Language de Requête Configurable pour la Composition de Services Web Semantiques

Laurent Henocque, Mathias Kleiner

Language de Requête Configurable pour la Composition de Services Web Semantiques

Article jeune chercheur

Patrick Albert, Laurent Henocque et Mathias Kleiner

Laboratoire LSIS, Université de Saint-Jérôme,
Avenue Escadrille Normandie-Niemen, 13397 Marseille, France
ILOG S.A, 9 rue de Verdun, 94253 Gentilly, France
palbert@ilog.fr, laurent.henocque@lsis.org, mkleiner@ilog.fr

Résumé

La composition est un des principaux challenge pour la communauté des services web sémantiques (SWS). Parmi les approches existantes, il a été montré efficace d'utiliser des techniques à base de contraintes (telle que la configuration) pour créer des orchestrations à partir des choréographies. Des expérimentations supplémentaires ont révélé les limitations et les ambiguités sémantiques qui peuvent survenir à partir de la requête à un composeur. Cet article propose une approche originale où la requête est vue comme un problème en lui-même, et montre comment la configuration peut être utilisée pour le résoudre grâce à un modèle objet contraint.

Abstract

Composition is a main challenge for the Semantic Web Services (SWS) community. Among existing approaches, it was shown efficient to use constraint based technics (as configuration) to create orchestrations out of choreographies. Further experiments have revealed issues on the semantic ambiguities and limitations that could raise from the requests made to a composer. The present paper presents an original approach where the request is seen as a problem in itself, and shows how configuration can be used to solve it using a given constrained object model.

1 Introduction

1.1 Contexte

La composition est définie comme "l'acte de combiner et coordonner un ensemble de services web sémantiques (SWS)". Les services web sémantiques sont communément définis par :

- une capabilité : décrit ce qui est réalisé par un SWS et ce qu'il requiert en terme de messages d'entrée(s)/sortie(s) (Les types de messages sont définis grâce à des concepts pris dans une ontologie spécifique).
- une choréographie : décrit comment intéragir avec le service (protocole d'échange des messages)
- une orchestration : dans le cas d'un web service composite, décrit de quelle manière il utilise et communique avec les SWS externes pour réaliser sa capabilité.

La composition automatique de workflow ou de services web sémantiques est un domaine d'intense activité, avec des applications à au moins deux grands pôles : la modélisation de processus métiers et les services webs (sémantiques). Des tentatives pour résoudre cès problèmes ont été expérimentées à travers plusieurs formalismes et techiniques, parmi lesquels : le "situation calculus" [16], la programmation logique [21], le "type matching" [6, 5], les réseaux de Petri colorés [7], la logique linéaire [20], les méthodes de résolution de problèmes [3, 11, 25], la planification [4, 19], les processus de décision de Markov [8]. Les travaux présentés ici poursuivent une approche à base de contraintes formulée dans [1, 2] où il a été prouvé possible et efficace de composer des choréographies à l'aide de la configuration. Nous donnons dans cet article une technique élaborée pour résoudre les ambiguïtés sémantiques qui

pouvaient survenir. Ce processus prend sa place durant la définition de la requête de l'utilisateur, avant de créer l'orchestration. Nous introduisons tout d'abord quelques notions qui seront utilisées tout au long du document.

1.2 Brève introduction à la configuration

Une tâche de configuration consiste à fabriquer (une simulation d') un *produit complexe* à partir de *composants* choisis dans un catalogue de *types*. Ni le nombre ni le type des composants requis n'est connu à l'avance. Les composants sont sujets à des *relations*, et leurs types à des relations d'*héritage*. Des *Contraintes* définissent tous les produits valides. Un configurateur prend en entrée un fragment de la structure de l'objet cible, et l'étend à une solution du problème de configuration (si elle existe), en ajoutant tous les éléments nécessaires durant la recherche. Ce problème est semi-décidable dans le cas général. Le lecteur peut se référer à [15] pour une introduction plus poussée à la configuration. Un programme de configuration peut être décrit grâce à un *modèle objet contraint* sous la forme d'un diagramme de classes standard, accompagné de contraintes ou règles de bonne formation. Il est possible de résoudre le problème d'énumération associé à travers plusieurs formalismes ou approches techniques : extensions du paradigme CSP [17, 10], approches basées sur la connaissance [24], logiques terminologiques [18], programmation logique (chainage avant ou arrière et sémantiques non standard) [22], approches orientées objet [15, 24].

1.3 Abstractions des SWS : les goals

Un utilisateur (ou un outil de composition) à la recherche d'un web service sémantique particulier exprime des conditions que le service à découvrir doit remplir. Nous appelons l'ensemble de ces conditions un *goal atomique*. Un goal est une abstraction de SWS dans le sens où un certain nombre de services peuvent convenir. Définir un goal équivaut à exprimer une requête pour récupérer les services enregistrés dans un répertoire (un processus appelé *discovery*). Un goal atomique se présente donc de façon similaire à une capabilité, spécifiant ce que l'on souhaite réaliser en sortie et ce que l'on peut fournir en entrée. Par la suite, nous appelerons *input* et *output roles* l'abstration des messages en entrée/sortie des SWS. La figure 1 illustre un goal atomique pour un service d'expédition, dans lequel on spécifie une ville de destination que le service devra couvrir.

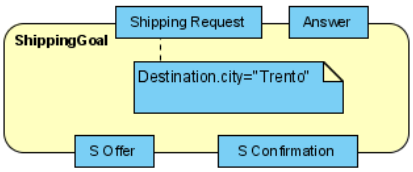

Figure 1: Goal atomique pour un service d'expédition

2 Buts et exigences d'un langage de requête pour la composition de SWS

Il a été montré dans [1, 2] qu'il est possible de composer efficacement des choréographies de SWS en utilisant un configurateur et un sous-ensemble des diagrammes d'activité UML2 comme modèle de workflows. Dans cette spécification, le but de la composition est un message unique : la sortie souhaitée du web service composé. La requête inclut également un certain nombre de messages que l'utilisateur peut fournir en entrée. Cependant, une des principales restrictions de cette méthode est qu'un composeur a souvent besoin de requêtes plus précises pour créer directement le service souhaité, et ainsi éviter de nombreuses solutions valides mais non réalistes. Pour illustrer ce problème, prenons l'exemple d'une agence de voyage virtuelle qui utilise des services web externes pour réserver les hotels et les trajets. Un voyage classique peut être constitué d'un vol aller, d'un hotel dans la ville d'arrivée, et d'un vol retour. Les web services externes peuvent prendre en entrée un message "ville" indiquant l'emplacement de l'hotel ou (pour les trajets) les villes de départ et d'arrivée. Si on laisse le composeur calculer toutes les solutions possibles, nous obtiendrons certains workflows où la ville de départ est utilisée comme emplacement de l'hotel, ce qui n'est vraisemblablement pas le comportement souhaité. De telles ambiguïtés suggèrent l'utilisation d'un langage de requête précis afin d'être résolues efficacement. Nous appelerons ces instances de requêtes complexes des *goals de composition*. Bien que de nombreux articles traitent de la composition de SWS, le seul qui aborde la question de requêtes complexes est [19], où le langage EaGLe permet de définir des alternatives et des règles temporelles. Dans cette section, nous allons essayer de lister tous les éléments requis pour un langage de requête de composition.

2.1 Flexibilité

Une originalité de ce qui est présenté ici est que nous considérons la requête comme un problème à résoudre en elle-même. Bien que nous nécessitions un langage

complexe permettant à l'utilisateur d'entrer précisément les détails de la composition, nous ne devons pas écarter la possibilité de traiter des requêtes simples. Par flexibilité nous entendons donc qu'il doit être possible d'exprimer des requêtes simples aussi bien que des goals de composition déjà complets. Cela autorise des utilisateurs "finaux" (comme des particuliers utilisant un service de composition sur le web et "à la volée"), qui ne possèdent pas les compétences nécessaires pour manipuler des constructions complexes, de formuler une requête minimale et laisser l'outil l'étendre à des goals de composition possibles parmi lesquels il pourra choisir le comportement souhaité. Cependant, cela autorise également des utilisateurs "moyens" (comme des industriels réalisant leur propre web service composite) à spécifier précisément ce qu'ils en attendent.

2.2 Abstraction des détails techniques

Une condition importante pour un langage de requête de composition est de rester au niveau le plus abstrait possible, comme le niveau des goals présenté en sous-section 1.3. Cela permet aux utilisateurs de travailler uniquement sur les propriétés et les relations sémantiques de leur web service composé, en ignorant les détails techniques des choréographies. Cela offre également l'avantage de diminuer l'espace de recherche durant la phase de composition des workflows puisque seuls les SWS qui répondent aux goals atomiques seront pris en considération.

2.3 Liaison sémantique des messages

Afin de lever les ambiguïtés sémantiques discutées en introduction de cette section, le langage doit fournir un moyen de lier des messages entre eux. Cependant cette liaison sémantique ne doit pas imposer au workflow final de placer directement un flux de données entre ces messages, mais plutôt indiquer quel rôle ils jouent dans la composition. Par exemple, lier une arrivée d'un vol à une ville d'hotel ne doit pas empêcher le workflow composé d'utiliser cette ville d'arrivée pour d'autres constructions telles que des opérations, des duplications, ou encore des choix pour des chemins alternatifs.

2.4 Restrictions sur les objets

Les utilisateurs ont souvent besoin de placer des contraintes sur les valeurs des objets présents dans la composition : les messages qu'ils attendent et fournissent ainsi que les propriétés des web services composites. Nous pouvons identifier trois principaux types de restrictions :

- Contraintes sur un seul message : e.g le prix attendu doit être inférieur à une certaine valeur.
- Contraintes entre messages : e.g le prix doit être inférieur à une valeur donnée par le rôle "budget" en entrée
- Contraintes sur les propriétés non-fonctionnelles d'un service web : e.g imposer un fournisseur particulier sur les web services à envisager.

2.5 Chemins alternatifs

Des chemins alternatifs doivent pouvoir être calculés automatiquement par le composeur. Si l'utilisateur requiert d'un web service composé la possibilité d'acheter des disques ou des livres, et les seuls web services disponibles proposent uniquement l'un ou l'autre, des chemins alternatifs doivent être crées pour gérer chacun des cas selon une valeur donnée par l'utilisateur en entrée. Cependant, il doit également être permis d'imposer une alternative même si cela n'est pas nécéssaire pour obtenir les sorties désirées. Dans l'exemple de l'agence de voyage, on pourrait choisir différents moyens de transport selon les destinations entrées.

2.6 Annulation et règles de compensation

Dans [1], le résultat du composeur est un workflow dans lequel au moins un chemin mène aux sorties désirées. En conséquence, certains chemins peuvent nécessiter l'annulation des actions effectuées jusqu'à présent (compensation). Un langage de requête peut définir les comportements souhaités dans ces cas, et indiquer des règles permettant de traiter ces exceptions dans des chemins alternatifs.

3 Un modèle objet contraint

A partir des pré-requis décrits précédemment, nous introduisons un langage de requête pour la composition de SWS pouvant être résolu comme un problème en lui-même. La nécessité d'ajouter dynamiquement des éléments au cours de la recherche et les constructions proches des contraintes font de la configuration un choix naturel et approprié afin de le résoudre. Un configurateur utilise un modèle objet contraint pour lequel il recherches des instances valides à partir d'un objet racine. Nous présentons dans les figures 2 et 3 un modèle de goals de composition sous la forme d'un modèle abstrait UML accompagné de contraintes Z spécificiant les règles de bonne formation. Nous utilisons un sous-ensemble du langage Z [23, 13] pour

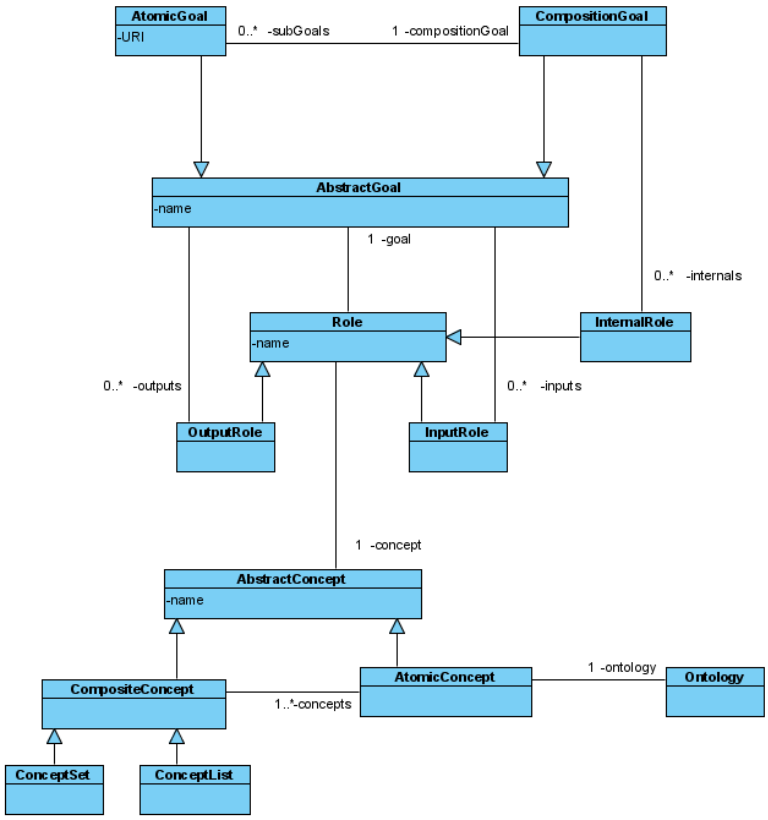

Figure 2: Goals, roles et concepts

documenter les restrictions sur le diagramme. Z permet une spécification totalement formelle et sans ambiguïtés en remplacement du langage OCL, moins expressif. Nous donnons également la sémantique des constructions utilisées indiquant comment elles seront traitées dans la phase de composition des workflow.

Semantique

- Atomic goals : abstractions des SWS. Le processus de discovery permet de récupérer les SWS disponibles pour la composition du workflow final.
- Roles : Entrées et sorties. Les roles internes peuvent être utilisés comme objets intermédiaires dans l'orchestration. Comme les messages, chaque rôle a pour type un concept pris dans une ontologie.
- Value Constraints : le workflow solution respecte les contraintes posées sur les valeurs.
  - Unary Value Constraints : dans le workflow final, les *tokens* traversant l'objet désigné respectent la (ou le champ de) valeur(s) donnée(s).
  - Relational Value Constraints: dans le worklow final, les tokens respectent les contraintes données entre plusieurs objets.
- Dataflow Constraints: le workflow final contient un flux de données spécifique entre sources et cibles :
  - IdentityFlow: le flux entre la source et les cibles ne modifie pas les données. Les concepts source et cibles doivent être identiques.
  - OperationFlow: le flux réalise une opération sur les tokens sources pour obtenir les cibles. Applicable uniquement aux entiers et décimaux.
  - MediationFlow: le flux entre sources et cibles utilise un médiateur spécifique (Les médiateurs permettent de passer d'une ontologie à une autre).
  - AggregationFlow: le flux aggrège les sources pour former la cible (un concept composite).
  - ExtractionFlow: le flux extrait des parties du concept composite source pour former les cibles.

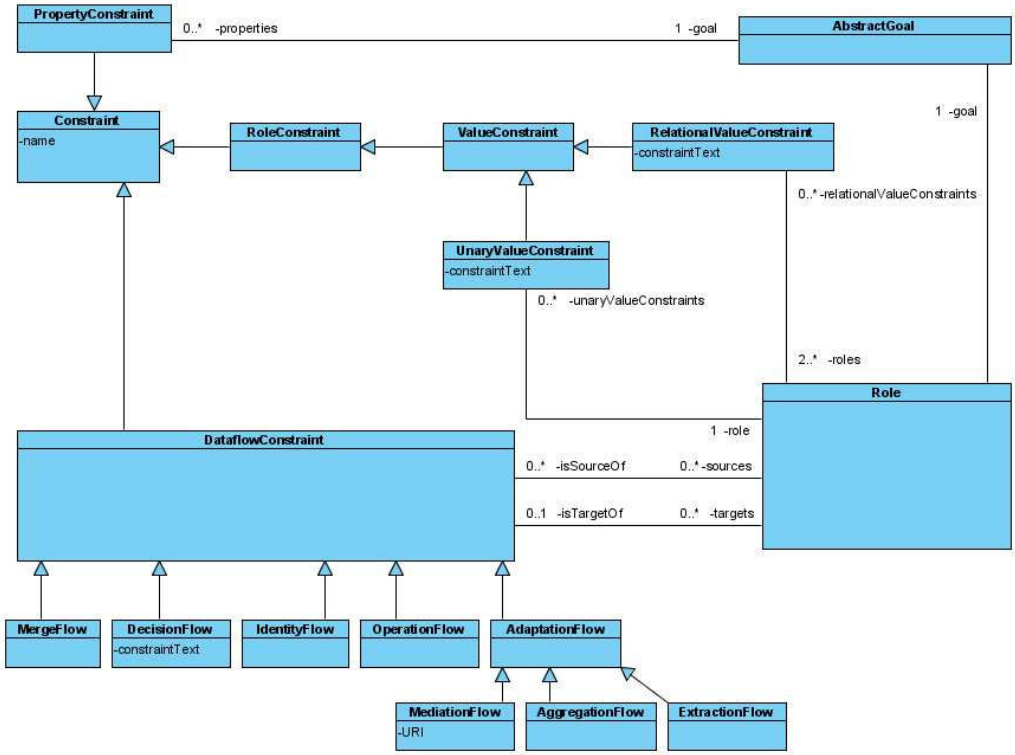

Figure 3: Contraintes sur les Goals et les Roles

- DecisionFlow: le flux offre des alternatives selon la contrainte donnée (contraintText). Les concepts sources et cibles doivent être identiques.
- MergeFlow: le flux accepte n'importe quelle entrée pour former la cible. Les concepts sources et cibles doivent être identiques.

Contraintes

- Relation de réciprocité entre Goals et Roles:

$$\forall\, n : AbstractGoal \bullet \\ inputs(n) = \{e : Role \mid e.goal = n\} \\ \forall\, n : AbstractGoal \bullet \\ outputs(n) = \{e : Role \mid e.goal = n\} \\ \forall\, n : AbstractGoal \bullet \\ internals(n) = \{e : Role \mid e.goal = n\}$$

- Les sources de flux de données doivent être de classe OutputRole ou InternalRole:

$$\forall\, n : Role \mid \#(n.isSourceOf) > 1 \bullet \\ n \in OutputRole \lor \\ n \in InternalRole$$

- Les cibles de fux de données doivent être de classe InputRole ou InternalRole:

$$\forall\, n : Role \mid \#(n.isTargetOf) > 1 \bullet \\ n \in InputRole \lor \\ n \in InternalRole$$

- Les sources et cibles des IdentityFlow, DecisionFlow et MergeFlow doivent partager le même concept:

$$\forall\, n : DataflowConstraint \mid (n \in IdentityFlow \\ \lor\ n \in MergeFlow \lor n \in DecisionFlow) \bullet \\ sources(n) \rightarrow concept \\ = targets(n) \rightarrow concept$$

- DecisionFlow a une unique source:

$$\forall\, n : DecisionFlow \bullet \\ \#(n.sources) = 1$$

- MergeFlow a une unique cible:

$$\forall\, n : MergeFlow \bullet \\ \#(n.targets) = 1$$

- ExtractionFlow a une unique source dont le concept est composite, et les cibles sont des concepts atomiques inclus dans la source:

$$\begin{aligned}&\forall\, n : \mathit{ExtractionFlow} \bullet \\ &\#(n.\mathit{sources}) = 1 \\ &\wedge\ \mathit{sources}(n) \rightarrow \mathit{concept} \in \mathit{CompositeConcept} \\ &\wedge\ \mathit{targets}(n) \rightarrow \mathit{concept} \in \mathit{AtomicConcept} \\ &\wedge\ \mathit{targets}(n) \rightarrow \mathit{concept} \\ &\qquad \subset \mathit{sources}(n) \rightarrow \mathit{concept} \rightarrow \mathit{concepts}\end{aligned}$$

- AggregationFlow a une unique cible dont le concept est composite, et les sources sont des concepts atomiques inclus dans la cible:

$$\begin{aligned}&\forall\, n : \mathit{AggregationFlow} \bullet \\ &\#(n.\mathit{targets}) = 1 \\ &\wedge\ \mathit{targets}(n) \rightarrow \mathit{concept} \in \mathit{CompositeConcept} \\ &\wedge\ \mathit{sources}(n) \rightarrow \mathit{concept} \in \mathit{AtomicConcept} \\ &\wedge\ \mathit{sources}(n) \rightarrow \mathit{concept} \\ &\qquad \subset \mathit{targets}(n) \rightarrow \mathit{concept} \rightarrow \mathit{concepts}\end{aligned}$$

3.0.1 Représentation graphique

Nous fournissons une représentation graphique basée sur les diagrammes d'activité UML2 afin de pouvoir dessiner et présenter des goals de composition de façon claire. La figure 4 liste toutes les constructions et leur notation graphique. Nous utiliserons cette notation pour les exemples de la section suivante.

4 Vue d'ensemble du processus de composition

Nous pouvons désormais identifier un processus complet pour une composition efficace des services web sémantiques. Dans une première étape, le composeur reçoit une requête qui nécessite d'être étendue à un goal de composition complet et valide, d'après le modèle abstrait présenté dans la section précédente. Nous donnons un exemple de requête et une de ses extensions (solutions) dans les figures 5 et 6. Ce processus de configuration recherche une instance valide du modèle abstrait contraint, en utilisant les sorties souhaitées comme objets racines (goal-oriented). De plus, le composeur peut utiliser les éléments suivants :

- une librairie de goals atomiques
- une librairie de médiateurs
- une librairie d'opérations

Des requêtes de discovery sont ensuite crées sur la base des goals atomiques présents dans le goal de composition final afin de récupérer les SWS disponibles et leurs choréographies.
Enfin, le composeur peut assembler ces workflows comme montré dans [1], en respectant maintenant les contraintes additionnelles et objectis du goal de composition.

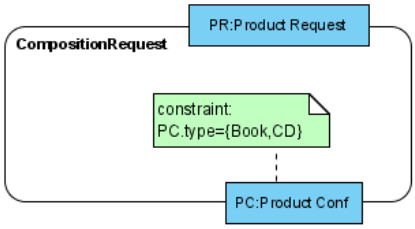

Figure 5: Exemple graphique de requête (goal de composition partiel)

5 Conclusion

La composition de SWS est un des problèmes les plus difficiles du web sémantique. Nous avons pointé la nécessité d'un langage de requête flexible et donné une liste d'exigences pour celui-ci. Nous avons par conséquent introduit une nouvelle approche de la composition en deux étapes, où les ambiguïtés sémantiques sont résolues dans une première partie. A cette fin, nous avons également présenté un modèle abstrait permettant d'étendre les requêtes grâce à un outil de configuration. Les recherches à venir incluront des expérimentations sur ce processus à deux phases afin de le comparer à ce qui avait été obtenu précédemment. De nombreux problèmes liés à la composition devront être également résolus dans l'avenir, parmi lesquels : la scalabilité (le nombre élevé de SWS disponibles sur le web pourrait requérir l'ajout de techniques de recherche locale), la fiabilité des services web composés (temps d'accès, disponibilité), la compensation (prévoir les problèmes d'éxécution et créer les chemins alternatifs nécessaires).

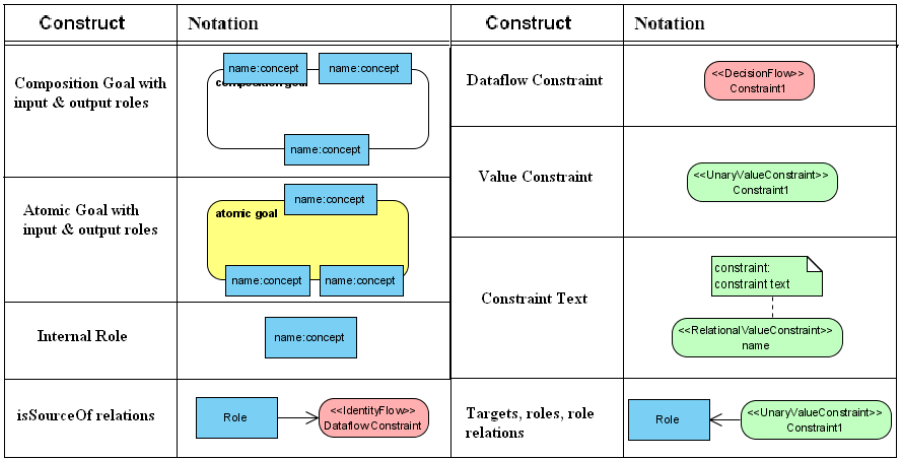

Figure 4: Représentation graphique pour les goals de composition

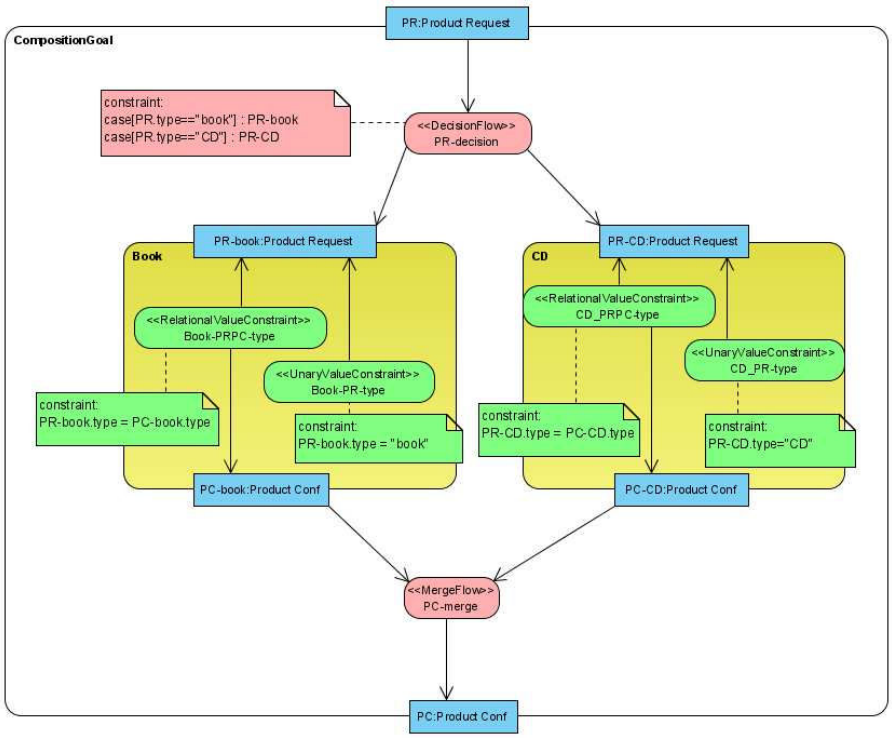

Figure 6: Exemple graphique d'un goal de composition complet et valide

References

[1] Patrick Albert, Laurent Henocque, and Mathias Kleiner. Configuration based workflow composition. In *proceedings of International Conference on Web Services ICWS'05*, pages 285–292, Orlando, Florida, USA, 2005.

[2] Patrick Albert, Laurent Henocque, and Mathias Kleiner. A constrained object model for configuration based workflow composition. In *Revised Selected papers of the Third International Conference on Business Process Management Workshops BPM'05-WSCOBPM*, pages 102–115, Nancy, France, september 2006.

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

[4] 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.

[5] Ion Constantinescu, Walter Binder, and Boi Faltings. Flexible and efficient matchmaking and ranking in service directories. In *2005 IEEE International Conference on Web Services (ICWS 2005)*, pages 5–12, Florida, USA, July 2005. IEEE Computer Society.

[6] Ion Constantinescu, Boi Faltings, and Walter Binder. Large scale, type-compatible service composition. In *IEEE International Conference on Web Services (ICWS 2004)*, pages 506–513, San Diego, USA, July 2004. IEEE Computer Society.

[7] R. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach, ctit technical report series no. 04-09. Technical report, Centre for Telematics and Information Technology, University of Twente, The Netherlands, February 2004.

[8] 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.

[9] 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.

[10] 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.

[11] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. A framework for design and composition of semantic web services. In *Semantic Web Services, 2004 AAAI Spring Symposium Series*, pages 113–120, 22nd-24th March 2004.

[12] Object Management Group. Uml 2 superstructure. Technical Report 2.0, OMG, 2004.

[13] Laurent Henocque. Modeling object oriented constraint programs in Z. *RACSAM (Revista de la Real Academia De Ciencias serie A Mathematicas), special issue about Artificial Intelligence and Symbolic Computing*, pages 127–152, 2004.

[14] Ulrich Junker and Daniel Mailharro. The logic of ilog (j)configurator: Combining constraint programming with a description logic. In *proceedings of Workshop on Configuration, IJCAI'03*, pages 13–20, 2003.

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

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

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

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

[19] 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.

[20] 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.

[21] 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.

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

[23] J. M. Spivey. *The Z Notation: a reference manual*. Prentice Hall originally, now J.M. Spivey, 2001.

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

[25] 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.

[26] M. Viroli. Towards a formal foundation to orchestration languages. In *proceedings of 1st International Workshop on Web Services and Formal Methods (web service-FM 2004)*, Pisa, Italy, February 23-24 2004.

[27] 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.

[28] 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.