Des Orchestrations de Services Web aux Aspects

Cédric Joffroy, Sébastien Mosser, Mireille Blay-Fornarino, Clémentine Nemo

Laboratoire I3S (CNRS - UNSA), Bâtiment Polytech'Sophia – SI
930 route des Colles – B.P. 145, F-06903 Sophia Antipolis Cedex
{joffroy,mosser,blay,nemo}@polytech.unice.fr

RÉSUMÉ. *Les Architectures Orientées Services (SOA) visent à supporter les évolutions rapides des applications en fonction des services attendus et disponibles. Un des paramètres essentiels de cette flexibilité est le couplage faible entre les services. Pour atteindre cet objectif, les compositions entre les services sont explicitées dans des fonctions d'orchestrations 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 à ces 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. Est-il dans ce cadre possible d'envisager les orchestrations elles-même comme des aspects explicitant les interactions sur une application constituée des services de base? En quoi les orchestrations présentent-elles de manière intrinsèque des caractéristiques liées à la programmation par aspects? En quoi les interactions entre orchestrations sont-elles similaires aux interactions entre aspects? Cet article amorce la discussion de ces différents points et envisage une approche étendue des orchestrations pour exprimer des aspects afin de supporter une évolution dynamique des compositions de services.*

ABSTRACT. .

MOTS-CLÉS : *Orchestrations, Aspects, Services Web, Composition, Interactions, Fusion, SOA*

KEYWORDS: *Orchestrations, Aspects, Web Services, Composition, Interactions, Merge, SOA*

1. Orchestrations et séparation des préoccupations

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. 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). Sans les orchestrations, la composition de services serait définie par 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 des services requis. Ce qui ne permettrait pas une programmation incrémentale non anticipée.

Au regard de la définition de l'AOP[1] par R. Douence dans (Douence, 2004), nous constatons une analogie entre aspects et orchestrations de services. 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 *et al.,* 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 des gros projets.

Les travaux actuels mêlant aspects et Services Web visent à intégrer les propriétés non fonctionelles au niveau des orchestrations et des moteurs d'exécution (Charfi *et al.,* 2006, Courbis *et al.,* 2005) 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. Contrairement à cela, nous proposons 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. Notre approche est indépendante du moteur d'exécution et ne force pas la modification des codes clients. Nous ne prenons pas en compte à ce jour la mise en place de contrôles impliquant une modification des enveloppes SOAP. Ceci restreint notre approche aux propriétés comportementales que les aspects peuvent ajouter aux Services Web, mais n'induit aucune modification sur les clients de ces services.

Notre contribution, dans un premier temps, justifie notre appréhension des orchestrations comme des aspects sur un exemple en soulignant les besoins de tissage entre orchestrations. Nous en déduisons une approche aspect de la définition des orchestrations, à la fois par l'introduction de la notion d'activités pivots et par la recherche de conflits lorsqu'apparaissent des interactions entre orchestrations. Pour finir, nous

1. *"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."*

exposons un cycle de vie possible pour ces orchestrations et concluons sur les perspectives offertes par cette approche.

2. Besoin de tissage dans les orchestrations

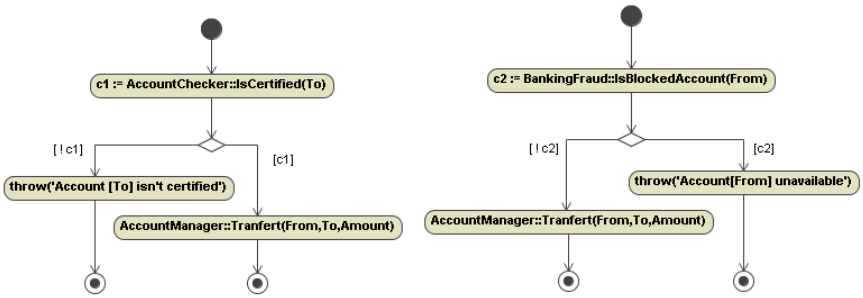
Les orchestrations permettent de faire collaborer de manière efficace plusieurs Services Web, en définissant des Services Composites par assemblage de services. Ce formalisme est adopté par les industriels, à travers différents langages (BPEL, SCUFL, ...) et différentes plateformes (.NET, WEBSPHERE, AXIS, ...) Au sein d'une entreprise, elles sont un moyen de respecter les principes d'une architecture SOA (Service Oriented Architecture) (MacKenzie *et al.,* 2006) en autorisant une expression explicite des communications entre les Services Web.

Néanmoins dès que la taille des *«workflows»* ainsi exprimés devient importante, des problèmes de compatibilité et de recouvrement se posent. La gouvernance est alors indispensable pour régler les conflits potentiels et pour diriger les développements. Les aspects sur Services Web peuvent alors jouer leurs rôles en se présentant comme naturellement composables et répondant au besoin de découplages des services.

Nous illustrons notre approche par l'exemple suivant. Nous nous positionnons au sein d'un système d'information bancaire, sur lequel nous disposons de trois services métiers suivants : (i) `AccountManager` pour gérer les comptes bancaires des clients de l'établissement, (ii) `BankingFraud` pour obtenir la liste de comptes soupçonnés de fraude et (iii) `AccountChecker` pour administrer une liste de comptes certifiés pour les transactions électroniques. Suite à des évolutions de la législation française, le système d'information doit évoluer afin de respecter la Loi.

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 utilise une orchestration afin de combiner les services `AccountManager` et `AccountChecker`. Il écrit donc l'orchestration O_1, décrite dans le formalisme des Diagrammes d'Activités UML en Figure 1(a). L'orchestration ainsi définie est ensuite chargée dans un moteur d'exécution BPEL, et est publiée comme un nouveau service web `AccountManagerWithCertification`. Il faut maintenant informer tous les services développant des logiciels adressant l'ancien service pour qu'ils adressent maintenant le nouveau service publié.

Selon la Loi, aucune opération bancaire ne peut être effectuée si le compte source est soupçonné de fraude bancaire. Un second programmeur P_2 décrit une nouvelle orchestration combinant les services `AccountManager` et `BankingFraud`. L'orchestration O_2 ainsi produite (Figure 1(b)) est publiée dans le moteur d'exécution BPEL sous le nom `AccoutManagerWithFraud`, et les consommateurs du service `AccountManager` doivent être informés de la publication de ce nouveau service.



(a) (b)

Figure 1. *(a) O_1 : Certification électronique. (b) O_2 : Fraude Bancaire.*

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 différents pour gérer le même but, chacun doté de sa sémantique : (i) `AccountManager`, historiquement présent dans le système d'information, (ii) `AccountManagerWithCertification` qui respecte la directive sur la certification électronique et (iii) `AccountManagerWithFraud` qui respecte la législation en vigueur sur les 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. La combinaison des deux orchestrations ci–dessus définies place intuitivement l'invocation `AcountManager::Transfert` comme un pivot de composition, car les deux orchestrations le contrôlent. Lorsque les conditions *! c1 && c2* sont réunies, le chef de produit fait le choix de lever l'exception *«Fraude Bancaire»*. Une approche par aspect nous paraît ici appropriée pour automatiser ce type de composition tout en préservant une expression séparée des préoccupations. La nouvelle orchestration est publiée dans le moteur d'exécution BPEL, sous le nom `AccountManagerLawCompliant`.

A présent, afin d'obtenir des statistiques sur l'utilisation du système d'information, P_L demande à son équipe de développement de journaliser durant trente jours certaines opérations effectuées sur le système en utilisant un nouveau Service Web `Historic` qui permet la journalisation d'opérations. Un développeur crée une nouvelle orchestration couplant `AccountManagerLawCompliant` et `Historic`, comme décrit dans la figure 3. Nous nous focalisons sur cette seule orchestration dans cet article, mais le développeur fait de même pour tous les services critiques mis en jeux dans le système d'information. La tâche est répétitive et cause d'erreurs, et implique un arrêt du système nécessaire au remplacement des orchestrations pré–existante par les nouvelles orchestrations journalisées. Une approche par aspects permettrait de localiser l'introduction des codes et de l'appliquer de manière transversale à plusieurs

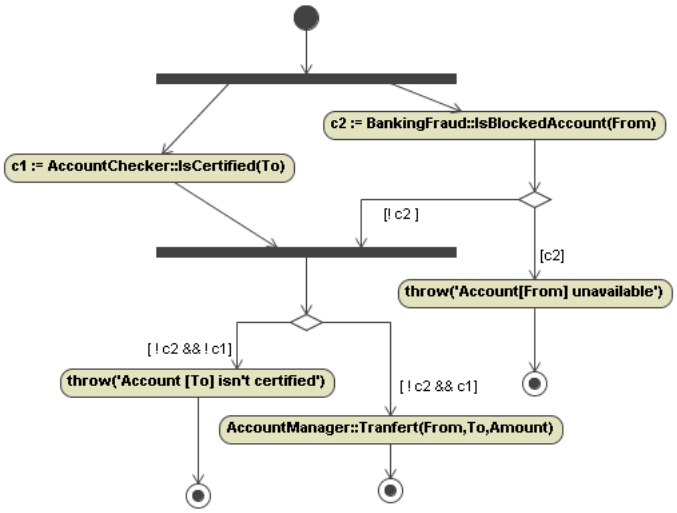


Figure 2. $O_3 \equiv merge(O_1, O_2)$

services et inversement de retirer ces codes trente jours plus tard de manière automatique.

3. Des aspects pour Services Web

Dans le formalisme BPEL de description des orchestrations, on ne peut que définir un nouveau service par assemblage de services existants. Or, nous nous intéressons à des orchestrations redéfinissant un comportement sans définir un nouveau service. Dans l'exemple précédent, nous constatons que les compositions sont construites autour d'une invocation de service pivot qu'il est possible de reconnaître mais qui suppose une approbation du programmeur. A la manière des langages d'aspects nous proposons donc de les considérer comme des points d'interceptions que le développeur explicite.

Nous décidons d'étendre le langage BPEL pour intégrer la notion d'activités faisant référence au comportement original (`proceed`) et l'expression des sélecteurs (point d'intercession, `target` & `overload`). Ce langage se nomme ODML (*Orchestration Description Markup Language*). La figure 4 visualise le code (partiel, sans les entêtes de définition nécessaire aux Services Web) ODML correspondant à la définition de l'aspect de journalisation O_4. La figure 5 visualise le code (partiel) ODML pour l'expression du contrôle O_1 sur l'opération `Transfert`.

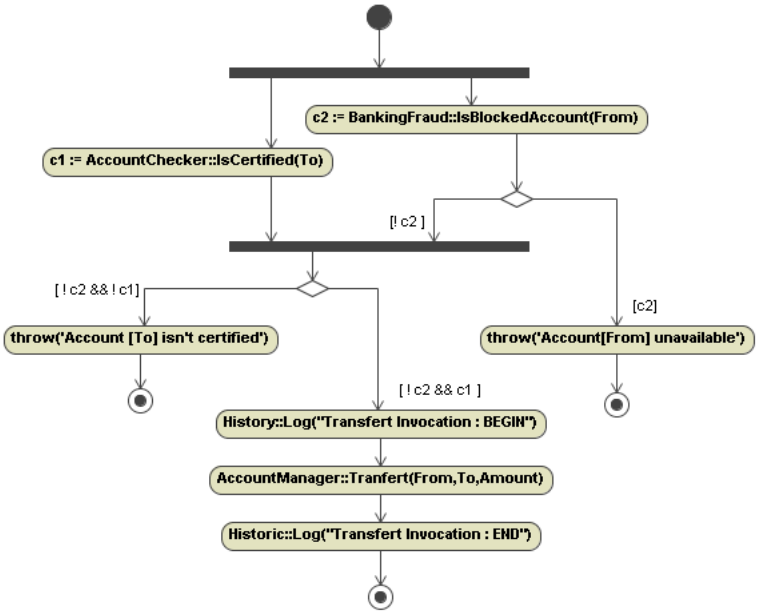


Figure 3. O_4 *: Journalisation des opérations et contrôles de transferts*

```
<ControlOrchestration name="OperationTrace"
            target="AccountManager" overload="*">
  <variable name="before" type="xml:string" value="..." />
  <variable name="after"  type="xml:string" value="..." />
  <sequence>
    <invoke webservice="History" operation="Log">
      <parameters> <parameter variable="before"/> </parameters>
    </invoke>
    <proceed/>
    <invoke webservice="History" operation="Log" >
      <parameters> <parameter variable="after" /> </parameters>
    </invoke>
  </sequence>
</ControlOrchestration>
```

Figure 4. ODML *: description partielle de l'aspect de journalisation* O_4

```
<ControlOrchestration name="CertifiedTransfert"
          target="AccountManager" overload="Transfert">
  <sequence>
    <invoke webservice="AccountChecker" operation="isCertified"
            output="c1">
      <parameters> <parameter variable="To" /> </parameters>
    </invoke>
    <if condition="c1">
      <then>
        <proceed>
          <parameters>
            <parameter variable="From" />
            <parameter variable="To" />
            <parameter variable="Amount" />
          </parameters>
        </proceed>
      </then>
      <else>
        <throw message="Account [To] isn't certified">
          <parameters><parameter variable="To" /></parameters>
        </throw>
      </else>
    </if>
  </sequence>
</ControlOrchestration>
```

Figure 5. ODML *: Description partielle de l'aspect* O_1

Les approches usuelles par *before*, *after* ou *around* qui n'interdisent pas le comportement initial sont intrinsèquement prises en charge par le langage d'orchestration qui permet nativement ce type de description. Pour expliciter le point de contrôle, nous avons introduit deux nouvelles activités dites activités pivots qui permettent d'identifier les invocations au message contrôlé : *proceed* et *delegate*. La première est conforme au *proceed* des langages d'aspects tel qu'ASPECTJ, la seconde correspond à une redéfinition du message initial selon une approche similaire à (Klein *et al.*, 2007) avec la surcharge de la méthode *proceed*.

Nous définissons un aspect de Service Web par un sélecteur s et un greffon G (*advice*). Un sélecteur s identifie la ou les opération(s) contrôlée(s) pour un service. Un greffon G se définit par un tuple $(A, P, Pred, Cond)$ 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 gardes sur les activités définies dans A (cf. Figure 6 et 7). Nous avons pris en charge dans A, outre les activités pivots, l'*affectation de variable*, le *retour* de résultat, l'*invocation* d'une opération distante et la *levée d'exception* qui correspondent aux instructions élémentaires BPEL (`assign`,

`reply`, `invoke`, `throw`). Une activité $a \in A$ est définie par un tuple $(type, in, out)$, où $type$ est le type de l'activité, in l'ensemble des variables d'entrées et out l'ensemble des variables de sorties. Les figures 6 et 7 montrent la modélisation formelle des greffons correspondant aux orchestrations O_1 et O_2.

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 (cf. Figure 8).

$$
\begin{aligned}
G_1 &= (\{a_1, a_2, a_3\}, \{a_3\}, \{(a_1 < a_2), (a_1 < a_3)\}, \{(c_1, a_3), (\neg c_1, a_2)\}) \\
a_1 &= ((\boldsymbol{invoke}, AccountChecker, isCertified), \{To\}, \{c_1\}) \\
a_2 &= (\boldsymbol{throw}, \{\text{"Account [To] isn't certified"}, To\}, \emptyset) \\
a_3 &= (\boldsymbol{proceed}, \{From, To, Amount\}, \emptyset)
\end{aligned}
$$

Figure 6. *Greffon correspondant à l'orchestration* O_1

$$
\begin{aligned}
G_2 &= (\{a_5, a_6, a_7\}, \{a_3\}, \{(a_5 < a_6), (a_5 < a_7)\}, \{(c_2, a_6), (\neg c_2, a_7)\}) \\
a_5 &= ((\boldsymbol{invoke}, BankingFraud, isBlockedAccount), \{From\}, \{c_2\}) \\
a_6 &= (\boldsymbol{throw}, \{\text{"Account [From] unavailable"}, From\}, \emptyset) \\
a_7 &= (\boldsymbol{proceed}, \{From, To, Amount\}, \emptyset)
\end{aligned}
$$

Figure 7. *Greffon correspondant à l'orchestration* O_2

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. La figure 8, résultat de la composition de O_1 et O_2 expose un conflit `reply/reply` lorsque les conditions $!c_1 \&\& c_2$ sont réunies.

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* et *multipleDelegate*. Le conflit levé dans la figure 8 est instancié par le tuple $([!c1 \&\& c2], G_3, multipleThrowActivities, [i_2, i_6])$.

$$
\begin{aligned}
G_3 &= (\{a_1, a_2, a_5, a_6, a_8\}\}, \{a_8\}, \{(a_1 < a_2), (a_1 < a_8), (a_5 < a_6), (a_5 < a_8)\}, \\
&\quad \{(c_1, a_8), (\neg c_1, a_2), (c_2, a_6), (\neg c_2, a_8)\}) \\
a_1 &= ((\boldsymbol{invoke}, AccountChecker, isCertified), \{To\}, \{c_1\}) \\
a_2 &= (\boldsymbol{throw}, \{\text{"Account [To] isn't certified"}, To\}, \emptyset) \\
a_8 &= (\boldsymbol{proceed}, \{From, To, Amount\}, \emptyset) \\
a_5 &= ((\boldsymbol{invoke}, BankingFraud, isBlockedAccount), \{From\}, \{c_2\}) \\
a_6 &= (\boldsymbol{throw}, \{\text{"Account [From] unavailable"}, From\}, \emptyset)
\end{aligned}
$$

Figure 8. *Greffon conflictuel après fusion des greffons G_1 et G_2*

4. Cycle de vie des aspects pour Services Web

Écrire une orchestration qui met en jeu différentes préoccupations reste un exercice difficile. C'est pourquoi nous proposons une approche par aspects basés sur le formalisme des orchestrations pour faire évoluer une application à base de Services Web. Le caractère dynamique de ces applications nous oblige à proposer un processus de composition qui supporte la définition collaborative d'orchestrations ainsi que la détection de conflits. Pour ce faire, on utilise un serveur d'aspects qui intègre une base de connaissances (Douence *et al.,* 2004) alimentée par les développeurs et qui permet la résolution de conflits dus aux interactions entre aspects. Ce serveur a une connaissance globale des orchestrations utilisées et est adressé par ceux qui définissent les aspects. La Figure 9 décrit le diagramme de séquence mis en jeu lors de la pose d'une orchestration.

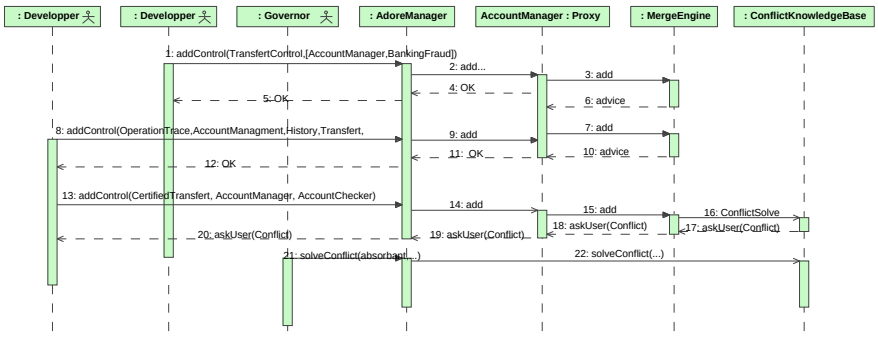


Figure 9. *Scénario partiel de pose d'aspects et détection de conflicts*

La démarche globale pour définir et poser des aspects suit le processus suivant :

1) L'application est constituée d'un ensemble de services en exploitation : $E = s_1..s_n$. Ces services sont orchestrés par un ensemble d'aspects qui ont été composés avec succès. A chaque opération j d'un service s_i, nous associons un greffon résultant des compositions : g_{ij}.

2) La pose d'un nouvel aspect a est demandée au serveur d'aspect sur une opération j d'un service s_i.

3) La composition entre ce nouvel aspect a et g_{ij} est alors tentée par composition des greffons concernés par le sélecteur défini dans a. Elle peut conduire à la détection de conflits que le programmeur devra résoudre pour valider la pose de l'aspect. Lorsque tous les conflits ont été résolus, un nouveau greffon est calculé. Il sera interprété lors de la réception d'un message correspondant au sélecteur par le service s_i. Ce greffon peut être visualisé comme une orchestration définissant l'opération j sur le service s_i. Nous disposons d'un client riche capable d'afficher sous forme de graphe les greffons obtenus (Figure 10).

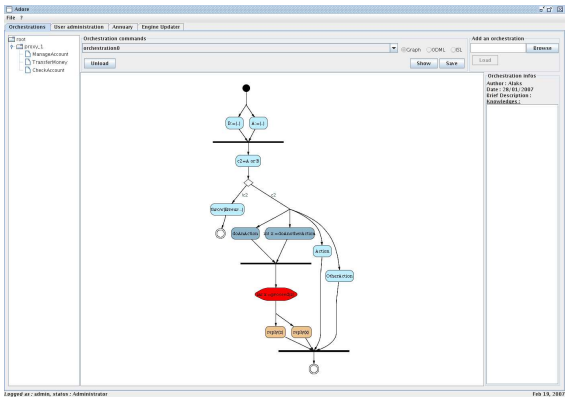


Figure 10. *Client riche d'administration* ADORE

4) Lorsqu'un conflit est détecté, la solution du conflit est recherchée dans la base de connaissances. La base de connaissances est composée de règles correspondant à des connaissances générales de l'application ou d'ajouts des programmeurs en réponses à des conflits. Une règle est composée d'un contexte identifiant les conditions d'application de la règle et des activités en conflits. L'application d'une règle entraîne une modification du greffon en cours de calcul.

L'absence de condition stipule que la première levée d'exception est absorbante vis-à-vis de la seconde quelles que soient les conditions. Cette déclaration équivaut à ajouter dans la base de connaissances la règle suivante :

```
conflictSolve(Cond,G,multipleThrowActivities,
              [throw('Account [To] isn't certified'),
               throw('Account [From] unavailable')], NewG) :-
  removeActivity(Cond,throw('Account [To] isn't certified'),G,NewG).
```

Lors de la suppression d'un aspect, le serveur d'aspect va recalculer le greffon correspondant aux sélecteurs concernés en fusionnant les contrôles restant pour ces sélecteurs.

Un langage dédié est défini permettant dans les cas récurrents de simplement décrire la résolution d'un conflit. Dans notre exemple, le chef de produit précisera :

```
absorbent(_,throw('Account[From] unavailable'],
          throw('Account [To] isn't certified')).
```

5. Conclusion et Perspectives

Les orchestrations se présentent comme un formalisme pour composer des Services Web. Lorsque plusieurs orchestrations traitent de l'introduction de nouveaux comportements autour des mêmes invocations de services, les mécanismes propres à la programmation par aspects se révèlent nécessaires pour tisser les orchestrations entre elles. Nous défendons donc l'idée qu'un langage d'aspect pour les Services Web doit s'appuyer sur les langages d'orchestration qui par leur nature même définissent l'introduction de nouveaux comportements et permettent une approche incrémentale. L'intérêt de notre approche est d'appliquer dynamiquement les modifications de comportements. Pour cela nous avons mis en place l'explicitation d'un pivot au sein des orchestrations, ainsi que des mécanismes de tissage, détection et résolution de conflits à l'aide d'une base de connaissances.

Lorsque ces modifications sont définitives, nous envisageons de transformer en orchestrations l'ensemble des greffons associés aux services mis en jeux par le système d'informations. Dans ce contexte, notre travail pourra aider à la définition et à la découverte de conflits au fur et à mesure des évolutions d'un système ; et ceci dans le cas d'applications de grandes tailles définies et maintenues par des programmations séparées dans le temps et l'espace. La transformation se fait de la manière suivante : le message initial BPEL `receive` correspond au sélecteur et les activités *proceed* sont remplacées par une invocation aux services contrôlés. L'intégration de cette nouvelle orchestration avec celles déjà existantes dans le système d'informations est un travail en cours (Nemo *et al.*, 2007). Cependant les langages d'orchestrations comme BPEL supportent plus de structures de contrôles que celles que nous avons pris en charge pour l'instant, notamment les boucles, les attentes (en durée de temps), les récupérations d'erreurs. La prise en compte de ces structures dans la définition des aspects nécessite de définir un processus de fusion associé. Ce travail est lui aussi en cours.

A l'heure actuelle, nous disposons d'une implémentation basée sur une architecture à *proxies* mise en place avec le framework DOTNET, connectée à un moteur de composition d'aspects implémenté en PROLOG. Une présentation plus détaillée de ce travail est disponible sur le site wiki du projet, à l'adresse `http://rainbow.polytech.unice.fr/adore`. On trouve dans (Joffroy *et al.*, 2007) les détails de la modélisation et de l'implémentation dont nous disposons.

6. Bibliographie

Bartoli A., Jiminez-Peris R., Kemme B., Pautasso C., Patarin S., Wheater S., Woodman S., « The ADAPT Framework for Adaptable and Composable Web Services », *IEEE Distributed Systems Online*, September, 2005.

Blay-Fornarino M., Charfi A., Emsellem D., Pinna-Déry A.-M., Riveill M., « Software interaction », *Journal of Object Technology (ETH Zurich)*, vol. 3, n° 10, p. 161-180, 2004.

Chandran P., Poduval A., Adding BPEL to the Enterprise Integration Mix, Technical report, ORACLE, November, 2005.

Charfi A., Mezini M., « AO4BPEL : An Aspect-Oriented Extension to BPEL », *World Wide Web Journal : Recent Advances on Web Services (special issue), to appear*, 2006.

Courbis C., Finkelstein A., « Weaving Aspects into Web Service Orchestrations. », *ICWS*, IEEE Computer Society, p. 219-226, 2005.

Douence R., « A Restricted Definition of AOP », *in* , K. Gybels, , S. Hanenberg, , S. Herrmann, , J. Wloka (eds), *European Interactive Workshop on Aspects in Software (EIWAS)*, September, 2004.

Douence R., Fradet P., Südholt M., « Composition, reuse and interaction analysis of stateful aspects », *3rd international conference on Aspect-oriented software development (AOSD '04)*, ACM Press, Lancaster, UK, p. 141-150, 2004.

Joffroy C., Mosser S., Blay-Fornarino M., Aspects & Distributed ORchEstrations, Technical Report n° I3S/RR-2007-09-FR, I3S, Sophia Antipolis, France, March, 2007.

Klein J., Baudry B., Barais O., Jackson A., « Introduction du test dans la modédlisation par aspects », 2007.

MacKenzie M., Laskey K., McCabe F., Brown P., Metz R., Reference Model for Service Oriented Architecture 1.0, Technical Report n° wd-soa-rm-cd1, OASIS, February, 2006.

Nemo C., Blay-Fornarino M., Kniesel G., Riveill M., « SEMANTIC ORCHESTRATIONS MERGING - Towards Composition of Overlapping Orchestrations », *in* , J. Filipe (ed.), *9th International Conference on Enterprise Information Systems (ICEIS'2007)*, Funchal, Madeira, June, 2007.

White S. A., *Business Process Modeling Notation (BPMN)*, IBM Corp. May, 2006a.

White S. A., Using BPMN to model BPEL process, Technical report, IBM Corp, 2006b.