Pour lancer la plateforme ADORE, il suffit de lancer l'ensemble des Services Web présents. Des scripts de lancements permettent de simplifier cette opération.

```
mosser@bartimeus:~$ cd Adore/webservices
mosser@bartimeus:~/Adore/webservices$ ./start-web-server.sh
#####
## ADORE Project ##
#####

#####
## Compiling ... ##
#####

## [/home/mosser/Adore/webservices/admin/Makefile]
make: Rien à faire pour all.

#####
## Runing WebServer ##
#####
```

```
####
## server   : xsp2
## port     : 9000
## url      : http://localhost:9000/index.aspx
## WebApps  :
## --> /
## --> /admin
####
```

xsp2

```
Listening on port: 9000 (non-secure)
Listening on address: 0.0.0.0
Root directory: /home/mosser/Adore/webservices
Hit Return to stop the server.
```

E.3 Déclaration d'un nouveau Service Web

Pour ajouter un service web au sein de la plateforme, il est nécessaire de modifier le script de lancement pour qu'il tienne compte du nouvel arrivant.

Configuration UNIX : Il suffit de rajouter en fin du script `start-web-server.sh` une ligne du type

```
load_app relative/path/to/my/new/service
```

Configuration WINDOWS : L'ajout d'un service sur une plateforme Windows implique l'ajout dans le script de lancement des lignes suivantes :

```
set apps=%apps%,/relative/service/path:./relative/service/path
echo ## + /relative/service/path
```

E.4 Scripts de lancement

E.4.1 Plateforme UNIX

```

1  #!/bin/bash
2
3  BINARY="xsp2"
4  PORT=9000
5
6  if [ $# == 1 ]
7  then
8      PORT=$1
9  fi
10
11 APPS_OPT="/:."
12 APPS="##_-->_/_/"
13
14 function make_ws() {
15     cd $1
16     for f in *
17     do
18         if [ -d $f ]
19         then
20             make_ws $f

```

```

    fi
22 done
    if [ -e ./Makefile ]
24     then
        echo "##_[ `pwd` /Makefile]"
26     make

    fi
28     cd ..
}

30
function load_app() {
32     APPS_OPT="$APPS_OPT,/$1:./$1"
    APPS="$APPS\n##_-->/$1"
34 }

36 function compile() {
    echo ""
38     echo "#####"
    echo "##_Compiling_.._###"
40     echo "#####"
    echo ""
42     make_ws .
    cd webservices
44 }

46 function run() {
    echo ""
48     echo "#####"
    echo "##_Runing_WebServer_##"
50     echo "#####"
    echo ""
52     echo "#####"
    echo "##_server_:_$_BINARY"
54     echo "##_port_:_$_PORT"
    echo "##_url_:_http://localhost:$_PORT/index.aspx"
56     echo "##_WebApps_:"
    echo -e $APPS
58     echo "#####"
    echo ""
60     $_BINARY --port $_PORT --applications $APPS_OPT
}

62
function main() {
64     echo "#####"
    echo "##_ADORE_Project_##"
66     echo "#####"
    compile
68     run
}

70
## load_app declarations goes here ...
72 load_app admin
74 main

```

Listing E.1 – Script start-web-server.sh

E.4.2 Plateforme WINDOWS

Remarque : contrairement au script UNIX, le script de lancement ne compile pas les Services Web lors du lancement. Il faut utiliser pour cela le script `compile-web-services.bat`.

```

@echo off
2 echo #####
echo ## ADORE Project ##
4 echo #####

```

```
6 echo ## Runing WebServer ##
echo #####
echo --
8 echo ####
echo ## server : xsp2
10 echo ## port : 9000
echo ## url : http://localhost:9000/index.aspx
12 echo ####
echo --
14
16 rem loading Mono PATH
call C:\PROGRA~1\MONO-1~1.1\bin\setmonopath.bat
18
20 rem Setting Web Applications
echo --
echo ####
echo ## WebApps :
22
24 set apps=/.
echo ## + /
set apps=%apps%,/admin:./admin
26 echo ## + /admin
28
30 rem new web apps goes here ...
echo ####
echo --
32
xsp2 --port 9000 --applications "%apps%"
```

Listing E.2 – Script start-web-server.bat

Bibliographie

- [Bartoli et al., 2005] Bartoli, A., Jiminez-Peris, R., Kemme, B., Pautasso, C., Patarin, S., Wheeler, S., and Woodman, S. (2005). The ADAPT framework for adaptable and composable web services. *IEEE Distributed Systems Online*, 6(9).
- [Berger, 2001] Berger, L. (2001). *Mise en oeuvre des interactions en environnements distribués, compilés et fortement typés : le Modèle MICADO*. PhD thesis, Université de Nice - Sophia Antipolis.
- [Blay-Fornarino et al., 2004] Blay-Fornarino, M., Charfi, A., Emsellem, D., Pinna-Déry, A.-M., and Riveill, M. (2004). Software interaction. *Journal of Object Technology (ETH Zurich)*, 3(10) :161–180.
- [Chandran and Poduval, 2005] Chandran, P. and Poduval, A. (2005). Adding BPEL to the Enterprise Integration Mix. Technical report, ORACLE.
- [Charfi and Mezini, 2006] Charfi, A. and Mezini, M. (2006). AO4BPEL : An Aspect-Oriented Extension to BPEL. *World Wide Web Journal : Recent Advances on Web Services (special issue)*, to appear.
- [Cook, 2004] Cook, J. J. (2004). P# : a concurrent prolog for the .net framework. *Softw. Pract. Exper.*, 34(9) :815–845.
- [Courbis and Finkelstein, 2005] Courbis, C. and Finkelstein, A. (2005). Weaving aspects into web service orchestrations. In *ICWS*, pages 219–226. IEEE Computer Society.
- [Douence, 2004] Douence, R. (2004). A restricted definition of AOP. In Gybels, K., Hanenberg, S., Herrmann, S., and Wloka, J., editors, *European Interactive Workshop on Aspects in Software (EIWAS)*.
- [Douence et al., 2004] Douence, R., Fradet, P., and Südholt, M. (2004). Composition, reuse and interaction analysis of stateful aspects. In *3rd international conference on Aspect-oriented software development (AOSD '04)*, pages 141–150, Lancaster, UK. ACM Press.
- [Klein et al., 2007] Klein, J., Baudry, B., Barais, O., and Jackson, A. (2007). Introduction du test dans la modélisation par aspects.
- [MacKenzie et al., 2006] MacKenzie, M., Laskey, K., McCabe, F., Brown, P., and Metz, R. (2006). Reference Model for Service Oriented Architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS.
- [Nemo, 2006] Nemo, C. (2006). Vers la composition d’orchestrations de services. Master dissertation, DEA PLMT, Nice, France.
- [Nemo et al., 2006] Nemo, C., Blay-Fornarino, M., and Emsellem, D. (2006). Composition d’orchestrations de services. In *Atelier sur l’Evolution du Logiciel*, pages 53–60, Nimes. Salah Sadou.
- [White, 2006a] White, S. A. (2006a). *Business Process Modeling Notation (BPMN)*. IBM Corp.
- [White, 2006b] White, S. A. (2006b). Using BPMN to model BPEL process. Technical report, IBM Corp.

Table des matières

I	Contexte	2
1	Introduction	3
1.1	Un monde de Service ...	3
1.2	Un monde de «normes» ...	3
1.3	Deux approches complémentaires	4
1.4	Positionnement d'ADORE	4
2	Contrôler un Service Web	5
2.1	Exemple : Architecture Bancaire	5
2.1.1	Composition de processus métiers	5
2.1.2	Préoccupations orthogonales : Journalisation & <i>Debug</i>	7
2.1.3	<i>The all together</i> ...	10
2.2	Du contrôle sur les opérations de Services Web	11
2.2.1	Analyse des exemples précédents	11
2.2.2	Des Orchestrations de contrôle	11
2.3	Langage de description des Orchestrations	11
2.3.1	BPEL pour orchestrer	11
2.3.2	ODML pour contrôler	11
3	Une plateforme multi-tâches	13
3.1	Objectifs de la plateforme	13
3.1.1	Distribuer des orchestrations ?	13
3.1.2	Fusionner des orchestrations ?	13
3.1.3	Modifier dynamiquement des orchestrations ?	13
3.2	Choix technologiques & Utilisateurs	13
3.2.1	Technologies	13
3.2.2	Rôles utilisateurs	14
3.3	Une architecture à <i>proxies</i>	14
3.3.1	<i>Proxies</i> ADORE	14
3.3.2	<i>Proxies</i> Utilisateurs	16
3.3.3	Ajout d'un service sur la plateforme	16
3.4	Services techniques	16
3.4.1	<i>Proxies</i> Annuaire	16
3.4.2	Users Management	17
3.4.3	Engine Updater	17
3.4.4	ADORE <i>proxies</i>	18
3.4.5	Sauvegarde et restauration des services	18
3.4.6	Communication inter-service	18
3.4.7	Configuration des services techniques	19
3.5	Cycle de vie des Orchestrations de Contrôles	19
3.5.1	Chargement d'une orchestration	19
3.5.2	Déchargement d'une orchestration	19

3.5.3	Exemple <i>pas à pas</i>	19
3.6	Administration & Utilisation	19
II Un développement guidé par les modèles		22
4	Une plateforme multi-modèles	23
4.1	Les Objectifs	23
4.2	Différents paradigmes	23
4.3	Différents modèles	23
4.4	Différentes transformations	25
5	ODML : Des aspects pour contrôler	26
5.1	Pivots de composition : une approche AOP	26
5.1.1	Des aspects sur les Services Web ?	26
5.1.2	Définition des aspects sur les Services Web	26
5.1.3	Conflits de tissage	27
5.1.4	Processus de tissage	27
5.2	Du document à l'objet	27
5.2.1	ODML, un langage d'aspects sur Services Web	27
5.2.2	ODML, un modèle objet	28
5.3	Des Objets au cœur de la plateforme	30
6	OMSM : De la logique pour fusionner	31
6.1	Programmation Logique : un paradigme adéquat	31
6.1.1	Indépendant de la description des Orchestrations	31
6.1.2	Puissance de l'interpréteur logique	31
6.2	OMSM en détail	31
6.2.1	Définition du modèle	31
6.2.2	D'ODML à OMSM, et réciproquement	32
6.2.3	Processus de fusion	32
6.3	Une fusion par l'exemple	32
6.3.1	Orchestration O_1 : Certification électronique	32
6.3.2	Orchestration O_2 : Fraude Bancaire	33
6.3.3	Calcul de $O_3 \equiv merge(O_1, O_2)$	34
6.4	Gestion des conflits	35
6.4.1	Définition	35
6.4.2	Résolution des conflits	35
6.4.3	Dans l'exemple précédent	35
7	GOD : Des objets pour afficher	37
7.1	Visualisation des Orchestrations	37
7.2	Un nouveau modèle ?	37
7.2.1	Hétérogénéité des langages	37
7.2.2	Vers un modèle supportant l'hétérogénéité	38
7.3	GOD, un raffinement d'ODML	38
7.3.1	Aplatissement de l'héritage	38
7.3.2	Disparition des activités composites	38
7.3.3	Transformation de Modèle	38
7.3.4	Modélisation	39
7.4	Affichage des Orchestrations	39

8	DEMON : Des <i>threads</i> pour exécuter	40
8.1	Une évaluation distribuée	40
8.2	<i>Distributed Execution Model for Orchestration</i>	40
8.2.1	Un nouveau modèle ?	40
8.2.2	Ordonancement événementiel .NET	40
8.2.3	DEMON en détail	41
8.3	Validation par l'exemple	41
8.3.1	Une Orchestration ODML	41
8.3.2	Instanciation DEMON	42
8.3.3	Exécution <i>pas à pas</i>	42
III	Conclusion	43
9	Démonstration : «<i>Proof Of Concept</i>»	44
9.1	Objectifs de ce chapitre	44
9.2	SÉDUITE – EPUB	44
9.2.1	Implantation de l'application	44
9.2.2	Vocabulaire & Fonctionnement de la plateforme SEDUITE	44
9.2.3	Ajout d'un partenaire dans le fournisseur	45
9.3	Étude de cas & Exemple d'utilisation	46
9.3.1	Scénario	46
9.3.2	Orchestration O_b : Ajout des <i>Anniversaires du Jour</i>	46
9.3.3	Orchestration O_s : Ajout de la <i>Vitrine des Projets</i>	47
9.3.4	Orchestration $O_{bs} \equiv merge(O_b, O_s)$	48
10	Conclusion & Évolutions futures	50
10.1	Un projet exploratoire	50
10.2	Une plateforme <i>quasi</i> -opérationnelle	50
10.3	Évolutions envisagées	51
10.3.1	Refonte de la plateforme SÉDUITE	51
10.3.2	Intégration du projet POLLUX	51
10.3.3	Commutativité et Associativité des contrôles ADORE	51
10.3.4	Adaptation à la manipulation de composants d'IHM	51
IV	Annexes	52
A	Une orchestration BPEL	53
B	ODML : Syntaxe du langage	56
B.1	Orchestration de contrôle	56
B.2	Activités de contrôle	57
B.2.1	<i>proceed</i>	57
B.2.2	<i>delegate</i>	57
B.3	Activité Atomiques	57
B.3.1	<i>assign</i>	57
B.3.2	<i>empty</i>	58
B.3.3	<i>invoke</i>	58
B.3.4	<i>reply</i>	58
B.3.5	<i>throw</i>	59
B.4	Activités Composites	59
B.4.1	<i>sequence</i>	59
B.4.2	<i>flow</i>	59

B.5	Activités Conditionelles	59
B.5.1	Définition des conditions	60
B.5.2	if/then/else	60
C	Orchestration GOD : Schéma XSD	61
D	Configuration des services ADORE	63
D.1	Configuration Générale	63
D.2	ProxyAnnuary	63
D.3	EngineUpdater	64
D.4	UsersManagement	64
D.5	ADORE Proxy	64
D.6	Configuration type	65
E	Lancement de la plateforme	66
E.1	Système de fichiers	66
E.2	Démarrage d'ADORE	66
E.3	Déclaration d'un nouveau Service Web	67
E.4	Scripts de lancement	67
E.4.1	Plateforme UNIX	67
E.4.2	Plateforme WINDOWS	68

Table des figures

2.1	O_1 : Certification électronique	6
2.2	O_2 : Fraude Bancaire	6
2.3	$O_3 \equiv merge(O_1, O_2)$	7
2.4	O_4 : Journalisation des Opérations	8
2.5	O_5 : <i>Debug</i> du transfert de fond	9
2.6	$O_{all} \equiv merge(O_1, O_2, O_4, O_5)$	10
3.1	Architecture Générale de la plateforme ADORE	15
3.2	Services techniques de la plateforme ADORE	17
3.3	Interface Web d'ADORE	20
3.4	Interface MONO d'un Service Web	20
3.5	Client riche ADORE (Projet SI2)	21
4.1	ADORE : Une plateforme <i>multi-modèles</i>	24
4.2	Modèles mis en jeu au sein d'ADORE	24
5.1	ODML : Diagramme de classes général	29
5.2	ODML : Diagramme de classe des Activités	29
7.1	GOD : Diagramme de classes UML	39
9.1	Architecture SÉDUITE déployée sur POLYTECH'NICE	45
9.2	O_b : Ajout du service d'anniversaire au sein de SÉDUITE	46
9.3	O_s : Ajout du service vitrine au sein de SÉDUITE	47
9.4	$O_{bs} \equiv merge(O_b, O_s)$	49
A.1	<i>ActiveBpel</i> : Loan Approval Process	53

Listings

5.1	Définition d'un <i>before</i> en ODML	27
5.2	Définition d'un <i>around</i> en ODML	28
5.3	Définition d'un <i>after</i> en ODML	28
6.1	ODML : description de O_1	33
6.2	ODML : description de O_2	33
8.1	Orchestration ODML pour exécution DEMON	41
9.1	O_b : Ajout des anniversaires dans le système SÉDUITE	46
9.2	O_s : Ajout de la vitrine des projets dans le système SÉDUITE	48
9.3	$O_{bs} \equiv merge(O_b, O_s)$	49
A.1	<i>ActiveBpel</i> Loan Approval pseudo-code	53
A.2	<i>ActiveBpel</i> Loan Approval BPEL code	54
B.1	Définition d'une Orchestration ODML	56
B.2	ODML : Syntaxe du <i>proceed</i>	57
B.3	ODML : Syntaxe du <i>delegate</i>	57
B.4	ODML : Syntaxe du <i>assign</i>	57
B.5	ODML : Syntaxe du <i>empty</i>	58
B.6	ODML : Syntaxe du <i>invoke</i>	58
B.7	ODML : Syntaxe du <i>reply</i>	58
B.8	ODML : Syntaxe du <i>throw</i>	59
B.9	ODML : Syntaxe de la <i>sequence</i>	59
B.10	ODML : Syntaxe du <i>flow</i>	59
B.11	ODML : Syntaxe du <i>is/then/else</i>	60
C.1	GOD : Schéma XSD	61
D.1	Fichier <i>web.config</i> usuel	63
D.2	Paramétrage du service <i>ProxyAnnuary</i>	63
D.3	Paramétrage du service <i>EngineUpdater</i>	64
D.4	Paramétrage du service <i>UsersManagement</i>	64
D.5	Paramétrage d'un proxy ADORE	64
D.6	Fichier <i>/Adore/webservices/admin/web.config</i>	65
E.1	Script <i>start-web-server.sh</i>	67
E.2	Script <i>start-web-server.bat</i>	68