Composition de Workflows à l'aide de la Configuration

Patrick Albert, Laurent Henocque, Mathias Kleiner

Composition de Workflows à l'aide de la Configuration

Patrick Albert Laurent Henocque Mathias Kleiner

ILOG, 9 rue de Verdun,94250 Gentilly France
and
Laboratoire des Sciences de l'Information et des Systèmes,
Université Paul Cézanne, Avenue Escadrille Normandie Niemen
13397 Marseille cedex 20, France
{palbert, mkleiner}@ilog.com, laurent.henocque@lsis.org

Abstract

La composition automatique ou assistée de workflows est un domaine de recherche intense pour des applications dédiées au web ou à la modélisation de processuss métiers (business process). La composition de workflow peut être résolue de diverses manières, généralement grâce à des techniques de preuve de théorèmes. L'originalité de cette recherche provient de l'observation que la construction d'un workflow composite comporte de fortes similitudes avec la recherche d'un modèle fini, et que certains langages de workflows peuvent être définis comme des métamodèles contraints [9, 23]. Ceci conduit à étudier la possibilité d'appliquer à ce problème les techniques de configuration. Notre principale contribution est de prouver la faisabilité d'une telle approche, avec certains avantages relativement à l'utilisation de techniques purement logiques.

Nous présentons un modèle objet contraint pour la composition de workflows, basé sur un metamodèle de workflows et d'ontologies pour les processus et les flux de données. Des résultats expérimentaux sont listés pour une implémentation qui génère des workflows composites complexes incluant des transformations et des noeuds de synchronisations (fork/join) et de branchements (split/merge).

1 Introduction

La composition automatique ou assistée de workflows est un domaine de recherche intense pour des applications dédiées au web ou à la modélisation de business process. La composition de workflow peut être résolue de diverses manières, et généralement grâce à des techniques de preuves de théorèmes.

L'originalité de cette recherche provient de l'observation que la construction d'un workflow composite comporte de fortes similitudes avec la recherche d'un modèle fini, et que certains langages de workflows peuvent être définis comme des métamodèles contraints [9, 23], ce qui conduit naturellement à étudier la possibilité d'appliquer à ce problème les techniques de configuration. Notre principale contribution est de prouver la faisabilité d'une telle approche, qui présente certains avantages si on la compare à des techniques purement logiques.

Le plan de l'article est le suivant : la section courante 1 est introductive, et présente brièvement la configuration dans 1.1, comment traiter le problème de composition grâce à un configurateur dans 1.2 et un bref état de l'art sur le sujet dans 1.3. La section 2 donne la spécification d'un modèle objet contraint combinant un métamodèle des activités et les ontologies de données. La section 3 détaille la formulation du problème. La section 4 explicite le processus de configuration. Deux exemples sont présentés en section 5 ainsi que les résultats obtenus avec le programme ILOG JConfigurator. Enfin la section 6 conclut and propose des perspectives de recherches.

1.1 Brève introduction à la configuration

La configuration consiste à construire (simuler la construction d') un *produit complexe* à partir de *composants* choisis dans un catalogue de *types*.

Le nombre et le type des composants requis n'est pas connu à l'avance. Les composants sont sousmis à des *relations*, et leurs types sont décrits par une *hié-*

rarchie de types. Des *contraintes* (où 'règles de bonne formation') définissent tous les produits valides. Un configurateur reçoit en entrée un sous-ensemble de la structure objet souhaitée, et l'étend à une solution du problème de configuration, si elle existe. Ce problème est semi-décidable dans le cas général (certains problèmes n'ont que des solutions infinies).

Un programme de configuration est correctement décrit au moyen d'un *modèle objet contraint* combinant un diagramme de classes standard (comme illustré par les figures 2, 3), conjointement avec des contraintes (aussi appelées 'sémantique' dans le cadre UML/OCL). Résoudre le problème associé d'énumération peut être réalisé avec des approches techniques ou des formalismes variés : les extensions du cadre CSP [12, 6], les approches à base de connaissances [21], les logiques terminologiques [13], la programmation logique (utilisant le chaînage avant ou arrière, et des sémantiques non standard) [20], les approches orientées objet [10, 21]. Nos expériences ont été menées avec le configurateur orienté objet ILOG JConfigurator [10].

Un configurateur orienté objet tel qu'ILOG JConfigurator représente son catalogue de types grâce à un modèle objet contenant des classes, leurs attibuts et leurs relations. Les relations sur les classes sont des associations ou l'héritage. L'utilisateur d'un configurateur définit un modèle objet pour le champ de connaissance à implémenter, conjointement avec des contraintes associées. En effet, le modèle objet n'est généralement pas suffisant pour représenter exactement la sémantique du domaine.

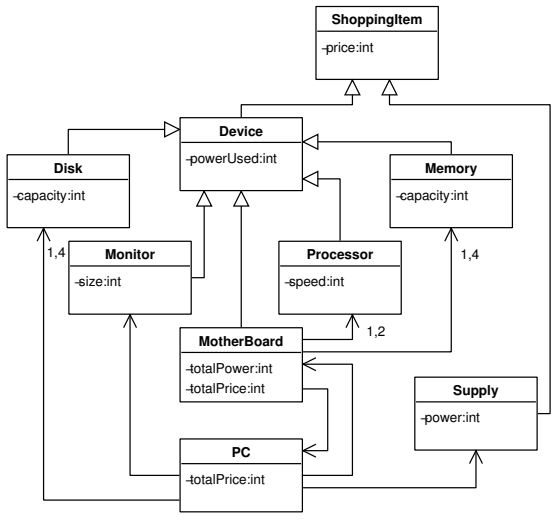

FIG. 1 – Un modèle objet pour la configuration de PC

Par exemple, pour représenter des PC's[1], on pourrait utiliser le modèle décrit par la figure 1. On y voit qu'un *PC* doit avoir exactement une *Motherboard*[2], exactement un *Power Supply*, et un unique *Monitor*. En revanche, il peut contenir jusqu'à quatre *Disk(s)*[3]. Tout PC (simplifié) peut être représenté par une instance de ce modèle. Des contraintes additionnelles garantissent la validité de l'instance. Par exemple, l'attribut *totalPrice* d'une instance de la classe *PC* est égal à :

$$sum(MB.totalPrice, Su.price, Mon.price, Disks.price)$$

Dans cette exemple, la notation pointée[4] est utilisée pour déréférencer les attributs d'une classe à travers leurs associations, et $Disks.price$ représente l'ensemble aggrégé des prix sur tous les disques connus du PC, quel que soit leur nombre.

1.2 De la configuration à la composition de workflows

La configuration est une technique émergente de l'IA avec des applications à de nombreux domaines, dans lequels le problème peut être formulé comme la production d'une instance finie d'un modèle objet soumis à des contraintes. Raisonner sur les workflows relève de cette catégorie car la description d'un workflow est une instance d'un métamodèle donné (tel que peut l'être notamment le métamodèle UML pour les diagrammes d'activité [9]). La composition de workflow est un problème de configuration dans le sens où il est nécessaire d'introduire un certain nombre de transitions et de noeuds n'existant pas dans les workflows donnés en arguments (fork, join, split, merge, transformations, séquences prédéfinies d'interaction avec l'utilisateur), et d'interconnecter les messages en entrée et sortie des actions pourvu qu'ils aient un type compatible. L'utilisateur d'un système de composition de workflows fournit :

- une liste des workflows candidats, sous la forme d'instances partiellement définies du métamodèle utilisé (e.g un producteur de certains objets, un transporteur, un service de paiement electronique, etc.)
- les ontologies pour les types de données des messages entrants/sortants de ces workflows (e.g type de voyage, les méthodes de paiement, etc.)
- le but à atteindre pour le résultat de la composition (e.g la réservation d'un ticket de train)
- la liste des informations qui pourront être fournies par l'utilisateur du workflow composite (e.g un

[1] I.e. Personal Computers

[2] La cardinalité par défaut est 1 dans nos diagrammes UML.

[3] les flèches simples labellisées X, Y représentent une relation dont la cardinalité va de X à Y

[4] Cette notation tirée du langage UML+OCL language [9] a été identifiée comme relativement adaptée à la spécification de problèmes de configuration dans [5].

numéro de carte de crédit, le nom d'un produit, accepter/refuser une offre,etc.)

Dans notre approche, le but est défini comme un type de message (dans une ontologie appropriée) attaché au noeud final du workflow composite. Les entrées fournies par l'utilisateur sont modélisées grâce aux signaux externes (un type particulier d'activité).

L'utilisateur du sytème de composition attend en retour un workflow composite complet, qui combine les workflows candidats en garantissant la validité de toutes les contraintes d'intégrité. Parmi ces contraintes, certaines proviennent du métamodèle : par exemple des contraintes garantissent que deux (ou plus) workflows ne peuvent pas se bloquer mutuellement, chacune attendant que l'autre envoie un message. D'autres contraintes sont plus spécifiques au problème : par exemple celles qui assurent que l'objet produit est effectivement celui qui est transporté.

1.3 Travaux précédents

La composition automatique de workflows est un champ d'intense activité, avec des applications possibles à au moins deux domaines majeurs : la modélisation de business process et le web sémantique. Des tentatives pour résoudre ce problème sont expérimentées avec de nombreux formalismes et techniques.

- Situation calculus [17] : dans [11], l'extension concurrente ConGolog de Golog est montrée adaptée à la composition de workflows. Pour contourner la difficulté provenant de ce que l'instruction Golog "Sequence" est statique, et ne permet pas l'insertion dynamique d'actions, les auteurs introduisent une instruction supplémentaire "Order".
- Logic programming : les résultats de [18] illustrent la possibilité d'utiliser Prolog pour générer interactivement des compositions de services Web en s'appuyant sur leurs descriptions sémantiques (en sus de WSDL). Cette approche illustre la possibilité de voir la composition de services Web comme un processus récursif.
- Type matching : [3] décrit un algorithme pour la composition de services Web s'appuyant sur l'appariement partiel des types de messages. Cette approche est montrée comme augmentant de façon considérable le nombre total de compositions réussies. De façon particulièrement intéressante, cette approche de la composition entremêle la composition proprement dite avec la découverte des services satisfaisant les conditions requises, ce qui répond à un certain nombre de problèmes pratiques.
- Coloured Petri nets : [26] illustre concrètement comment des réseaux de Petri colorés peuvent être générés en partant de spécifications BPEL, ce qui permet de gérer de façon précise la compatibilité des "chorégraphies" (on appelle traditionnellement chorégraphie la manière dont un agent (Service Web) interagit avec le monde extérieur)
- Linear logic : [16] propose une application de LL à la composition de services Web. Les auteurs affirment qu'une description WSDL d'un service Web peut être automatiquement transformée en un ensemble d'axiomes LL. Ensuite, ils utilisent un démonstrateur pour le fragment multiplicatif de LL (i.e. sans les disjonctions) pour déduire la composition des services.
- Process solving methods [1] : PSM n'est pas un système formel, mais décrit un modèle des processus pouvant être utilisés pour composer des services Web. Le travail décrit par [8] fournit un cadre pour l'utilisation de PSM dans cette perspective. Les intuitions qui guident le modèle PSM peuvent êtrre rapprochées des expérimentations conduites avec le système de médiation Ariadne [22], un travail qui a inspiré le notre dans une certaine mesure.
- AI Planning : par certains aspects la composition de workflows peut être vue comme un problème de planification. Ce point de vue est celui de [2], où les descriptions des états sont ambiguës et les définitions des opérateurs incomplètes. La mêm approche est choisie par la bibliothèque de composition intercative de services Web SWORD [15] qui génère les plans au moyen d'un langage de règles logiques exploitées en chaînage avant.
- Hierarchical Task Network (HTN) planning : une application de Shop2/BPEL à la composition de services Web est présentée dans [19] and [25].
- Constraint programming : un point de vue origial sur la composition de services Web est défendu dans [7], qui la présente comme un problème d'optimisation discrète non linéaire. Il s'agit d'une approche isolée, autant que nous le sachions.
- Markov decision processes : l'algorithme présenté dans [4] les combine avec de l'apprentissage bayesien de modèles pour générer des workflows robustes au non déterminisme et aux changement de l'environnement.

2 Un métamodèle pour la composition de workflows

Le raisonnement sur les workflows requiert un langage avec suffisamment de généralité pour être viable pratiquement. Puisque nous traitons le problème via la configuration, il est d'autant plus important que le langage de workflow soit modulaire vis à vis de la plu-

part sinon tous les workflow patterns référencés dans [24]. Le langage le plus simple remplissant ces conditions est le langage YAWL [23], pour l'essentiel inclus dans les diagrammes d'activité UML2 [9].

Nous présentons notre modèle objet contraint en accord avec les recommandations standards pour l'architecture MDA (Model Driven Architecture). La prochaine sous-section introduit les différentes classes et leurs associations et attributs via un diagramme de classe. Elle est suivie d'une présentation détaillée des principales contraintes en OCL. Une traduction sous la forme de contraintes JConfigurator est donnée à la fin. Le métamodèle proposé n'est pas complet dans le sens où il ne supporte pas toutes les constructions de workflows (e.g. les exceptions). Encore sous forme de recherche et d'expérimentations, il doit être vu comme une base correcte pour un métamodèle contraint complet.

2.1 Spécifications du métamodèle

2.1.1 Les activités et leurs entrées/sorties

Les activités sont les éléments principaux d'un workflow. Comme défini dans UML2, les activités peuvent avoir un certain nombre de messages en entrées/sorties, qui ont un type défini pris dans une ontologie de données. Toutes les activités ont un "propriétaire" (le workflow originel auquel elle appartient : par exemple, le workflow du Producteur, dans la section 5). Tous les éléments de workflow qui sont ajoutés dynamiquement par le configurateur appartiennent à un propriétaire spécialement introduit : le *Composition Workflow*. Il existe différents types d'activités, comme illustré dans le métamodèle de la figure 2 :

- Noeuds initiaux (initial node) : le point de départ du workflow. Ils n'ont pas d'entrées et sont représentés par un rond noir.
- Noeuds finaux (final node) : un point de sortie possible du workflow. Ils n'ont aucune sortie et sont représentés par un cercle blanc contenant un point noir.
- Noeuds de contrôle (control nodes) : join (synchronisation), fork (lancement concurrent), merge (fin de branchement), decision (branchement conditionnel). Un fork débute des actions en parallèle en dupliquant son unique entrée vers toutes ses sorties. Un join est la construction correspondante pour la synchronisation. Les contrôles décision et merge sont les constructions classiques de type si/alors.
- Actions : activités comprenant une action locale effectuée par le propriétaire du workflow.
- Transformations : activités qui transforment les types de messages sans effet de bord additionnel. Dans le contexte de la composition de services Web, elles sont appelées médiateurs. Les transformations disponibles peuvent être choisis par l'utilisateur du système ou découvertes (e.g dans le cadre du web sémantique).
- Signaux externes (external signals) : une activité qui fournit des messages externes, typiquement les messages transmis par et vers l'utilisateur.

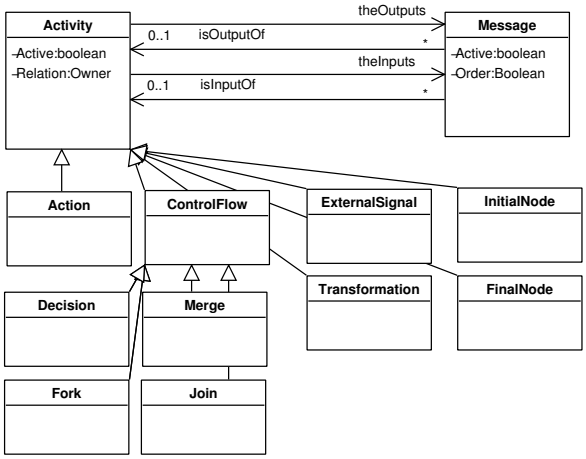

FIG. 2 – Metamodèle pour les activités

2.1.2 Contraintes d'activation

Ces contraintes ne sont pas spécifiques à un problème et peuvent donc être appliquées à toute composition. Sur le sous-ensemble d'UML2 simplifié et légèrement adapté de la figure 2, on observe que les classes *Activity* et *Message* implémentent un attribut booléen appelé “active". Il indique quelles parties du workflow participent effectivement à la composition. En effet, un workflow candidat peut contenir des chemins decision/merge ignorés à l'exécution et dont le composeur sait que les conditions sont impossibles (par exemple, si l'on sait que le type du message connecté provoquera toujours un fail sur le test).

D'où l'introduction des contraintes suivantes :

- Si une action est active, alors toutes ses entrées sont des messages actifs
- Si un message actif est sortie d'un join, toutes ses entrées doivent être actives
- Si un message actif est sortie d'une décision ou d'un fork, son entrée doit être active.
- Si un message actif est sortie d'un merge, au moins une de ses entrées doit être active

Avec ces contraintes, le programme construit des solutions telles qu'au moins un chemin intégralement actif

mène du noeud initial au noeud final (via un message ayant le type du but souhaité). Dans le cas où ce chemin valide traverse un fork ou un join, les autres chemins entrants/sortants doivent être valides également. Si l'utilisateur veut une solution plus robuste (où toutes les branches sont valides), cela peut être obtenu en forçant toutes les parties du workflow à être actives.

2.2 Ontologie des types de messages

Chaque message est relié à un type de donnée, pris dans une ontologie spécifique. Nous utilisons des ontologies prédéfinies pour les séquences d'interaction avec l'utilisateur, et importons celles requises par les services Web sélectionnés. Un exemple d'interaction avec l'utilisateur est l'activité *OfferAcceptance* (un sous-type d'action nouvellement introduit), qui contraint les types de ses entrées/sorties. D'un point de vue global, une activité "OfferAcceptance" fournit un *OfferAnswer* si un *Offer* et un *UserAcknowledgement* lui sont donnés en entrée. Cependant, la correspondance de type *Offer*/*OfferAnswer* est contrainte : elles doivent appartenir au même workflow.

Par exemple, un *ShipperOfferAnswer* ne peut être fourni que si un *ShipperOffer* est en entrée de l'activité d'acceptation d'offre. La figure 3 illustre le fait que ces types appartiennent à la fois à la hiérarchie standard des types et de celle des ontologies des services importés.

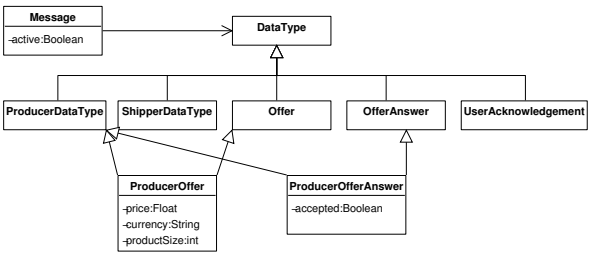

FIG. 3 – Un modèle abstrait pour les ontologies de données

2.3 Semantique

Nous listons ici un certain nombre des contraintes en OCL, assez représentatives pour en dégager l'idée générale.

2.4 Contraintes génériques

Ces contraintes ne sont pas spécifiques au problème et s'appliquent donc à toute composition.

1. Si une action est active, alors toutes ses entrées sont des messages actifs :

   $Context\ Action$
   $inv{:}self.active{=}true\ implies$
   $self.inputs{\rightarrow}forAll(m{:}Message|m.active{=}true)$

2. Si un merge est actif, au moins une de ses entrées doit être active

   $Context\ Merge$
   $inv{:}self.active{=}true\ implies$
   $self.inputs{\rightarrow}exists(m{:}Message|m.active{=}true)$

3. Tous les messages entrant dans un workflow externe sont une sortie du workflow composition :

   $Context\ Message$
   $inv{:}self.isInputOf.Owner{<>}Composition\ implies$
   $self.isOutputOf.Owner{=}Composition$

4. Tous les messages sortant d'un workflow externe sont une entrée du workflow composition :

   $Context\ Message$
   $inv{:}self.isOutputOf.Owner{<>}Composition\ implies$
   $self.isInputOf.Owner{=}Composition$

5. L'ordre d'un message en entrée d'une activité est plus petit que l'ordre de n'importe quel message sortant de cette activité (Cette contrainte d'ordonnancement permet d'éliminer les workflows contenant des boucles non exécutables) :

   $Context\ Message$
   $inv{:}self.outputOf.Outputs{\rightarrow}forAll(m{:}$
   $message|self.Order{<}m.Order))$

2.5 Contraintes prédéfinies

1. Les noeuds fork ont les mêmes types en entrée/sortie :

   $Context\ CompoCopy$
   $inv{:}self.outputs{\rightarrow}forAll(m.OclIsTypeOf(self.input))$

2.6 Contraintes spécifiques au problème

1. Types de messages pouvant être fournis par l'utilisateur :

   $Context\ ExternalSignal$
   $inv{:}self.outputs{\rightarrow}forAll(\ m.theData.OclIsTypeOf(type1)$
   $OR\ m.theData.OclIsTypeOf(type2))$

2. Transformations disponibles : le composeur doit avoir une liste des transformations disponibles. Dans le contexte du web sémantique, cette liste peut être obtenue via une requête à un repertoire de médiateurs.

3. Contraintes de préférences (Policy) : contraintes représentant les volontés de l'utilisateur, afin d'éliminer des compositions non souhaitées (par exemple imposer un prix maximum pour l'offre globale).

2.7 Préférences

Il est possible de spécifier des préférences sur les solutions, ce qui fait du problème un cas d'optimisation combinatoire. Par exemple, on peut préférer les solutions comportant un paiement par carte de crédit, ou dans lesquelles le nombre d'interactions avec l'utilisateur est le plus petit possible.

2.8 D'UML à JConfigurator

Nous donnons ici la traduction de deux des contraintes précédentes vers l'API Java fournie par la librairie JConfigurator, afin de rendre compte de l'adéquation à un langage de configuration.

- Si une activité est "Active", au moins une de ses entrées est "Active" :

```
om.add(om.forAll(A,
  om.eq(A.getIntField("Active"),1),
  om.ge(om.sum(A.getObjectSetField("inputs"),"Active"),1)
));
```

- Pour toute activité, l'ordre maximum des entrées est inférieur à l'ordre minimum des sorties :

```
om.add(om.forAll(A,om.lt(
    om.max(A.getObjectSetField("outputs"),"Order"),
    om.min(A.getObjectSetField("inputs"),"Order")
)));
```

3 Le problème de composition

Un problème de composition de workflows (M_c, W, T, I_t, O_m) peut être paraphrasé ainsi : "Etant donnés un métamodèle contraint M_c, un ensemble W de workflows partiels fournis comme candidats à la composition, un ensemble T de transformations disponibles, un ensemble I_t de messages pouvant être fournis par l'utilisateur, et un *but* sous la forme d'un ensemble O_m de messages souhaités en sortie, produire un workflow composite contenant un sous-ensemble $W' \subseteq W$ des workflows candidats, plus un certain nombre de constructions auxiliaires contenant un sous-ensemble $T' \subseteq T$ et tel que : a/ le workflow composite est *valide* et b/ il produit tous les messages souhaités O_m".

Par valide, nous entendons que le résultat satisfait toutes les contraintes (notamment celles interdisant les situations d'interdépendance). De plus, nous exigeons que chaque workflow participant à la solution communique exclusivement avec le workflow composition. Ce dernier point peut paraître artificiellement complexe car des solutions pourraient être obtenues en reliant directement les workflows entre eux, mais se révèle nécessaire pour des problèmes plus complexes où par exemple des contraintes s'appliquent aux messages échangés entre participants (comme dans [14]), ou plus généralement quand le workflow composite contient des élements pré-existants qui ne peuvent établir de connections directes différentes (comme pour les services Web).

3.1 Instances de workflow

Chaque workflow candidat est défini comme une instance partielle du métamodèle, contenant des activités contraignant le type des messages et des transitions. Ce modèle offre l'avantage de permettre de raisonner sur l'intégrité de la composition complète. Particulièrement, puisque les relations entre entrées et sorties impliquent des contraintes d'ordonnancement temporel, aucune composition ne peut être produite qui viole ces contraintes. En conséquence, les workflows conduisant à une bloucle du processus ne peuvent être générés.

3.2 Transformations

Disposer de transformations qui agissent comme des adaptateurs dans le flux de données est primordial. Ces transformations, appelées médiateurs de données (ou parfois d'ontologies) dans le domaine du web sémantique, sont nécessaires puisque des services Web ou processus indépendants peuvent définir des types de messages compatibles mais pas strictement identiques. Des contraintes spécifient les entrées possibles en fonction des sorties, de telle façon qu'il soit possible de créer des transformations abstraites disponibles avant le début de la recherche (e.g. un extracteur de données).

3.3 La requête au configurateur

Nous voulons résoudre le problème de composition de workflows (M_c, W, T, I_t, O_m) avec des techniques de configuration. Ceci peut être vu comme le processus consistant à connecter le but O_m à certains messages fournis de type I_t, via un certain nombre de workflows intermédiaires, de contrôles de flux et de transformations.

Le choix d'un configurateur spécifique (ici ILOG JConfigurator) influence clairement l'implémentation du problème en une requête de composition. Une des

spécificités de JConfigurator est l'absence de la création par nécessité. Dans un processus de configuration, lorsque la cardinalité minimum d'une relation ne peut être satisfaite par les objets existants, la création automatique des instances nécessaires peut être déclenchée. Pour pallier à cette limitation, toutes les instances pouvant participer à une solution doivent être fournies en *nombre suffisant*. Cette difficulté est cependant minimisée car JConfigurator implémente un raisonnement de classification[5] : des instances peuvent être créees pour des types abstraits, qui seront classifiés par la suite. Cela évite d'avoir à préparer des instances pour toutes les feuilles de la hiérarchie de types.

Parmi les objets qui doivent être créés avant le début de la recherche figure le workflow propriétaire "composition". Il prépare les interactions avec tous les workflows candidats grâce à un nombre suffisant de transitions (ou contrôles de flux : fork, join, split, merge) ainsi que des transformations et workflows partiels représentant des schémas d'interaction avec l'utilisateur. Les éléments de ce groupe seront ensuite disponibles pour être sélectionnés par le processus de configuration.

La requête de composition contient donc les informations suivantes :

- le workflow propriétaire composition,
- un noeud initial,
- un ou plusieurs noeuds finaux,
- le "goal" sous la forme de types de messages souhaités en sortie (et connectés au noeud final),
- les informations pouvant être fournies par l'utilisateur (e.g. accepter une offre, donner une numéro de CB) sont donnés comme des types possibles de sortie des signaux externes,
- un certain nombre de séquences d'interaction avec l'utilisateur,
- un certain nombre de transitions nécessaires pour synchroniser les flux entrant/sortant de la composition.

4 Le processus de configuration

Nous utilisons une procédure de configuration récursive et orientée objet. La configuration est orientée but : elle procède en partant du noeud final, et tente de connecter ses messages entrants sur les sorties de workflows partiels fournis en argument, ou, si impossible, sur des transformations introduites exprès. Une fois cette connexion réalisée, cela conduit récursivement à de nouveaux besoins de rattachements, jusqu'à pouvoir fermer la configuration en atteignant le noeud initial, qui n'a pas de message entrant. Cette approche ressemble mais seulementpartiellement au chaînage arrière, car le processus de configuration peut parcourir des chemins divers en fonction des heuristiques. Parmi les choix d'énumération essentiels figure celui de configurer récursivement immédiatement un objet nouvellement connecté, ou de déférer cette décision.

Le modèle objet contraint que nous utilisons n'exclut pas l'utilisation d'autres stratégies, y compris le chaînage avant - qui débuterait en ancrant la configuration sur les messages en entrée - ou un mélange des deux. Il doit être noté que bien que si le but (un ou plusieurs messages attendus) doit être connecté, certaines des entrées (de l'utilisateur) peuvent rester inutilisées. Par exemple, seule une fraction des méthodes de paiement possibles pourra être utile dans une composition incluant une transaction en ligne. Cette situation disqualifie normalement les approches "chaînage avant" guidées par les données en entrée.

5 Résultats expérimentaux

Nous rapportons ici les résultats obtenus sur deux problèmes de la littérature, choisis à cause de l'étendue des difficultés qu'ils soulèvent. Les expérimentations ont été conduites sur un Pentium mobile à 2,8 Mhz, ayant 1 GO de Ram, et utilisant la Java JVM 1.5 et ILOG JConfigurator 2.1. Les temps de calcul sont exprimés en secondes. Le nombre de points de choix est celui requis pour atteindre une solution, et le nombre d'échecs compte les backtracks effectivement réalisés (un nombre toujours inférieur au précédent).

5.1 Exemple : Cinéma et Film

Ce problème est issu de [8], et réalise l'exécution automatique des tâches suivantes : trouver un cinéma, puis les horaires d'une film donné, puis acheter un billet pour la séance choisie, du film, dans le cinéma. Le workflow résultant de la composition de ces trois services élémentaires est illustré par la figure 4.

Dans la figure 4, nous observons que toutes les entrées utilisateur sont dirigées vers un noeud de synchronisation spécifique. Il en est ainsi car comme expliqué précédemment, tous les messages sont échangés via le workflow de composition, qui peut rediriger et/ou dupliquer les messages utilisateur vers les services qui le requièrent. La solution ainsi présentée ne demande pas la mise en oeuvre de transformations, mais les données du problème pourraient être aisément changées de façon réaliste pour le nécessiter.

Il faut noter que des noeuds "fork" sont requis pour dupliquer les jetons message vers divers points de

[5] JConfigurator révise dynamiquement le type des objets pour satisfaire les contraintes de type survenant lors de la recherche

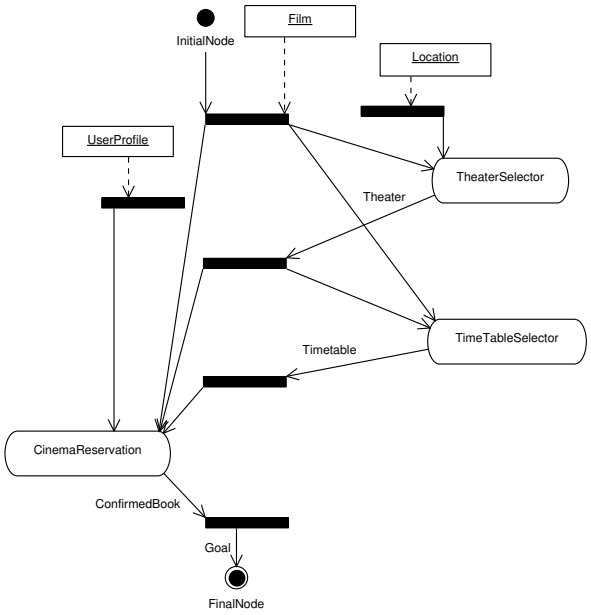

FIG. 4 – Le workflow composite Cinéma et Film

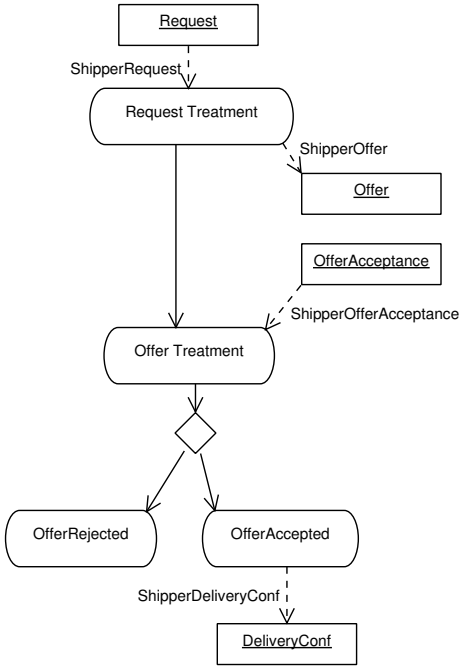

FIG. 5 – Le workflow partiel du transporteur

la composition. Cette considération n'est qu'implicite dans [8] mais demande une prise en compte rigoureuse.

5.2 Exemple : Producteur and Transporteur

Ce problème est issu de [14], un article appliquant des techniques de planification à la composition de services. Nous l'avons choisi à cause de ses propriétés intéressantes :

- le transporteur (shipper) et le producteur (producer) font chacun une offre sur la base de la requête de l'utilisateur et des résultats intermédiaires du producteur. Ces deux offres doivent être agrégées pour les présenter à l'utilisateur, qui peut les accepter ou les refuser.
- le producteur et le transporteur sont spécifiés sous la forme de workflows partiels assez complexes, et pas seulement par des activités isolées,
- les deux workflows partiels ne peuvent pas être exécutés l'un après l'autre, mais sont entremêlés, car chacun doit attendre que l'autre ait reçu un OfferAcceptance avant de clore la transaction,
- le workflow du transporteur demande une taille en entrée, qui ne peut être obtenue que par extraction (e.g. transformation) à partir de l'offre du producteur,
- finalement, le but est décomposé en deux sous buts : les confirmations du producteur et du transporteur.

La figure 5 illustre le workflow partiel du transporteur, comme défini avant que la composition ne débute. Le workflow du producteur est similaire, modulo les ontologies de types de messages.

Le résultat de la composition est présenté par la figure 6. Ce workflow composite intègre synchronisation, exécution entremêlée, des transformations (les boites ovales) et il devrait être noté que certains chemins d'exécution sont ignorés : en effet, si l'utilisateur rejette l'offre, le but ne peut pas être réalisé. Cela illustre pourquoi nous avons besoin d'un attribut caractérisant les chemins actifs dans le métamodèle.

5.3 Résultats

Nous avons réalisé des expériences sur les deux exemples précédemment détaillés. Les résultats d'exécution pour les deux problèmes sont décrits par la table 1. Nous n'avons utilisé aucune heuristique particulière pour ces calculs, laissant le système fonctionner par défaut. Ainsi que les résultats le montrent, l'important ratio échecs/points de choix dans le cas pro-

TAB. 1 – Résultats expérimentaux (temps en secondes)

Problème	temps	points de choix	échecs
Cinéma	0.46	13	0
Producteur	20.06	103151	103109

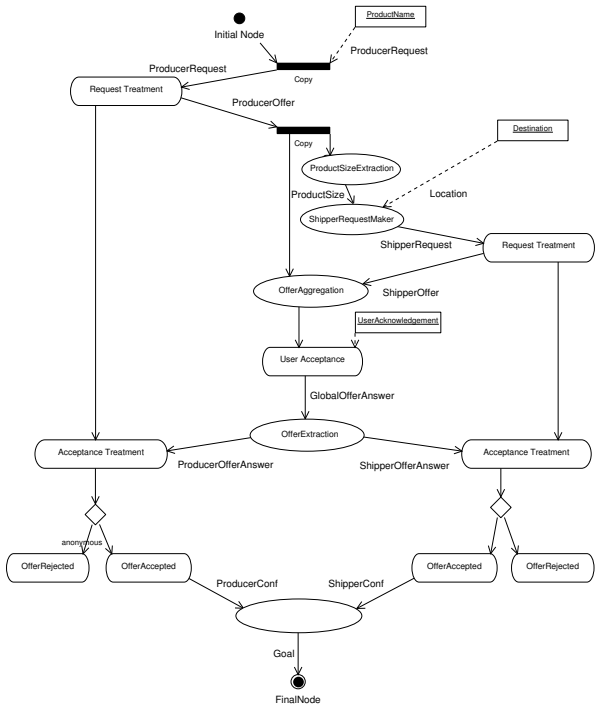

FIG. 6 – Le workflow composite

ducteur indiquent que le programme n'a pu trouver la solution rapidement. Cela suggère que des heuristiques appropriées devraient améliorer ces statistiques. Il doit être noté que dans [14], le workflow de l'utilisateur est fourni en entrée au programme (ce n'est pas le cas ici) et que le temps de calcul du résultat est de 18000 secondes[6].

6 Conclusion

Cette recherche montre la faisabilité de l'utilisation de la configuration pour réaliser la composition automatique de workflows, qui possède des applications pratiques à de nombreux domaines, par exemple la compositions de sevices web ou de processus métier. Cette possibilité établit clairement la composition de workflows comme un cas particulier de recherche de modèles finis.

La configuration requiert un modèle objet pour s'effectuer, ce qui place la conception de telles applications dans un champ familier à de nombreux ingénieurs. Une part essentielle du modèle objet considéré, le métamodèle des diagrammes d'activité, existe par ailleurs dèjà comme un sous ensemble de la méthode de spécification UML2.

[6]Les conditions expérimentales diffèrent toutefois significativement dans les deux cas, ce qui interdit une comparaison utile des temps de calcul

La configuration place aussi ce problème dans le paradigme de la programmation par contraintes. Ce faisant, la prise en compte de préférences sur les solutions en fait un problème d'optimisation ou un problème de recherche guidée par les préférences.

Etant combinatoire par nature (un workflow est un graphe), le problème que nous adressons est difficile. Les techniques de configuration peuvent aider à réduire la complexité par :

- l'utilisation d'abstractions, grâce au raisonnement sur les types,
- des structures de données appropriées (puisque des portions importantes des workflows partiels sont invariantes, elles pourraient être implantées au moyen de structures de données spécifiques plus légères que dans la version actuelle),
- la recherche locale,
- des techniques de rejet d'isomorphismes ...

Toutes ces options relèvent de recherches actives ou futures. Nous travaillons actuellement à la montée en charge, en augmentant le nombre de workflows inutilisés.

Remerciements

Les auteurs remercient le projet Européen DIP et la société ILOG pour leur support financier lors de cette recherche.

Références

[1] V.R. Benjamins and D. Fensel. *Special Issue on Problem-Solving Methods. International Journal of Human-Computer Studies (IJHCS)*, 49(4) :305–313, 1998.

[2] M. Carman, L. Serafini, and P. Traverso. Web service composition as planning. In *proceedings of ICAPS03 International Conference on Automated Planning and Scheduling*, Trento, Italy, June 9-13 2003.

[3] I. Constantinescu, B. Faltings, and W. Binder. Large scale, type-compatible service composition. In *proceedings of IEEE International Conference on Web Services (ICWS 2004)*, San Diego, USA, 2004.

[4] Prashant Doshi, Richard Goodwin, Rama Akkiraju, and Kunal Verma. Dynamic workflow composition using markov decision processes. *International Journal of Web Services Research*, 2(1) :1–17, Jan-March 2005.

[5] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Configuration

knowledge representation using uml/ocl. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 49–62. Springer-Verlag, 2002.

[6] G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring large-scale systems with generative constraint satisfaction. *IEEE Intelligent Systems - Special issue on Configuration*, 13(7), 1998.

[7] R. Ginis and K.M. Chandy. Service composition issues for distributed business processes. In *proceedings of The 2003 International Conference on Web Services (ICWS'03)*, pages 27–33, Las Vegas, Nevada, USA, June 23 - 26 2003.

[8] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lamaa. A framework for design and composition of semantic web services. In *Semantic Web Services, 2004 AAAI Spring Symposium Series*, 22nd-24th March 2004.

[9] Object Management Group. *UML v. 2.0 specification.* OMG, 2003.

[10] D. Mailharro. A classification and constraint based framework for configuration. *AI-EDAM : Special issue on Configuration*, 12(4) :383 – 397, 1998.

[11] S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *proceedings of Conference on Knowledge Representation and Reasoning*, April 2002.

[12] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, pages 25–32, 1990.

[13] B. Nebel. Reasoning and revision in hybrid representation systems. *Lecture Notes in Artificial Intelligence*, 422, 1990.

[14] Marco Pistore, F. Barbon, Piergiorgio Bertoli, D. Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In *proceedings of the Workshop on Planning and Scheduling for Web and Grid Services held in conjunction with ICAPS 2004*, Whistler, British Columbia, Canada, June 3-7 2004.

[15] S.R. Ponnekanti and A. Fox. Sword : A developer toolkit for web service composition. In *proceedings of the 11th International WWW Conference*, page to appear, Hawaii, May 7-12 2002.

[16] J. Rao, P. Kungas, and M. Matskin. Logic-based web service composition : from service description to process model. In *proceedings of the 2004 IEEE International Conference on Web Services, ICWS 2004*, San Diego, California, USA, July 6-9 2004.

[17] R. Reiter. *Knowledge in Action : Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[18] E. Sirin, J. Hendler, and B. Parsia. Semi automatic composition of web services using semantic descriptions. In *proceedings of the ICEIS-2003 Workshop on Web Services : Modeling, Architecture and Infrastructure*, Angers, France, April 2003.

[19] Evren Sirin, Bijan Parsia, Dan Wu, James Hendler, and Dana Nau. HTN planning for web service composition using SHOP2. *Journal of Web Semantics*, 1(4) :377–396, 2004.

[20] T. Soininen, I. Niemelõ, J. Tiihonen, and R. Sulonen. Unified configuration knowledge representation using weight constraint rules. In *ECAI 2000 Configuration Workshop*, 2000.

[21] Markus Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2) :111–125, June 1997.

[22] S. Thakkar, C.A. Knoblock, J.L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In *proceedings of AAAI-02 Workshop on Intelligent Service Integration*, Edmondon, Canada, July 2002.

[23] W.M.P. van der Aalst, L. Aldred, and M. Dumas. Design and implementation of the yawl system. qut technical report, fit-tr-2003-07. Technical report, Queensland University of Technology, Brisbane, 2003.

[24] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, pages 5–51, July 2003.

[25] M. Vukovic and P. Robinson. Adaptive, planning based, web service composition for context awareness. In *proceedings of the Second International Conference on Pervasive Computing*, page to appear, Vienna, Austria, 2004.

[26] X. Yi and K. Kochut. A cp-nets-based design and verification framework for web services composition. In *proceedings of 2004 IEEE International Conference on Web Services, July 2004*, San Diego, California, USA, July 6-9 2004.