

Réflexions autour de la construction dirigée par les modèles d'un atelier de composition d'orchestrations

Sébastien Mosser, Mireille Blay-Fornarino

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

Résumé. Il est aujourd'hui courant d'utiliser une approche «*orientée service*» pour définir des applications complexes. Dans ce cadre, les services constituent les unités de base, assemblés par des mécanismes de plus haut niveau comme les orchestrations. Nous présentons ici un atelier logiciel construit dans une démarche *dirigée par les modèles* permettant la composition d'orchestrations de services, et discutons les choix effectués ainsi que les difficultés rencontrées lors de sa mise en œuvre.

1 Introduction

L'ingénierie dirigée par les modèles (IDM) appliquée aux architectures logicielles vise à répondre aux besoins d'intégration des applications et d'adaptation à l'évolution des technologies (Richard Soley, 2000). Avec pour objectif la rapidité de développement, nous avons appliqué cette démarche au développement d'un atelier de composition d'orchestrations.

Dans le domaine des architectures orientées services SOA (Papazoglou et Heuvel, 2006), les fonctionnalités élémentaires des applications sont identifiées par des *services* et les processus métiers inhérents à l'entreprise sont construits par des assemblages de services. Une des technologies de mise en œuvre usuelle est l'utilisation de Web Services pour implémenter les unités de base et la définition d'*orchestrations* pour les assembler. Une orchestration se présente donc schématiquement comme un ensemble d'activités décrivant le flot des appels aux différents services qu'elle assemble. Un langage «standard» de définition des orchestrations est BPEL (et ses dérivés) (OASIS, 2007), où la définition d'un assemblage se présente sous la forme d'un document XML difficile à manipuler par un humain. Des environnements de programmation des orchestrations ont donc été définis (ECLIPSE BPEL DESIGNER, NETBEANS SOA SUITE, ...), et permettent une manipulation graphique des assemblages.

En complément aux travaux sur les orchestrations, visant à assurer la correction des codes (Pourraz, 2007) ou à améliorer l'abstraction de l'expressivité (White, 2006), nous nous intéressons à valider et optimiser la composition d'orchestrations. En effet la composition et l'évolution des orchestrations ne sont pas réduites à l'assemblage ou à la modification des enchaînements des activités. Elles consistent également à éliminer ou regrouper les appels à

des services économiquement coûteux ou lents afin de gagner en rapidité ou en coût. Cette logique d'optimisation a été mise en œuvre à travers un algorithme de composition inductif (Mosser et al., 2008a; Nemo et al., 2007b) où la représentation des relations entre les activités est indépendante des opérateurs du langage d'orchestration utilisé, et où l'implémentation en Prolog s'est révélée adaptée au problème grâce aux mécanismes d'unification de variables et à la recherche de solutions multiples...

Afin que l'algorithme puisse être utilisé par un concepteur d'orchestrations (non expert en langage logique), un atelier (DRADORE) de composition d'orchestrations exprimées en BPEL, permet de relier les architectures orientées services et la logique de composition. Une approche dirigée par les modèles nous permet d'établir un pont bidirectionnel entre ces deux mondes.

Notre objectif dans la suite de cet article est de présenter cette expérience d'utilisation de l'IDM pour gérer la composition des orchestrations en soulignant les difficultés rencontrées, les limites des outils et les problèmes que soulèvent nos choix technologiques. Dans un premier temps, nous clarifions ce que nous entendons par composition d'orchestrations (cf. 2). Nous explicitons ensuite l'architecture dirigée par les modèles que nous avons mise en place et l'environnement résultant (cf. 3). Cette section caractérise et discute en particulier les choix d'outils et de modélisation effectués. Le résultat est un atelier interactif de composition des orchestrations qui est brièvement présenté dans la section (cf. 4) avant de conclure cet article en section 5.

2 Composition d'orchestrations pour l'évolution des SOA

Nous avons exploré dans des travaux précédents différents mécanismes de composition d'orchestrations. Initialement, les mécanismes décrits dans (Nemo et al., 2007a) permettent de mettre en place une composition interactive de différentes orchestrations. Cette approche a été validée dans le cadre des grilles de calculs en l'appliquant au BRONZE STANDARD (Nemo et al., 2007b), une orchestration de référence utilisée dans le domaine de l'imagerie médicale. Une autre approche vise à mettre en place des mécanismes permettant de supporter l'évolution d'orchestrations préalablement existantes (Mosser et al., 2008a). Nous présentons cette dernière sur un exemple dans cette section.

2.1 SEDUITE : une application SOA de diffusion d'informations

Nous considérons comme exemple l'application SEDUITE, servant de validation du projet ANR FAROS¹. Cette application permet la diffusion d'informations dans les milieux scolaires. Nous définissons les *sources* d'information disponibles (événements, horaires de bus, emploi du temps, ...) comme des Web Services, agrégés par composition au sein d'orchestrations. La figure 1 donne un aperçu de l'application déployée sur le site de l'École Polytechnique Universitaire de Nice – Sophia Antipolis (EPU).

Le point d'entrée de l'application est une orchestration nommée `InfoProvider`. Cette orchestration invoque toutes les sources d'informations disponibles au sein de l'infrastructure, agrège les informations obtenues et les retourne à l'utilisateur. Dans la version de l'orchestration présentée dans la figure 1, l'orchestration `MyTimeTable` filtre les données d'emploi du

¹ANR exploratoire, « *Fiabilité des ARchitectures Orientées Services* » [<http://www.lifl.fr/faros>]

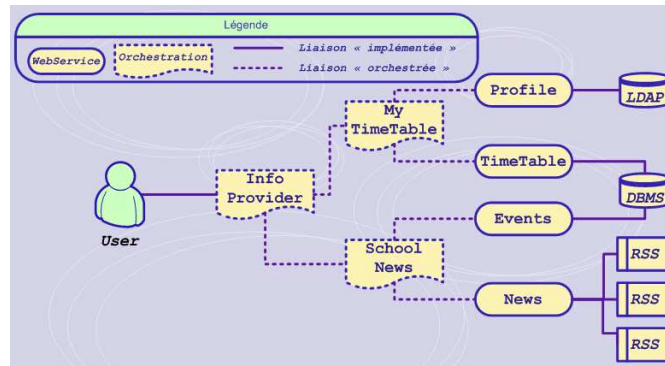
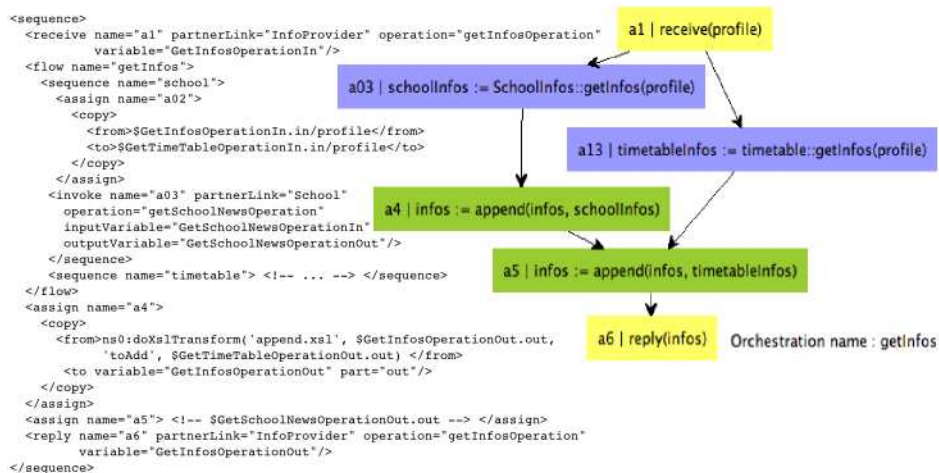


FIG. 1 – Aperçu global de l'application SEDUITE

temps en fonction des groupes d'appartenance de l'utilisateur. L'orchestration `SchoolNews` permet de récupérer des informations de divers sites partenaires (flux de syndication RSS) et de les mêler aux informations propres à la communication interne de l'établissement.

Un extrait du code BPEL correspondant à l'orchestration `InfoProvider` est présenté par la figure 2. Une autre version du logiciel est déployée au sein de l'Institut d'Éducation Sensoriel Clément Ader de Nice qui accueille et accompagne des enfants handicapés. Elle met en jeu d'autres services propres à cet établissement. De plus amples informations concernant SEDUITE et son implémentation sont disponibles sur le site de diffusion du projet².

FIG. 2 – Orchestration `InfoProvider` : Code BPEL & Représentation graphique

²<http://anubis.polytech.unice.fr/jSeduite>

2.2 Évolutions d'orchestrations

La maintenance évolutive d'un logiciel passe par son adaptation aux besoins des utilisateurs. La démarche SOA prêche une telle réactivité, favorisée par le couplage faible inter-services et les mécanismes d'assemblage de haut niveau fournis par les orchestrations. Dans le cadre de la maintenance évolutive des deux applications SEDUITE aujourd'hui en production, nous observons un ensemble d'évolutions courantes à appliquer aux orchestrations déployées. L'application de ces évolutions requiert une modification manuelle des codes en production, source d'erreur et de rupture de service.

D'un point de vue « fonctionnel », il est nécessaire dans notre contexte de pouvoir manipuler au sein de l'infrastructure SEDUITE les sources d'informations disponibles. Nous laissons aux élèves ingénieurs de l'EPU la possibilité de développer leurs propres sources d'informations pour les intégrer à SEDUITE. Ainsi, les photos des soirées étudiantes et les communications du bureau des élèves peuvent dorénavant être publiées via SEDUITE. Différents partenariats avec les acteurs du monde universitaire nous permettent aussi de publier le menu du restaurant universitaire, ou encore les horaires de départ des prochains bus. D'un point de vue « non fonctionnel », l'intégration de mécanismes d'authentification a permis d'interdire la diffusion d'informations interne à l'établissement aux utilisateurs non autorisés.

Nous avons fait le choix de représenter les évolutions d'orchestrations comme des orchestrations auxquelles nous avons ajouté des activités : *hook* qui représente l'activité à faire évoluer³, *Predecessors* (resp. *Successors*) qui désignent le positionnement des prédécesseurs (resp. successeurs) de l'activité à faire évoluer (cf. figure 4).

Ces évolutions appliquées à l'architecture se caractérisent entre autre par leurs répétitions. Il est en effet courant d'ajouter une nouvelle source d'informations au système, et un tel ajout suit toujours le même patron. Il devient alors nécessaire de pouvoir définir et composer les évolutions à appliquer au système pour permettre l'application de différentes évolutions sur l'architecture. La figure 3 montre une évolution d'ajout de source d'information dans SEDUITE, et le résultat obtenu après composition de cette évolution avec l'InfoProvider présenté en figure 2.

2.3 Une approche formelle pour la composition

La composition d'orchestrations est un processus complexe, car il ne s'agit pas de définir une composition structurelle des entités comme lors d'une composition de modèles *classique* (diagrammes de classes UML), mais d'identifier les « activités » à composer et de valider les flots résultants. Par exemple, l'ajout d'une nouvelle source d'informations ne doit pas rendre caduques les mécanismes d'authentification mis en place ou introduire des accès en écriture concurrents sur une même variable. De manière plus générale l'orchestration résultante doit respecter les contrats garantis par le service modifié comme par exemple le temps de réponse (Mosser et al., 2008b).

Nous avons défini dans (Mosser et al., 2008a) un métamodèle (ADORE) (cf. figure 4) permettant de représenter des orchestrations de Web Services et des évolutions de manière uniforme. Sur la base de ce métamodèle, nous composons les orchestrations via des règles d'inférence (implémentées dans le langage PROLOG). Lors de la composition des orchestrations

³Une analogie avec un point de coupe relativement à la programmation par aspect peut être faite ici

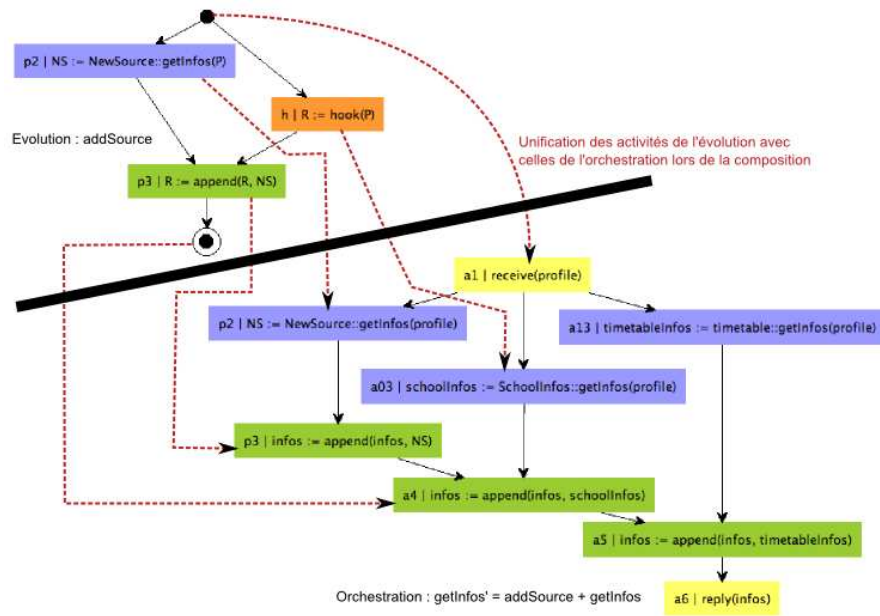


FIG. 3 – Ajout d’une nouvelle source à InfoProvider via une évolution

nos objectifs sont (i) le respect des ordres entre les activités qui composent les orchestrations, (ii) la validité des flots (en détectant et corrigeant l’indéterminisme que peut introduire une évolution : accès concurrent en lecture–écriture à une variable, levée d’exceptions différentes sous les mêmes conditions, ...) et (iii) l’optimisation des flots obtenus (en évitant par exemple des appels répétés à un même service).

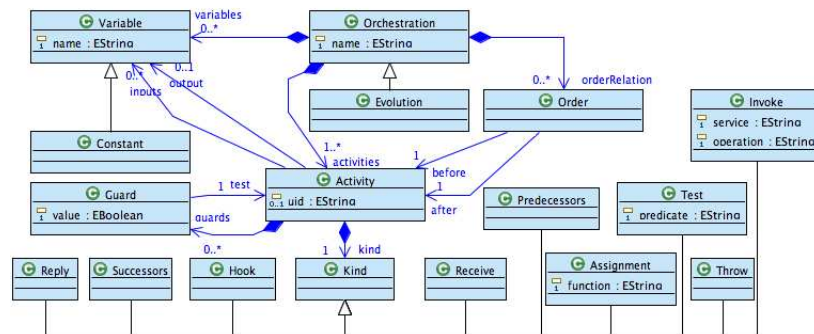


FIG. 4 – ADORÉ : « Activity moDel suppOrting oRchestratiOn Evolution »

La représentation supportant ce raisonnement repose sur l’expression de relations d’ordre

et de conditions entre les activités qui composent une orchestration. Il s'agit donc d'une représentation définitivement différente de celle utilisée par BPEL, que nous capturons dans le métamodèle ADORE. Le moteur de composition repose sur cette modélisation. Un modèle d'orchestration est représenté par une base de faits Prolog (cf. Listing 1), et l'algorithme de composition des évolutions et de tissage des évolutions est implémenté par des règles PROLOG. La présentation de cet algorithme sort du cadre de cet article ; sa description peut être trouvée dans (Mosser et al., 2008b), ou sur le site du projet⁴.

```
1 orchestration(uid,[act1, act2, ...]).
2 evolution(uid,[act1, act2, ...]).
3 activity(uid, kind, [input1, input2, ...], [output]).
4 order(activityUid, activityUid).
5 guard(activityUid,activityUid,value).
```

Listing 1 – Syntaxe d'une base de fait PROLOG conforme au moteur de composition

2.4 Scénario d'utilisation du processus de composition

L'utilisateur principal du système est un architecte logiciel spécialiste des systèmes construits par orchestration de services. Il a à sa disposition un ensemble de codes BPEL définis à l'aide des outils qu'il manipule dans son entreprise (environnements intégrés, générateurs de code, ...).

Pour faire face à la maintenance évolutive (ajout de fonctionnalités, ...) de son architecture de services, l'architecte doit faire évoluer ses orchestrations sans mettre en péril le système déployé. Pour cela, les codes BPEL sont chargés dans l'atelier de composition, où différentes évolutions lui sont proposées. Si aucune ne correspond à ses besoins, il lui est possible d'en définir de nouvelles et de les mettre à disposition dans l'atelier. Il choisit alors les évolutions qu'il souhaite appliquer à l'orchestration qu'il vient de charger.

L'algorithme de composition calcule alors l'orchestration résultante de l'application de ces évolutions au code chargé initialement. En cas d'erreur (détection de conflits de composition), la composition est interrompue et l'architecte informé. Il peut alors résoudre le conflit en ajoutant interactivement les connaissances manquantes comme une relation d'ordre entre des activités.

Du point de vue utilisateur sont uniquement manipulés des codes BPEL et des éléments de modèle conformes au métamodèle ADORE. Il n'a aucune connaissance de l'existence du moteur d'inférence PROLOG, ni du jeu de transformations mis en œuvre par l'outil qu'il manipule pour relier sa représentation BPEL à notre représentation logique.

3 Construction d'un atelier de composition *via* l'IDM

Nous présentons maintenant l'architecture dirigée par les modèles que nous avons mise en place pour élaborer l'atelier logiciel supportant la composition d'orchestrations. Nous avons fait le choix de nous focaliser dans cette présentation sur trois points clefs de cette architecture :(i) l'existence de métamodèles et d'outils de manipulation des orchestrations (cf. 3.1), (ii) le choix d'un métamodèle pivot support aux interactions avec l'utilisateur (cf. 3.2) , (iii)

⁴<http://www.adore-design.org>

une approche "programmative" des transformations impliquées dans les interactions avec l'utilisateur (cf. 3.3).

Pour chaque point clé, nous explicitons les *objectifs* qui nous ont menés à effectuer un *choix*. Nous expliquons ensuite leur *mise en œuvre* technique et *discutons* des raisons et des conséquences de nos choix.

3.1 Un point de vue global : Un monde entre deux mondes ...

Objectifs : Nous avons à notre disposition (i) un ensemble de systèmes « historiques » supportés par des outils adaptés et des ensembles d'orchestrations BPEL et (ii) un moteur de composition d'orchestrations ADORE implémenté en PROLOG. Nous sommes donc en présence de deux « *espaces technologiques* » tels que définis par Kurtev et al. (2002) : « *A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities.* ». D'une part, le domaine SOA offre de nombreux outils permettant la manipulation des orchestrations en particulier. Ce domaine recouvre un ensemble de normes requérant des compétences et une expertise particulière à la démarche de construction de ces applications à base de services. D'autre part, le domaine de la logique supporte la définition du processus de composition sur la base d'une représentation de la connaissance sous la forme de « faits ». Les objectifs de l'atelier sont alors (i) de permettre l'expression et la réutilisation d'orchestrations définies en BPEL, (ii) de semi-automatiser leurs compositions et (iii) de générer en retour une nouvelle orchestration BPEL.

Choix effectués : Nous faisons le choix d'adopter une démarche d'intégration par transformation. L'objectif est donc d'interfacer les deux domaines évoqués précédemment en opérationnalisant une chaîne de transformations allant des codes BPEL au moteur de composition PROLOG (et réciproquement), **sans modifier l'un ou l'autre**. Nous pouvons en effet appréhender à un niveau d'abstraction supérieur l'architecture comme une transformation d'un code BPEL vers des faits PROLOG et inversement. Néanmoins la nécessité d'interagir avec l'architecte voulant composer des orchestrations (et qui ne connaît pas un expert PROLOG), force la présence d'une représentation des orchestrations qui soit facilement compréhensible par l'utilisateur et par le moteur de raisonnement. Cette représentation est le métamodèle pivot ADORE

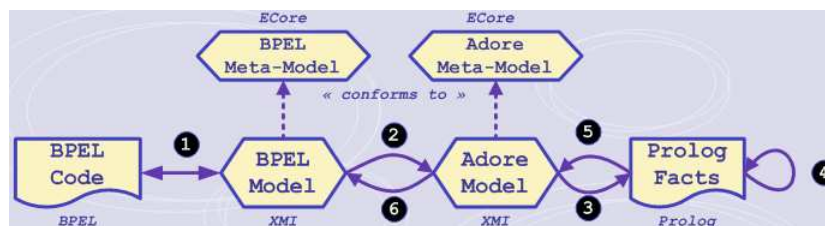


FIG. 5 – Transformations mises en œuvre (τ_1 & τ_4 données)

Nous avons identifié six transformations τ_n dans cette démarche :

- τ_1 : Passage du code BPEL à un modèle BPEL manipulable par les outils usuels.

Composition d'orchestrations dirigée par les modèles

- τ_2 : Passage d'un modèle BPEL industriel à une représentation ADORE.
- τ_3 : Passage du modèle de raisonnement ADORE à des faits PROLOG.
- τ_4 : Composition d'orchestrations (transformation endogène)
- τ_5 : Analyse d'une base de faits pour créer un modèle ADORE.
- τ_6 : Transformation d'un modèle de raisonnement en modèle exécutable.

A l'intérieur de ce jeu de transformations, nous disposons automatiquement de τ_1 et de τ_4 , que nous ne modifions pas. En effet, τ_1 est fournie par l'implémentation des outils BPEL du projet ECLIPSE WTP/STP, qui met à disposition un métamodèle de BPEL ainsi qu'un outillage permettant de manipuler le langage au travers d'outils «modèles». Quand à τ_4 , il s'agit du moteur de composition d'orchestrations que nous avons développé en PROLOG.

Discussion : *Transformations et évolutions des codes* : Notre objectif n'est pas de maintenir une synchronisation entre le code PROLOG et les modèles d'orchestrations initiaux puisque nous sommes dans un processus de génération : les orchestrations résultantes sont de nouvelles orchestrations. La cohérence que l'on veut maintenir entre le code initial et le code résultat porte alors sur le nommage des différentes activités et variables composant les orchestrations et évolutions initiales, et la vérification de propriétés telles que la préservation de toutes les activités, le respect des ordres entre les activités, l'absence de cycles, ... Le respect de ces propriétés est pris en charge par l'algorithme de composition. Par contre des modifications ultérieures des orchestrations initiales ne sont pas automatiquement répercutées sur les orchestrations créées actuellement. Ce type de transformations cataloguées de bi-directionnelles bijectives dans (Stevens, 2008) reste à notre connaissance difficile à mettre en oeuvre dans les systèmes de transformations existants. L'introduction de l'utilisateur dans le contexte de notre application ne simplifie pas les choses. Enfin l'introduction dans le procédé de transformation d'un métamodèle ne supportant pas tous les artefacts BPEL comme le nommage de liens entre activités nous conduit à une perte d'informations discutée ci-après.

3.2 Intérêts d'un métamodèle «pivot» support aux interactions avec l'utilisateur (τ_2, τ_6)

Objectifs : Le métamodèle d'orchestrations BPEL publié par ECLIPSE⁵ est une représentation conceptuelle dirigée par les opérateurs du langage sous-jacent. Dans l'univers PROLOG, nous disposons d'une représentation différente des orchestrations puisque dirigée par une relation d'ordre partiel entre les différentes activités de l'orchestration. Il s'agit ici de construire une passerelle entre un ensemble d'outils provenant de l'univers ECLIPSE et un moteur d'inférence ne contenant que les concepts nécessaires à la réalisation de nos objectifs, tout en interagissant avec l'utilisateur lors de la résolution de conflits de compositions.

Choix effectué : Nous défendons une approche «pivot», inspirée du principe de *code intermédiaire* présent dans le domaine de la compilation. Nous définissons une représentation intermédiaire entre les deux domaines qui nous permet de cristalliser les orchestrations dans une version adaptée à notre raisonnement et à la communication avec l'architecte et également plus concise et moins complexe que dans une représentation exécutable (modèle BPEL) ou relationnelle (faits PROLOG).

⁵<http://www.eclipse.org/bpel/developers/model.php>

Mise en œuvre : Nous avons fait le choix de représenter le métamodèle pivot ADORE dans le formalisme ECORE, fourni par le canevas logiciel EMF (Merks et al., 2003). La figure 4 représente dans ce formalisme le métamodèle ADORE. L'utilisation du canevas logiciel EMF fournit en effet de manière automatique un ensemble de fonctionnalités intéressantes (génération de code, persistance, contraintes, notification, réflexivité, ...) pour définir des métamodèles dans une démarche d'opérationnalisation. ECORE étant devenu un standard, nous obtenons aussi avec son utilisation un point d'entrée dans de nombreuses technologies de transformations (ATL, KERMETA, XSLT, ...). Si le formalisme ECORE possède une expressivité moindre qu'UML (absence de visibilité, contenance, ...), il est néanmoins suffisant pour représenter les concepts inhérents à notre vision des orchestrations de Web Services.

Les transformations de modèles permettant le passage de modèles BPEL à des modèles ADORE (τ_2 et τ_6) sont implémentés à l'aide du langage dédié KERMETA (Muller et al., 2005). Si l'importation de code BPEL est actuellement opérationnelle (τ_2), la transformation inverse τ_6 est en cours de développement.

Discussions : *Evolutions et métamodèle pivot ?* La complexité en terme du nombre de transformations dans la solution que nous proposons par rapport à des transformations directes de métamodèle BPEL au code PROLOG est plus importante, puisqu'au lieu de deux transformations, nous en avons défini quatre. Cependant une transformation directe du métamodèle BPEL vers PROLOG présente plusieurs désavantages. En effet, le métamodèle BPEL est complexe (97 concepts, ...) et basé sur la syntaxe abstraite du langage. Il évolue au fil des modifications des normes qui sont mouvantes car récentes et supportée par une communauté active. L'expression des orchestrations sous forme de faits PROLOG est elle-même sujette aux évolutions du moteur de composition. Des transformations directes entre ces deux domaines ne manipulent pas alors des concepts mais les artefacts des langages. Le choix d'un métamodèle pivot réduit la distance entre les deux domaines (Terrasse et al., 2002) en identifiant les concepts communs aux deux représentations, indépendamment des technologies de mises en œuvre. Aussi, l'approche pivot est-elle reprise par de nombreux projets (procédé FAROS, plate-forme ECLIPSE STP et son *Intermediate Model*, ...). Nous utilisons également le pivot, pour atteindre d'autres formalismes dont π -diapason (Pourraz, 2007) qui est à l'étude.

Un métamodèle pivot support aux interactions avec l'utilisateur Le métamodèle pivot n'est pas seulement là pour expliciter un pont, il est aussi un outil d'interaction avec l'utilisateur. Cet argument nous a conduit non pas à définir un métamodèle proche de PROLOG, mais de la représentation abstraite sur laquelle l'algorithme de composition est construit. Bien que nous manquions à ce jour d'expérimentations pour évaluer la pertinence de ce métamodèle dans les usages, nous sommes bien certains, pour avoir donné le métamodèle de BPEL à nos étudiants, que la manipulation de ce métamodèle intermédiaire est bien plus « simple ». La justification du métamodèle pivot est donc dans notre cas également étayée par un fort besoin d'une représentation intermédiaire compréhensible par l'utilisateur final et suffisamment abstraite pour pouvoir communiquer indépendamment de la syntaxe.

Le choix d'une représentation ECORE du métamodèle ADORE nous a permis d'opérationnaliser ce métamodèle en définissant les interfaces graphiques adaptées à moindre coût. Alors même que nous avons refusé de définir le moteur de composition sur cette représentation, nous sommes aujourd'hui convaincu de l'intérêt d'environnements intégrés pour supporter les

manipulations opérationnelles des modèles. Les travaux importants menés aujourd'hui autour des transformations devraient nous permettre de bénéficier de ces environnements tout en ne perdant pas les acquis d'autres domaines dont en l'occurrence la logique des prédicats.

Indépendance des métamodèles et implantation des transformations : Si le choix du langage de transformation s'est imposé en partie par notre collaboration dans le cadre du projet FAROS avec l'équipe TRISKELL qui définit le langage KERMETA, nous utilisons intensivement son approche non-invasive. KERMETA se définit en effet comme un tisseur d'aspects adapté aux métamodèles ECORE, capable de manipuler directement ceux-ci sans aucune modification. Définir une transformation en utilisant KERMETA revient à implanter un (ou plusieurs dans le cadre de transformations complexes) *visiteurs* (au sens du patron de conception *Visiteur*) du métamodèle source. Le tisseur d'aspects fourni par KERMETA permet le *tissage* du patron de conception dans le métamodèle source, ainsi que l'ajout des informations de traçabilité propres à la transformation. Ces informations ne sont associées au métamodèle que pendant l'exécution de la transformation et n'affectent donc en rien son contenu (l'intelligence de la transformation étant portée par le code du visiteur tissé).

Transformations bi-directionnelles et traçabilité : Les transformations τ_2 et τ_6 peuvent être appréhendées comme inverse l'une de l'autre puisque nous transformons un modèle BPEL en ADORE et inversement; nous aurions donc aimé écrire une seule transformation interprétable dans les deux sens. Le choix du langage KERMETA, impératif, ne nous permet pas d'explorer la bi-directionnalité. Cependant, en étudiant de plus près la transformation τ_2 , nous constatons que lors de la transformation du modèle BPEL vers ADORE, nous perdons des informations. Par exemple, une séquence entre activités ou un lien BPEL sont transformés de la même manière en ADORE (une relation d'ordre entre activités) parce qu'en terme d'exécution ces deux flots expriment la même chose. Nous ne savons pas actuellement retrouver cette information lors des transformations inverses. Or pour l'utilisateur final ce point est perturbant : même si sémantiquement les résultats sont équivalents, il n'y a pas une correspondance directe avec ce qu'il avait exprimé. Nous rencontrons la même difficulté avec les liens nommés en BPEL, dont nous perdons le nommage en ADORE. Par contre l'introduction d'annotations ou de traces lors des transformations (Amar et al., 2008) pourrait nous permettre de retrouver des informations et de conduire la construction des orchestrations composites résultantes, en utilisant le vocabulaire initial lorsque c'est possible. Dans ce cas, les deux transformations τ_2 et τ_6 ne sont plus vraiment l'inverse l'une de l'autre, puisque l'utilisation de la traçabilité diffère de l'une à l'autre.

3.3 Transformations utilisées à l'exécution (τ_3, τ_5)

Objectifs : Nous disposons maintenant d'un métamodèle pivot ADORE au format ECORE. L'algorithme de composition est implémenté dans le langage PROLOG. L'implémentation utilise l'inférence et le *backtracking* fournis nativement par l'interpréteur. La technique de composition utilisée (basée sur le principe *échec-réussite*) implique que l'utilisateur du procédé puisse interagir avec le moteur de composition durant le processus, en particulier lors de conflits de composition (accès concurrents en écriture à une même variable, composition de valeurs de retour, ...).

Choix effectué : Nous devons supporter des transformations automatiques allant d'ECORE à PROLOG et inversement, mais aussi permettre dans le même temps une interaction de l'utilisateur à destination de l'interprète PROLOG. Nous sortons donc du cadre des langages dédiés aux transformations de modèles et devons utiliser un langage généraliste.

Mise en œuvre : Dans le cadre du canevas logiciel EMF, nous disposons d'un générateur de code capable de générer un jeu de classes JAVA correspondant au métamodèle défini dans Ecore⁶. Coté PROLOG, l'éditeur SWI propose une interface avec le langage sous la forme d'une API (Wielemaker, 2003). Le listing 1 donne la syntaxe des faits à générer pour pouvoir utiliser le moteur de composition. La transformation τ_3 , implémentée en JAVA, consiste donc en une visite du graphe d'objets EMF ciblant les classes de l'API SWI-PROLOG. La transformation inverse (τ_5) est implémentée par interrogations successives de l'interprète afin de créer les objets JAVA correspondants aux faits reçus. Lors du raisonnement, les conflits détectés en PROLOG, sont remontés vers JAVA sous la forme d'exception pour interrompre l'exécution courante et une interface permet à l'utilisateur d'explicitier ses choix qui sont transformés en des directives PROLOG.

Discussions : *Pourquoi une transformation dédiée ?* Il existe des outils et transformations fournis entre autre par SWI-PROLOG qui permettent d'obtenir automatiquement une transformation bi-directionnelle entre un modèle objet et une base de faits PROLOG, voir directement d'une modélisation Ecore à PROLOG (Hessellund et al., 2007). Dans ces approches, ces transformations sont uniquement guidées par la structure du modèle objet, ce qui permet une transformation bi-directionnelle induite par une bijection naturelle entre les structures utilisées : à toute classe C est associé un fait `class(c, ...)`, ... Si cette transformation garantit un «aller-retour» sans aucun coût de développement entre le domaine de l'objet et l'univers de PROLOG, elle rend obligatoire l'adaptation de l'algorithme de composition pour raisonner sur ces nouveaux faits. Cette approche va donc à l'encontre du principe énoncé au départ - ne pas modifier l'algorithme de composition considéré comme donné- et suppose d'établir notre raisonnement sur un modèle dont l'intention ne correspond pas du tout à nos objectifs car pollué par des concepts qui lui sont étrangers (en l'occurrence `classe`, `inheritance`, ...)⁷.

«Opérationnalisation» des transformations : Nous trouvons dans la littérature des métamodélisations du langage PROLOG (Blanc et al., 2005). L'un des problèmes inhérents à ces métamodèles est le manque d'outillage sous-jacent. En effet, afin d'opérationnaliser la chaîne de transformation, nous devons atteindre de manière automatique un interprète PROLOG exécutant dans la plate-forme le processus de composition. L'administrateur du système doit ensuite pouvoir envoyer des commandes dans le moteur pour effectuer les composition voulues, de manière interactive. Une approche statique usuelle n'est alors pas envisageable : cette transformation doit s'effectuer à l'exécution de l'environnement de composition.

Outre les points évoqués précédemment, le choix de JAVA pour l'implémentation des transformations τ_3 et τ_5 a été motivé par la réutilisation d'ateliers existants. Nous disposons de

⁶Cette transformation de Ecore vers JAVA n'a pas été visualisée dans la figure 5 parce qu'elle est fournie par EMF et que nous ne la discutons pas.

⁷Rappelons une des définitions d'un modèle : "A model is a simplification of a system, with a certain purpose." [MOF]. Forcer la définition d'un modèle à se conformer à des outils ou à un courant va à l'encontre d'une approche dirigée par les modèles, de notre point de vue.

facto, suite au choix d'ECLIPSE, d'un ensemble d'outils de type *langages* ou canevas logiciels aisément exploitables, dès que nous manipulons les éléments du modèle comme des objets JAVA. Ainsi, l'intégration du langage PROLOG via l'implémentation des visiteurs *ad hoc* sur la représentation JAVA du métamodèle ADORE a été réalisée dans le cadre d'un projet de deux semaines par quatre étudiants de quatrième année (Fabien DILET, Gabriel GUY, Pauline MARTE et Grégory SOUTADE) en utilisant l'API SWI-PROLOG. Ce projet a aussi implémenté la transformation inverse de la représentation PROLOG vers la représentation ECORE des orchestrations.

4 DRADORE : Un atelier de composition des orchestrations

L'atelier logiciel permettant la composition interactive des orchestrations a été défini sur la base de l'architecture présentée précédemment (FIG. 6). Il interface les deux domaines à l'exécution, en permettant à l'utilisateur de naviguer d'un monde à l'autre. L'utilisateur dispose d'un interprète interactif capable d'appliquer des actions aux modèles manipulés. Il peut ainsi

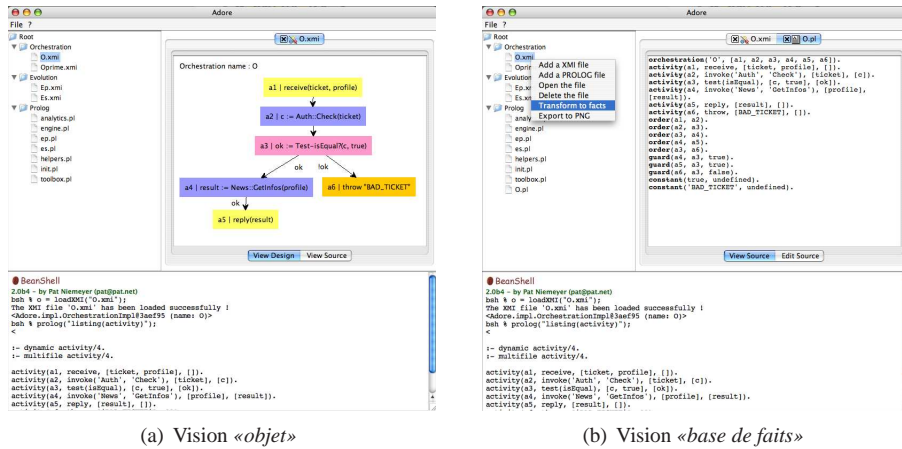


FIG. 6 – DRADORE, un atelier logiciel de composition d'orchestrations

charger et afficher sous forme graphique des orchestrations conformes au métamodèle ADORE (cf. FIG 6(a)), appliquer τ_3 depuis l'interface pour obtenir la représentation équivalente sous la forme de faits PROLOG (cf. FIG 6(b)). L'application des transformations est transparente pour l'utilisateur, qui peut aussi bien utiliser la représentation des orchestrations sous formes de faits PROLOG ou de graphes d'objets JAVA. Ainsi, il devient possible pour un utilisateur spécialiste de la composition d'utiliser des orchestrations émanant d'un expert métier SOA. Et, de manière réciproque, un spécialiste SOA peut bénéficier de l'algorithme de composition sur la même base de travail.

L'interactivité nécessaire au processus de composition (résolution de conflit, validation *métier* des codes produits, ...) est fournie à l'utilisateur sous la forme d'un «*shell*» (en intégrant l'outil BEANSHELL) interfacé avec le code JAVA généré depuis le métamodèle. Une trans-

formation à destination de l'API JGRAPH a été implémentée pour permettre la visualisation graphique des orchestrations.

4.1 Opérationnalisation de l'algorithme de composition

Nous nous positionnons ici sur un cas d'exemple inspiré de l'application SEDUITE (cf. section 2) pour illustrer le fonctionnement de l'atelier. Nous souhaitons composer une évolution d'ajout de sources d'information *AddEvents* à une évolution vérifiant la validité du profil de l'utilisateur *Profile*. Le listing 2 représente l'utilisation du moteur de composition sur ce cas d'utilisation.

```

1 bsh % events = loadXMI("AddEvents.xmi");
2   The XMI file 'AddEvents.xmi' has been loaded successfully !
3   <Adore.impl.EvolutionImpl@6bb0d4 (name: addEvents)>
4 bsh % profile = loadXMI("Profile.xmi");
5   The XMI file 'Profile.xmi' has been loaded successfully !
6   <Adore.impl.EvolutionImpl@b923ee (name: profile)>
7 bsh % profileAndEvents = apply("merge_evolution",
8     new Object[] { new Adore.Evolution[] {profile, addEvents}}, "Adore.Evolution");
9   <Adore.impl.EvolutionImpl@935c95 (name: mergeResult)>
10 bsh % draw(profileAndEvents);
11 bsh % saveXMI(profileAndEvents);
12   The orchestration 'profileAndEvents' has been saved successfully in
13   '~/workspaces/mm4wsoa/DrAdore/models/profileAndEvents.xmi' !

```

Listing 2 – *Composition interactive dans le shell DRADORE*

L'utilisateur commence par charger *AddEvents* et *Profile* (l. 1 et 4) à partir de leur représentation serialisée. Il obtient en réponse à l'invocation de la commande `loadXMI` des objets JAVA. Par effet de bord, cette action de chargement applique τ_3 pour alimenter la base de faits PROLOG de manière transparente. L'utilisateur applique ensuite la fonction de composition τ_4 en appliquant une règle PROLOG aux objets JAVA récupérés du chargement (l. 7). Il obtient en résultat un objet JAVA de type `Adore.Evolution`. Ce nouvel objet résultat est construit par application de τ_5 à la base de faits résultant de l'application de la règle de composition (un «*cast*» automatique est effectué pour faciliter la manipulation dans le *shell*). Ce nouvel objet est ainsi manipulable au même titre que les autres. Il peut être affiché sous forme graphique (l. 10) ou encore sérialisé (l. 11) à des fins de sauvegarde.

5 Conclusion & Perspectives

La composition d'orchestrations telle que présentée dans cet article repose sur une architecture dirigée par les modèles qui supporte l'interopérabilité entre le domaine des architectures de services et le domaine de la logique. Cette approche étend le cadre «classique» de l'IDM puisque nous ne cibons pas une production du code mais un cheminement du code aux modèles, composition des modèles en interaction avec l'utilisateur puis génération des codes. Nous avons montré comment cette architecture a été bâtie dans une approche pragmatique d'utilisation des outils existants et nous avons discuté nos choix.

Nous avons en particulier mis en avant l'intérêt d'un métamodèle pivot qui offre une représentation des orchestrations plus proches des artefacts nécessaire au raisonnement, aux interactions avec l'utilisateur et ainsi à la résolution des conflits. Nous avons déjà rencontré ce besoin de modèle intermédiaire dans (Caron et al., 2007). De récentes manipulations avec des

étudiants nous confortent dans cette approche ; le métamodèle BPEL trop proche de la syntaxe est très difficile à appréhender directement. Néanmoins, cette approche a exigé l'écriture d'un nombre plus important de transformations qu'une approche directe de transformation code vers code. Nous retrouvons ici les arguments de l'interprétation abstraite (Cousot, 2000).

La construction de l'atelier nous a conduit à nous interroger sur le choix des langages de transformation. Notre conclusion pour l'heure est qu'il n'existe pas de langage de transformation «idéal» qui réponde à l'ensemble des besoins. Ainsi nous avons utilisé des transformations Kermeta pour des transformations modèle vers modèle, des transformations programmées pour supporter l'interactivité et bénéficier du support d'API existantes, des transformations en PROLOG pour composer les orchestrations et bénéficier des capacités d'inférence PROLOG. Cette multiplicité des représentations et des outils a un coût important en terme d'expertises nécessaires au développement et à la maintenance. Heureusement l'intégration dans le framework EMF est assurément un moyen de réduire, après un temps d'apprentissage non négligeable, ces temps de développement. Des extensions en cours de l'atelier en utilisant GMF (Eclipse, 2009) pour la construction d'éditeurs dédiés nous confortent dans l'idée que des plates-formes IDM intégrant différents domaines technologiques sont vraiment nécessaire à un usage plus large de l'IDM.

Dans cette application de l'IDM, nous avons été également confrontés à la mise en place de systèmes de transformations qui auraient gagné à bénéficier de la trace - pour gérer la distance entre des métamodèles ne contenant pas les mêmes informations-, de la bi-directionnalité - pour écrire moins de transformations et faciliter la maintenance -, du maintien de cohérence entre les modèles, ... Ces différents points font partie de nos perspectives à court terme, en nous orientant vers des langages tels que QVT (Bast et al., 2005) ou CONDOR (Mens et al., 2006) qui ne présentaient cependant pas jusqu'à ce jour, de notre point de vue, une maturité suffisante pour être intégrés dans un environnement interactif.

Références

- Amar, B., J.-R. Falleri, M. Huchard, C. Nebut, et H. Leblanc (2008). Un framework de traçabilité pour des transformations à caractère impératif. In *Conférence sur les Langages et Modèles à Objets (LMO), Montréal (Canada)*, pp. 141–154. Cépactuèd Editions.
- Bast, W., M. Belaunde, X. Blanc, K. Duddy, C. Griffin, S. Helsen, M. Lawley, M. Murphree, S. Reddy, S. Sendall, J. Steel, L. Tratt, R. Venkatesh, et D. Vojtisek (2005). MOF QVT final adopted specification. OMG document `ptc/05-11-01`.
- Blanc, X., F. Ramalho, et J. Robin (2005). Metamodel Reuse with MOF. In L. C. Briand et C. Williams (Eds.), *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings*, Volume 3713 of *Lecture Notes in Computer Science*, pp. 661–675. Springer.
- Caron, P.-A., M. Blay-Fornarino, et X. Le Pallec (2007). La contextualisation de modèles, une étape indispensable à un développement dirigé par les modèles ? *RSTI - Série L'Objet (RSTI-Objet) 13/4*, 55–71.
- Cousot, P. (2000). Interprétation abstraite. *Technique et science informatiques 19*(1-2-3), 155–164.

- Eclipse (2009). Eclipse graphical modeling framework (gmf) website. <http://www.eclipse.org/modeling/gmf/>.
- Hessellund, A., K. Czarnecki, et A. Wasowski (2007). Guided Development with Multiple Domain-Specific Languages. In G. Engels, B. Opdyke, D. C. Schmidt, et F. Weil (Eds.), *MoDELS*, Volume 4735 of *Lecture Notes in Computer Science*, pp. 46–60. Springer.
- Kurtev, I., J. Bézivin, et M. Aksit (2002). Technological Spaces : an Initial Appraisal. In *CoopIS, DOA'2002 Federated Conferences, Industrial track*, Irvine.
- Mens, T., G. Kniesel, et O. Runge (2006). Transformation Dependency Analysis : a Comparison of Two Approaches. In *Proc. LMO 2006*, pp. 167–182. Hermes Science Publications, Lavoisier.
- Merks, E., R. Eliersick, T. Grose, F. Budinsky, et D. Steinberg (2003). *The Eclipse Modeling Framework*. Addison Wesley.
- Mosser, S., M. Blay-Fornarino, P. Collet, et P. Lahire (2008b). Vers l'intégration dynamique de contrats dans des architectures orientées services : une expérience applicative du modèle au code. In *2ème Conférence sur les Architectures Logicielles (CAL'08)*, Montréal, pp. 14.
- Mosser, S., M. Blay-Fornarino, et M. Riveill (2008a). Web Services Orchestration Evolution : a Merge Process for Behavioral Evolution. In *2nd European Conference on Software Architecture (ECSA'08)*, Paphos, Cyprus. Springer LNCS.
- Muller, P.-A., F. Fleurey, et J.-M. Jézéquel (2005). Weaving Executability into Object-Oriented Meta-Languages. In *Proc. of MODELS/UML'2005*, LNCS, Jamaica. Springer.
- Nemo, C., M. Blay-Fornarino, G. Kniesel, et M. Riveill (2007a). Semantic Orchestration Merging - Towards Composition of Overlapping Orchestrations. In J. Filipe (Ed.), *9th International Conference on Enterprise Information Systems (ICEIS'2007)*, Funchal, Madeira.
- Nemo, C., T. Glatard, M. Blay-Fornarino, et J. Montagnat (2007b). Merging Overlapping Orchestrations : an Application to the Bronze Standard Medical Application. In *International Conference on Services Computing (SCC 2007)*, Salt Lake City, Utah, USA, pp. 364–371. IEEE Computer Engineering.
- OASIS (2007). Web Services Business Process Execution Language Version 2.0. Technical report, OASIS.
- Papazoglou, M. P. et W. J. V. D. Heuvel (2006). Service Oriented Design and Development Methodology. *Int. J. Web Eng. Technol.* 2(4), 412–442.
- Pourraz, F. (2007). *Diapason : une approche formelle et centrée architecture pour la composition évolutive de services Web*. Ph. D. thesis, Université de Savoie.
- Stevens, P. (2008). A Landscape of Bidirectional Model Transformations. In *Post-proceedings of GTTSE'07*.
- Terrasse, M.-N., M. Savonnet, et G. Becker (2002). Métamodélisation et interopérabilité des systèmes d'information. In *INFORSID*, pp. 291–305.
- White, S. A. (2006). *Business Process Modeling Notation (BPMN)*. IBM Corp.
- Wielemaker, J. (2003). An Overview of the SWI-Prolog Programming Environment. In F. Mesnard et A. Serebenik (Eds.), *Proceedings of the 13th International Workshop on Logic Programming Environments*, Heverlee, Belgium, pp. 1–16. KUL. CW 371.

Summary

Services Oriented Architecture are a common solution to build complex systems. Following these guidelines, *services* reify elementary bricks of the system. Services are then assembled using high level mechanisms like *orchestrations*. We present here a software supporting Orchestration composition. This software was build following Moden Driven Development methodology. We expose technological choices and discuss them.