Un modèle de composition automatique et distribuée de services web par planification

Damien Pellier, Humbert Fiorino

Un modèle de composition automatique et distribuée de services web par planification

Damien Pellier* — **Humbert Fiorino****

* *Centre de Recherche en Informatique de Paris 5*
Université Paris Descartes
45, rue des Saints Pères, F-75270 Paris cedex
damien.pellier@parisdescartes.fr

** *Laboratoire d'Informatique de Grenoble*
110 av. de la Chimie, BP 53 - 38041 Grenoble cedex 9
humbert.fiorino@imag.fr

RÉSUMÉ. *L'avènement des services web comme une technologie incontournable du web et sa dissémination à grande échelle pose dorénavant la problématique de leur composition automatique. En effet, l'un des verrous les plus importants au développement des architectures orientées services réside dans l'élaboration manuelle par un expert de services composites. Afin de répondre à cette problématique, nous proposons dans cet article une architecture originale de composition automatique de services web par des techniques de planification. Son originalité repose sur la conception d'un modèle de planification entièrement distribué dans lequel les agents raisonnent conjointement sur leurs services respectifs pour atteindre un but commun prédéfini par l'utilisateur, créant ainsi un plan global représentant une composition possible de leurs services.*

ABSTRACT. *Web services advent as an inevitable technology of the Web and its dissimination on a large scale, poses the problem of their automatic composition. Indeed, one of the most important obstacle to the development of web services oriented architectures relies on the manual generation of composite services by human experts. In order to overtake this approach, we propose in this article a novel architecture for web services composition based on planning techniques. Its originality consists in its completely distributed planning model where agents reason together on their own services to achieve a shared goal defined by users and where the global shared plan built stand for a possible composition of their services.*

MOTS-CLÉS : *Composition de services web, planification multiagent*

KEYWORDS: *Web Service Composition, Multiagent Planning*

RSTI - RIA - 23/2009. Intelligence artificielle et web intelligence, pages 13 à 46

1. Introduction

L'intérêt des services web est de permettre à une entreprise d'exporter au travers du réseau internet ses compétences et son savoir-faire, d'interagir avec ses partenaires, de rechercher de nouveaux marchés et de nouveaux supports de vente. En juillet 2002, Amazon.com a ainsi été l'une des toutes premières à obtenir un fort écho médiatique en rendant sa base de données d'articles accessible par un service web (Wolverton, 2002). Ces architectures orientées services qui trouvent leurs origines dans l'informatique distribuée en prenant le réseau internet comme plate-forme d'exécution de composants logiciels interopérables conduisent à des interactions complexes à grande échelle et des défis nouveaux. En effet, contrairement aux interfaces de programmation (*Application Programming Interface*, API) « classiques », les services web sont conçus pour découvrir et invoquer d'autres services et tirent leur versatilité de leurs interfaces qui sont des abstractions n'imposant aucune contrainte en matière de mise en œuvre, *e.g.*, langage de programmation, système d'exploitation, *etc.*

Actuellement, les services web reposent principalement sur des standards XML : WSDL (*Web Services Description Language*) permet une description syntaxique des services en termes d'entrées, sorties ; OWL-S (Martin *et al.*, 2004) a pour objectif de faire une description « sémantique », c'est-à-dire explicitant le « profile » du service (quelles sont les informations nécessaires à l'exécution du service ? quelles sont les informations renvoyées ?), son *process model* (comment fonctionne le service ?) et son *grounding* (de quelle façon le service doit-il être utilisé ?). La convergence entre web sémantique et services web (Medjahed *et al.*, 2003) a pour but d'augmenter l'expressivité des descriptions et de rendre plus efficace la gestion, la découverte, la composition et l'invocation des services au travers d'un protocole de communication et d'un répertoire de services UDDI (*Universal Description, Discovery and Integration*). Ce protocole permet à un « fournisseur » d'enregistrer son service et à un « consommateur » de trouver le service adéquat. Finalement, les services web s'appuient sur SOAP (*Simple Object Access Protocol*), un protocole d'échange de messages entre services fondé sur HTTP.

L'un des verrous les plus importants au développement des architectures orientées services est la création manuelle de services composites. Cette composition par un expert nécessite la mise au point de *middlewares* permettant de sélectionner les services répondant à ses besoins fonctionnels et non fonctionnels (qualité de service, confiance, *privacy etc.*) ; d'ordonner les services sélectionnés, les flux de données et de contrôle ; d'exécuter les services et surveiller leurs aléas d'exécution.

Plus spécifiquement, l'orchestration de services web (Peltz, 2003; Jamal, 2005) permet de définir l'enchaînement des services selon un canevas prédéfini, et de les exécuter à travers des « scripts d'orchestrations ». Ces scripts décrivent les interactions entre services en identifiant les messages échangés, les branchements logiques et les séquences d'invocation. La chorégraphie de services quant à elle trace la séquence de messages pouvant impliquer plusieurs sources (les clients, les fournisseurs, les partenaires). Elle est associée à l'échange de messages publics entre services web

plutôt qu'à un processus métier exécuté par un seul partenaire. Il existe une différence importante entre orchestration et chorégraphie de services web (Jamal, 2005) : l'orchestration s'appuie sur un processus métier exécutable pouvant interagir avec des services web internes et externes. Elle offre une vision centralisée : le procédé est toujours contrôlé du point de vue de l'un des partenaires. La chorégraphie est de nature plus collaborative : chaque participant impliqué dans le procédé décrit le rôle qu'il joue dans cette interaction. Les principaux langages d'orchestration ou de chorégraphie de services web sont répertoriés dans (Peltz, 2003) : XLANG – XML Business Process Language (Microsoft), BPML – Business Process Modeling Language (BPMI), WSFL – web Service Flow Langage (IBM), WSCL – web Service Conversation Language (Hewlett-Packard), WSCI – web Service Choregraphy Interface (SUN), BPEL4WS – Business Process Execution Language for WS (IBM, Microsoft, BEA).

Ces dernières années, de nombreux travaux ont porté sur l'automatisation de la composition des services (Singh *et al.*, 2005; Milanovic *et al.*, 2004; Guitton, 2006; Bourdon, 2007). Ils trouvent leur justification dans l'évolution constante de l'offre de services en ligne ainsi que de leurs propriétés non fonctionnelles, ce qui rend une description experte de la composition difficile à maintenir. La composition automatique permet aussi une adaptation aux attentes fluctuantes des utilisateurs. En d'autres termes, elle vise à tirer le meilleur parti des propriétés intrinsèques de la plate-forme d'exécution des services : décentralisation, modularité et « plasticité » d'internet. L'étude bibliographique montre que la planification automatique est un outil intéressant pour construire ces compositions. En revanche, toutes les architectures proposées reposent sur une centralisation de la composition. Dans cet article, nous proposons une architecture de composition entièrement distribuée de services web. Les questions de la découverte et de l'exécution des services (Sycara *et al.*, 2003) n'entrent pas dans le champ de cette étude. Notre objectif est de comprendre comment des algorithmes de planification distribuée peuvent contribuer à la création d'un service composite vu comme un plan définissant des relations de précédence et de causalité entre services élémentaires. Nous considérons que chaque service est un agent autonome capable de planifier. Lorsque l'un d'eux ne peut pas répondre à la requête d'un utilisateur, cette requête devient le but d'un processus distribué de planification dont le plan solution, s'il existe, est une représentation du service composite constitué des agents ayant pris part à sa construction.

L'article est organisé de la manière suivante : dans une première section, nous présentons les différents travaux traitant de la composition de services web ; dans un deuxième temps, nous introduisons un exemple qui sert de fil conducteur à la présentation de notre modèle de composition distibuée de services web par planification ; puis, dans une troisième partie, nous en donnons formellement les définitions préliminaires ; et terminons par la présentation de sa dynamique.

2. Etat de l'art

L'étude de la composition des services web est traitée par plusieurs communautés scientifiques (Benatallah *et al.*, 2005; Benatallah *et al.*, 2003; Hamadi *et al.*, 2003; Jayadev *et al.*, 2007). Dans cet article, nous nous concentrons sur l'automatisation de la composition par des approches d'intelligence artificielle et plus particulièrement par de la planification automatique (Ghallab *et al.*, 2004), notre objectif se limitant ici à situer nos travaux par rapport aux approches comparables.

L'idée d'automatiser la composition des services web n'est pas neuve. Golog (Levesque *et al.*, 1997; Giacomo *et al.*, 2000) est un des premiers langages de programmation logique, dont les bases sont une version étendue du "situation calculus" (McCarthy *et al.*, 1987) qui a été adapté pour permettre la composition dynamique de services web (Narayanan *et al.*, 2002; McIlraith *et al.*, 2002). Le principe de résolution est le suivant : les services disponibles, décrits ici en DAML-S (Ankolekar *et al.*, 2002) (prédécesseur de OWL-S), sont traduits dans un formalisme permettant de les manipuler. Dans ce travail, deux traductions sont opérées : en Golog pour manipuler les services à un niveau logique (avec les outils dédiés au "situation calculus") ; en réseau de Petri pour pouvoir utiliser les outils de vérification de propriétés (atteignabilité, famines *etc.*), de simulation de l'évolution d'un service web selon différentes conditions, d'analyse de performances et finalement de composition en séquence de services web.

Des travaux sur la composition automatique (Sirin *et al.*, 2004) et semi-automatique (Sirin *et al.*, 2002) de services web avec le planificateur SHOP2 (du type Hierachical Task Network, HTN) ont été menés (Wu *et al.*, 2003). En semi-automatique, un système d'aide à la composition par un expert a été proposé. En automatique, la méthode de résolution par Golog est reprise : une grande partie des structures de OWL-S a été traduite en opérateurs et méthodes HTN. WSPlan (Peer, 2005; Peer, 2004) est comme SHOP2 un planificateur HTN conçu pour la composition de services web. Son approche diffère surtout des autres par la notion de re-planifications à l'exécution.

CASCOM[1] (*Context-aware business Application Service CO-ordination in Mobile computing environment*) était un projet européen (2004-2007) dont l'objectif principal était de mettre en œuvre, valider et tester une valeur ajoutée pour une infrastructure d'appui du web sémantique utilisant des services à travers des réseaux fixes et mobiles. Le planificateur OWLS-Xplan (Klusch *et al.*, 2006) (du type HTN & Fast Forward Chaining) prend en entrée un ensemble de services OWL-S, une description de domaine de planification fondée sur des ontologies OWL et une requête de planification (but à satisfaire) et renvoie une composition en séquence de services satisfaisant ce but. Un post-traitement permet d'enrichir cette séquence de services par d'autres structures de contrôle (par exemple, *split* + *join*). Les outils développés dans ce projet permettent de sélectionner, composer et exécuter les services. L'architecture de CAS-

1. http ://www.ist-cascom.org/

COM est du type multi-agent car un agent PA (*Personal Agent*) envoie une requête à l'agent SCPA (*Service Composition Planner Agent*) qui obtient les descriptions des services nécessaires à la composition via l'agent SDA (*Service Discovery Agent*). Le plan de composition obtenu de manière centralisée par l'agent SCPA est transmis à l'agent SEA (*Service Execution Agent*) qui se charge de l'invocation des services et renvoie les résultats vers l'agent PA.

INFRAWEBS est un projet européen[2] dont l'objectif est le développement d'une application axée sur les outils logiciels pour la création, la maintenance et l'exécution de services web sémantiques dans l'ensemble de leur cycle de vie. L'approche de résolution n'est pas directement fondée sur une technique de planification : un but est décrit comme un ensemble d'expressions logiques pouvant être décomposées en sous-buts. Les expressions en question sont décrites à l'aide d'ontologies, et peuvent décrire ce que l'expert pourra fournir au système, ainsi que ce qu'il attend des services capables de satisfaire le but. Il s'agit alors de chercher un service capable de satisfaire le but ou, à défaut, de décomposer ce but en sous-buts de manière récursive. Pour chaque sous-but, les services compatibles sont proposés à l'expert pour qu'il sélectionne celui qui lui semble le plus adéquat – la composition n'est donc pas automatique. Un système de substitution de service, dans le cas où le service préalablement sélectionné ne fonctionne pas, est mis en place. Lors de la sélection d'un service, les autres services disponibles pour le même but sont mémorisés, pour permettre cette substitution durant l'exécution du service composite obtenu.

Dans (Medjahed *et al.*, 2003), une technique de *rule-based planning* est utilisée pour engendrer des services composites à partir de descriptions déclaratives de haut niveau. Cette méthode utilise des règles de composabilité pour déterminer dans quelle mesure deux services sont composables. L'approche proposée se déroule en quatre phases : une phase de spécification de haut niveau de la composition désirée en utilisant le langage CSSL (Composite Service Specification Langage). La phase de correspondance utilise des règles de composabilité pour générer des plans conformes aux spécifications du service demandeur. Dans la phase de sélection, si plus d'un plan est généré, la sélection est effectuée par rapport à des paramètres de qualité de la composition. Dans la phase de génération, une description détaillée du service composite est automatiquement générée et présentée au demandeur. La principale contribution de cette approche est la notion de règles de composabilité. Les règles de composabilité considèrent les propriétés syntaxiques et sémantiques des services web. Les règles syntaxiques incluent des règles pour les types d'opérations possibles et pour les liaisons protocolaires entre les services (les bindings). Les règles sémantiques incluent des règles concernant la compatibilité des messages échangés, la compatibilité des domaines sémantiques des services, mais également des règles de qualités de la composition.

Le projet ASTRO[3] (Pistore *et al.*, 2005b; Pistore *et al.*, 2005a; Pistore *et al.*, 2004) a pour but de développer des outils soutenant l'évolution et l'adaptation des services

2. http ://www.infrawebs.org/
3. http ://www.astroproject.org/

web distribués au cours de leur cycle de vie, de la conception à l'exécution, pour finalement automatiser la composition de ces services. La méthode de composition est la suivante : après avoir traduit les services en automates à changements d'états, on utilise une planification centralisée et fondée sur les techniques de model checking (MBP, Model Based Planning). L'expressivité de cette planification en termes de structures de contrôle est supérieure. Un langage appelé EaGLe permet à l'expert de définir dans une description de haut niveau ses besoins de service composite. La planificateur MBP explore exhaustivement l'ensemble des états possibles des automates, ce qui impose de réduire le nombre de ces états en les représentant à un haut niveau d'abstraction (*knowledge level*). Ainsi, les états sont du type « objets du type y disponibles » plutôt que « l'objet x est disponible ». L'autre inconvénient de cette approche est que l'ensemble des états possibles doit être recalculé chaque fois qu'un service est ajouté ou retiré.

Par notre approche, nous cherchons à surmonter ces difficultés de planification d'un service composite. Contrairement aux travaux précédemment cités, la représentation utilisée dans notre modèle est une représentation symbolique et abstraite (langage HTN (Ghallab *et al.*, 2004)) qui se rapproche au mieux de la représentation WSDL. En outre des travaux récents montrent que les techniques de planification mises en œuvre dans le modèle peuvent faire l'objet d'heuristiques efficaces (Nguyen *et al.*, 2001). L'apport de nos travaux repose sur un modèle de composition par planification *complètement automatique et totalement distribuée* aussi bien au niveau du contrôle qu'au niveau des données. L'objectif est de tirer parti au mieux des propriétés intrinsèques d'internet : décentralisation, modularité etc. Les questions centrales de la découverte et de l'invocation des services web (Sycara *et al.*, 2003) ne sont pas traitées ici car elles font appel à des travaux dépassant le cadre de la planification automatique.

3. Exemple introductif

Le scénario suivant permet d'illustrer ce que nous attendons par composition automatique et distribuée : Bob habite Grenoble et doit se rendre à New-York pour une conférence. Il décide d'organiser son voyage par internet en faisant appel à trois services web. Chaque service est représenté par un agent : un agent SNCF capable de réserver des billets de train ; un agent Airways offrant un service de réservation de billets d'avion et un agent Bank (représentant la banque de Bob) qui est en charge de payer les différentes réservations que Bob sera amené à réaliser. Imaginons maintenant le dialogue que les trois agents pourraient construire pour que Bob puisse se rendre à sa conférence :

Bob :	« Je suis à Grenoble et je dois me rendre à New-York. Pouvez-vous m'aider ? »
SNCF :	« Je ne peux malheureusement pas t'aider, je ne sais pas comment aller à New-York. »

Airways : « En ce qui me concerne, je peux t'emmener à New-York à condition que tu sois capable de te rendre à Londres et que tu me paies la somme de 250 euros. »
SNCF : « Je ne sais pas non plus comment aller à Londres. »
Bank : « Je peux payer la somme de 250 euros, le compte de Bob est créditeur. »
Airways : « Bon ce n'est pas grave, j'ai un autre vol en partance de Paris pour New-York à condition que Bob soit à Paris et que je reçoive la somme de 100 euros ».
Bank : « Je peux payer les 100 euros du billet d'avion. »
SNCF : « Il existe un train de Grenoble à Paris. En revanche, il faut que Bob puisse s'acquitter de la somme de 50 euros. »
Bank : « Parfait, je crois que nous tenons la solution au problème de Bob, Je peux également payer les 50 euros du billet de train. »

Le plan solution est donc : « Prendre le train de Grenoble à Paris puis un vol de Paris à NY. » Sa construction ne repose pas sur une planification centralisée mais sur la coopération de plusieurs agents planificateurs.

4. Définitions préliminaires

Dans cette section, nous définissons les notions préliminaires nécessaires à la formalisation de notre modèle de planification distribuée pour la composition de services web illustré par la figure 1.

4.1. *Les états de croyance*

La représentation des croyances repose sur une notation dérivée de la logique du premier ordre (Ghallab *et al.*, 2004). En effet, la logique du premier ordre se prête bien à la description de propriétés générales sur le monde ; c'est donc elle que nous avons retenue comme base pour construire le langage $\mathcal{L}$ utilisé dans notre approche. Chaque terme de $\mathcal{L}$ est soit une variable soit une constante (nous ôtons les symboles de fonction du langage prédicatif). Les prédicats de la forme $P(t_1, \ldots, t_n)$ avec P un symbole de prédicat n-aire et $t_1, \ldots, t_n$ des termes de $\mathcal{L}$ codent les propriétés du monde manipulées par les agents. Finalement, les mots de $\mathcal{L}$ sont soit des termes soit des formules construites à partir des connecteurs classiques de la logique du premier ordre.

Un *état de croyance* est un ensemble de prédicats instanciés de $\mathcal{L}$. Étant donné que $\mathcal{L}$ ne possède pas de fonctions et que la description des croyances des agents fait intervenir un nombre fini de constantes, l'ensemble des états de croyance possibles est également fini. On peut alors traduire l'ensemble des formules de $\mathcal{L}$ sous forme d'un

ensemble de propositions et utiliser les algorithmes classiques du calcul propositionnel. Nous disons qu'un prédicat p est vérifié dans un état de croyance s si et seulement si p peut être unifié avec un prédicat de s tel que $\sigma(p) \in s$, où σ est la substitution résultat de l'unification de p avec le prédicat de s. Dans le cas contraire, nous considérons la propriété du monde représentée par p comme étant inconnue. Par conséquent, il n'est plus possible de poser l'hypothèse du monde clos. Autrement dit, un prédicat p n'est pas vérifié dans un état s si et seulement si $\sigma(\neg p) \in s$.

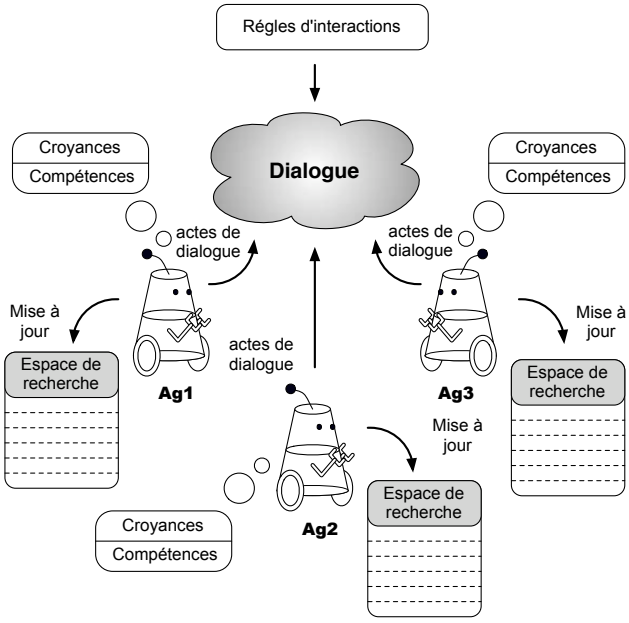

Figure 1. *Aperçu du modèle de planification distribuée pour la composition de services web*

4.2. *Opérateurs et méthodes*

Les opérateurs de planification sont définis comme des fonctions de transition au sens classique (Ghallab *et al.*, 2004) : actions instantanées, statiques, déterministes et observabilité totale.

Définition 4.1 (Opérateur) Un *opérateur* o est un triplet de la forme $(\mathit{name}(o), \mathit{precond}(o), \mathit{effects}(o))$ ou $\mathit{name}(o)$ définit le nom de l'opérateur, $\mathit{precond}(o)$ l'ensemble des préconditions de o à satisfaire et $\mathit{effects}(o)$ l'ensemble de ses effets. Par la suite nous noterons respectivement $\mathit{effects}^{+}(o)$ les effets positifs et $\mathit{effects}^{-}(o)$ les effets négatifs d'un opérateur o.

Exemple 4.1 À titre d'exemple, nous donnons ci-dessous les opérateurs associés à l'agent Airways :

;; L'agent Airways déplace le passager ?p de la ville ?from à la ville ?to
!move(*?p*, *?from*, *?to*)
precond: flight(*?from*, *?to*), at(*?p*, *?from*)
effects: at(*?p*, *?to*), ¬at(*?p*, *?from*)

;; L'agent Airways effectue une réservation pour le vol au départ de ?from pour ?to
;; pour le compte du passager ?p
!book(*?p*, *?from*, *?to*)
precond: flight(*?from*, *?to*), receive-cash(*?from*, *?to*, *?m*),
is-available(*?from*, *?to*, *?q*), (*?q*>1), account(*?a*)
effects: is-available(*?from*, *?to*, *?q*−1), ¬is-available(*?from*, *?to*, *?q*),
account(*?a*+ *?m*), ¬account(*?a*)

Nous complétons la définition des opérateurs en ajoutant le concept de *méthode* utilisé dans la planification hiérarchique (Nau *et al.*, 2003). Contrairement à un opérateur qui décrit un ensemble d'actions, une méthode définit un ensemble de décompositions d'une tâche en actions pouvant être réalisées par un agent.

Définition 4.2 (Méthode) Une *méthode* m est un triplet de la forme $(\mathit{name}(m), \mathit{precond}(m), \mathit{reduction}(m))$ où $\mathit{name}(m)$ est une expression de la forme $n(x_1, \ldots, x_k)$ telle que n représente le nom de la méthode et $x_1, \ldots, x_k$ ses paramètres, $\mathit{precond}(m)$ représente les préconditions (*i.e.*, un ensemble de prédicats) devant être vérifiées dans l'état des croyances de l'agent pour que m soit appliquée et $\mathit{reduction}(m)$ définit la séquence d'opérateurs ou de méthodes à accomplir pour réaliser m.

Exemple 4.2 Nous donnons ci-dessous la méthode de recherche d'un vol de l'agent Airways :

;; L'agent Airways cherche un vol de ?from à ?to pour le passager ?p
search-flight(*?p*, *?from*, *?to*)
precond: flight(*?from*, *?to*), at(*?p*, *?from*),
is-available(*?from*, *?to*, *?q*), (*?q*>1)
reduction: !book(*?p*, *?from*, *?to*), !move(*?p*, *?from*, *?to*)

4.3. *Agent et problème*

Jusqu'à présent, nous avons introduit le langage sur lequel repose la description des croyances des agents ainsi que les opérateurs nécessaires à leur manipulation. Dans cette section, nous posons formellement les définitions d'agent et de problème classiquement utilisées en planification.

Définition 4.3 (Agent) Un *agent* α est un quadruplet de la forme ($\mathit{name}(\alpha)$, $\mathit{operators}(\alpha), \mathit{methods}(\alpha), \mathit{belief}(\alpha)$) où $\mathit{name}(\alpha)$ est le nom de l'agent, $\mathit{operators}(\alpha)$ est un ensemble d'opérateurs, $\mathit{methods}(\alpha)$ est un ensemble de méthodes (l'ensemble des opérateurs et des méthodes définissent les compétences de α) et $\mathit{belief}(\alpha)$ décrit l'ensemble des propriétés du monde connues par α.

Il reste maintenant à définir un problème de planification. Un problème doit spécifier les états initiaux des croyances des agents, les opérateurs et méthodes qu'ils peuvent appliquer ainsi que le but qu'ils doivent réaliser. Le but est représenté par un ensemble de propositions décrivant les propriétés du monde qui doivent être vérifiées.

Définition 4.4 (Problème de planification) Un *problème de planification* $\mathcal{P}$ est un triplet $(s_0, \mathcal{O}, g)$ où s_0 et $\mathcal{O}$ représentent respectivement l'union des croyances, *i.e.*, l'état initial du problème de planification, et les opérateurs (méthodes incluses) des agents et g définit un ensemble cohérent de propositions, *i.e.*, les propriétés du monde devant être atteintes par les agents.

Nous faisons l'hypothèse restrictive que l'union des croyances des agents d'un problème de planification est cohérente (cas classique de la planification mono-agent), *i.e.*, pour deux agents α et β, si une proposition $p \in \mathit{belief}(\alpha)$ alors $\neg p \notin \mathit{belief}(\beta)$. Cependant, aucune hypothèse n'est faite sur le possible partage de croyances entre les agents en termes de faits ou d'opérateurs.

Exemple 4.3 Chaque agent est décrit par un fichier OWL-S et dispose d'une base de croyances sous la forme d'un ensemble de propositions. Le but des agents de notre exemple est défini formellement par $g = \{\mathsf{at}(\mathsf{bob}, \mathsf{NewYork})\}$. Par souci de concision, nous donnons ici pour chaque agent l'état initial simplifié de leurs croyances leur permettant de résoudre le problème.

$$\mathit{belief}(\mathsf{Airways}) = \left\{ \begin{array}{l} \mathsf{flight}(\mathsf{London}, \mathsf{NewYork}), \\ \mathsf{flight}(\mathsf{Paris}, \mathsf{NewYork}), \\ \mathsf{flight}(\mathsf{NewYork}, \mathsf{Paris}), \\ \mathsf{is\text{-}available}(\mathsf{London}, \mathsf{NewYork}, 8), \\ \mathsf{is\text{-}available}(\mathsf{Paris}, \mathsf{NewYork}, 10), \\ \mathsf{is\text{-}available}(\mathsf{NewYork}, \mathsf{Paris}, 2), \\ \mathsf{receive\text{-}cash}(\mathsf{London}, \mathsf{NewYork}, 250), \\ \mathsf{receive\text{-}cash}(\mathsf{Paris}, \mathsf{NewYork}, 100), \\ \mathsf{receive\text{-}cash}(\mathsf{NewYork}, \mathsf{Paris}, 150), \\ \mathsf{account}(10000) \end{array} \right\}$$

$$\mathit{belief}(\mathsf{SNCF}) = \left\{ \begin{array}{l} \mathsf{train}(\mathsf{Grenoble}, \mathsf{Paris}), \\ \mathsf{train}(\mathsf{Paris}, \mathsf{Grenoble}), \\ \mathsf{is\text{-}available}(\mathsf{Grenoble}, \mathsf{Paris}, 12), \\ \mathsf{is\text{-}available}(\mathsf{Paris}, \mathsf{Grenoble}, 6), \\ \mathsf{receive\text{-}cash}(\mathsf{Grenoble}, \mathsf{Paris}, 50), \\ \mathsf{receive\text{-}cash}(\mathsf{Paris}, \mathsf{Grenoble}, 70), \\ \mathsf{account}(12000) \end{array} \right\}$$

$$\mathit{belief}(\mathsf{Bank}) = \left\{ \begin{array}{l} \mathsf{account}(\mathsf{bob}, 500), \\ \mathsf{allowed\text{-}overdraft}(\mathsf{bob}, 300) \end{array} \right\}$$

4.4. *Représentation des plans*

Classiquement, un *plan* est un ensemble d'actions contenues dans une structure particulière exprimant des relations entre les actions. Dans le cas d'une séquence, la relation entre les actions est une relation d'ordre total. Le choix d'une telle structure semble trop restrictif pour s'appliquer dans un contexte multi-agent. En effet, elle ne permet pas de définir simplement la notion de plan mise en œuvre dans notre approche, ni de décrire des actions concurrentes. Ceci nous amène à retenir pour notre approche la notion de plan partiel utilisée par les algorithmes de planification dans un espace de plans tels que (Penberthy *et al.*, 1992).

Pour illustrer tous les aspects d'un plan partiel reprenons l'exemple 4.3 comme fil conducteur. Nous supposons qu'il existe un plan partiel initial constitué de deux actions proposées par l'agent Airways qui permet d'atteindre le but at(bob,NewYork) :

– !book(bob,Paris,NewYork)

– !move(bob,Paris,NewYork)

Regardons comment le plan partiel doit être raffiné par ajouts successifs d'actions[4] et de quelle manière s'effectue sa mise à jour. Cela nous permettra d'introduire de manière informelle la notion d'hypothèse ainsi que les quatre constituants d'un plan partiel : un ensemble d'actions, un ensemble de contraintes d'ordre, un ensemble de contraintes d'instanciation et un ensemble de liens causaux.

Les actions. Pour l'instant rien ne garantit au sein du plan partiel que Bob soit à Paris pour prendre son avion jusqu'à New-York. Par conséquent, la propriété at(Bob,Paris), requise par les préconditions de l'action !fly, est une hypothèse formulée par le plan partiel initial. Pour vérifier cette hypothèse, l'agent SNCF propose de raffiner ce plan partiel en ajoutant la séquence suivante de deux actions :

– !book(bob,Grenoble,Paris)

– !move(bob,Grenoble,Paris)

De la même manière, rien ne garantit que l'agent Airways soit payé pour effectuer la réservation du billet d'avion Paris – New-York. La précondition de l'action !book(bob,Paris,NewYork), receive-cash(Paris,NewYork,100), est également une hypothèse formulée par le plan partiel. Pour vérifier cette hypothèse, l'agent Bank propose d'ajouter l'action suivante : !pay(bob,100).

4. Par abus de langage, nous utiliserons dans la suite de cet article le terme générique d'action pour caractériser à la fois une action en tant qu'instance d'un opérateur de transformation et l'opérateur lui-même.

Les contraintes d'ordre. L'action proposée par l'agent SNCF (!move(bob, Grenoble,Paris)), et celle proposée par l'agent Bank (!pay(bob,100)), doivent être exécutées respectivement avant l'action !move(bob,Paris,NewYork) et !book(bob,Paris,NewYork) pour satisfaire les hypothèses formulées par le plan partiel initial. En effet, rien n'indique pour l'instant l'ordre dans lequel ces actions doivent être exécutées. Par conséquent, il est nécessaire d'ajouter des contraintes d'ordre précisant que !move(bob,Grenoble,Paris) doit être réalisée avant !move(bob,Paris,NewYork) et !pay(bob,100) avant !book(bob,Paris,NewYork).

En revanche, est-ce que l'action !move(bob,Paris,NewYork) doit être exécutée avant ou après !pay(bob,100) ? Les deux options sont possibles. Dans l'état actuel du plan partiel rien n'oblige à trancher pour l'une ou l'autre des solutions. Nous appliquons ici le principe de *moindre engagement*. L'ajout d'une contrainte n'a lieu que si elle est strictement nécessaire. Si aucune autre contrainte d'ordre n'est ajoutée au plan partiel au cours du processus de planification, alors les actions proposées par l'agent SNCF et l'agent Bank pourront être exécutées de manière concurrente.

Les liens causaux. Pour le moment, nous savons ajouter des actions et des contraintes d'ordre à un plan partiel. Mais est-ce suffisant ? À cause de la représentation non explicite de la notion d'état courant (car distribué sur l'ensemble des agents), les contraintes d'ordre ne suffisent pas à garantir, par exemple, que Bob restera à Paris jusqu'à ce que l'action !move(Bob,Paris,NewYork) soit réalisée. En effet, au cours du processus de planification, les agents peuvent trouver d'autres raisons de déplacer Bob dans une autre ville pour une correspondance et oublier la raison qui les a fait le déplacer à Paris. Par conséquent, il est nécessaire de coder explicitement au sein du plan partiel les raisons qui ont fait que les actions ont été ajoutées. Ainsi, dans notre exemple, il faut spécifier que l'action !move(Bob,GrenobleParis) de l'agent SNCF a été ajoutée pour satisfaire la précondition at(Bob,Paris) de l'action !move(Bob,Paris,NewYork).

La relation entre les actions !move(Bob,Grenoble,Paris) et !move(Bob,Paris, NewYork) portant sur la propriété at(Bob,Paris) est appelée un *lien causal*. L'action !move(Bob,Paris,NewYork) est appelée le consommateur et l'action !move(Bob,Grenoble,Paris) le producteur. Autrement dit, un lien causal exprime qu'une propriété du monde nécessaire à l'exécution d'une action est satisfaite par les effets d'une autre action. En l'absence de lien causal, la précondition de l'action n'est pas vérifiée et sera considérée comme une hypothèse formulée par le plan partiel. Les hypothèses sont alors assimilées à des *sous-buts* devant être réalisés par les autres agents.

Notons qu'une action qui supporte une hypothèse doit toujours être réalisée avant l'action qui la formule. Par conséquent, un lien causal est toujours associé à une relation d'ordre, mais il est possible d'avoir une contrainte d'ordre sans lien causal. Toutefois, d'autres actions peuvent être intercalées entre les deux actions liées par un lien causal. Un lien causal n'est donc pas garant de l'absence de conflit entre deux actions.

Les contraintes d'instanciation. Il est nécessaire de préciser les contraintes d'instanciation relatives aux variables manipulées par les opérateurs de transformation décrivant les actions. En effet, chaque opérateur, comme présenté dans la section 4.2 décrit un ensemble d'actions. L'unification des préconditions d'un opérateur avec l'état de croyance d'un agent peut définir plusieurs actions applicables à partir d'un même état. Il faut donc garantir, par exemple, que le nouvel opérateur !move(Bob,Grenoble,Paris) concerne bien Bob ainsi que le même lieu d'arrivée que le lieu de départ de l'opérateur !move(Bob,Paris,NewYork). Finalement, notons que certaines variables peuvent ne pas être instanciées. Les variables non instanciées traduisent le fait qu'un agent ne connaît pas, pour l'instant, la valeur exacte qui lui sera associée. Nous parlons alors d'action partiellement instanciée.

Pour résumer, nous avons ajouté au plan partiel des actions, des contraintes d'ordre, des liens causaux ainsi que des contraintes d'instanciation. Ces éléments constituent les éléments nécessaires à la formalisation de la notion de plan partiel utilisé dans notre approche.

Définition 4.5 (Plan partiel) Un *plan partiel* est un tuple $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ dont les éléments sont les suivants :

– $\mathcal{A} = \{a_0, \ldots, a_n\}$ est un ensemble d'actions ;

– $\prec$ est un ensemble de contraintes d'ordre sur les actions $\mathcal{A}$ de la forme $a_i \prec a_j$, *i.e.*, a_i précède a_j ;

– $\mathcal{I}$ est un ensemble de contraintes d'instanciation portant sur les variables des actions $\mathcal{A}$ de la forme $?x = ?y$, $?x \neq ?y$, ou $?x = \mathsf{cst}$ tel que $\mathsf{cst} \in D_{?x}$ et $D_{?x}$ est le domaine de $?x$;

– $\mathcal{C}$ est un ensemble de liens causaux de la forme $a_i \xrightarrow{p} a_j$ tels que a_i et a_j sont deux actions de $\mathcal{A}$, la contrainte d'ordre $a_i \prec a_j$ existe dans $\prec$, la propriété p est un effet de a_i et une précondition de a_j et finalement les contraintes d'instanciation qui lient les variables de a_i et de a_j portant sur la propriété p sont contenues dans $\mathcal{I}$.

Les hypothèses formulées par un plan partiel sont représentées par les préconditions des actions qui ne sont pas supportées par un lien causal.

Définition 4.6 (Hypothèse) Soit un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. Une *hypothèse* formulée par π est définie comme une précondition p d'une action $a_j \in \mathcal{A}$ telle que pour toutes actions $a_i \in \mathcal{A}$, le lien causal $a_i \xrightarrow{p} a_j \notin \mathcal{C}$. Nous notons respectivement *assump*(π) et *assump*(a_j) l'ensemble des hypothèses formulées par π et par a_j.

De plus, l'ordonnancement partiel des actions implique qu'un plan partiel définit un ensemble de séquences d'actions totalement ordonnées respectant $\prec$.

Définition 4.7 (Linéarisation) Soit un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. On appelle *linéarisation* de π toute séquence d'actions $\lambda = (\mathcal{A}, <, \mathcal{I}, \mathcal{C})$, où $<$ est un ordre total

sur $\mathcal{A}$ compatible avec $\prec$, qui définit une séquence de $n+1$ états $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ pour $0 \leq i \leq n$ avec

$$s_i = (((s_{i-1} \cup \textit{assump}(a_{i-1})) - \textit{effects}^-(a_{i-1})) \cup \textit{effects}^+(a_{i-1})$$

Définition 4.8 (Complétion) Soit un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. On appelle *complétion* de π l'ensemble des linéarisations de π, noté *completion*(π).

Nous dirons que l'ensemble des contraintes d'ordre $\prec$ d'un plan partiel π est *cohérent* si *completion*(π) est non vide. Cela signifie qu'il existe au moins une linéarisation possible de π. Pour tester la cohérence des contraintes d'ordre $\prec$ d'un plan partiel π, il faut vérifier que $\prec$ n'exprime aucun cycle de dépendance entre les actions de π. La vérification de cette propriété s'effectue en calculant la fermeture transitive de la relation d'ordre définie par $\prec$. Le calcul de la fermeture transitive permet de déterminer pour chaque couple d'actions a_i et a_j s'il existe une relation d'ordre[5].

Finalement, l'absence de la notion d'état oblige à représenter les buts par une action particulière. Étant donné que les préconditions d'une action définissent les hypothèses potentielles pouvant être formulées par une action, les buts g sont représentés par une action fictive a_∞ qui ne possède pas d'effets. De manière similaire, la représentation de l'état initial nécessite l'introduction d'une action fictive a_0. Cette action ne possède pas de précondition mais des effets qui représentent l'état initial. Notons que l'état initial global n'est pas accessible directement puisqu'il est réparti sur l'ensemble des agents. Par conséquent, cet état est construit au cours du processus de synthèse de plans par ajout d'effets à a_0. Une représentation graphique du plan partiel permettant d'atteindre le but du problème 4.3 est donnée à la figure 2.

4.5. *Plans solutions et réfutations*

Classiquement, un *plan-solution* se définit comme un chemin dans un espace d'états. Le passage d'un état à l'autre s'effectue par l'application d'une action, *i.e.*, un opérateur complètement instancié, respectant la définition 4.7. Par conséquent, un plan-solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ est une séquence d'actions décrivant un chemin d'un état initial s_0, représentant l'union des croyances des agents, à un état final s_n tel que $g \subseteq s_n$. Or, dans notre approche, nous devons tenir compte du fait qu'un plan partiel peut contenir des hypothèses et définit non pas une séquence mais un ensemble de linéarisations. Par conséquent, toutes les linéarisations d'un plan partiel doivent décrire un chemin de l'état s_0 à s_n pour que le plan partiel soit un plan-solution valide. En outre, il est clair que si un plan partiel π ne définit pas un ensemble de contraintes d'ordre $\prec$ cohérent, alors π ne peut être un plan-solution. Ceci fournira un moyen d'éliminer des voies de recherche inutiles, en

5. Une relation d'ordre sur un ensemble E est une relation binaire dans E, à la fois réflexive, antisymétrique et transitive. Cette relation d'ordre est totale si deux éléments quelconques de E sont comparables sinon elle est partielle.

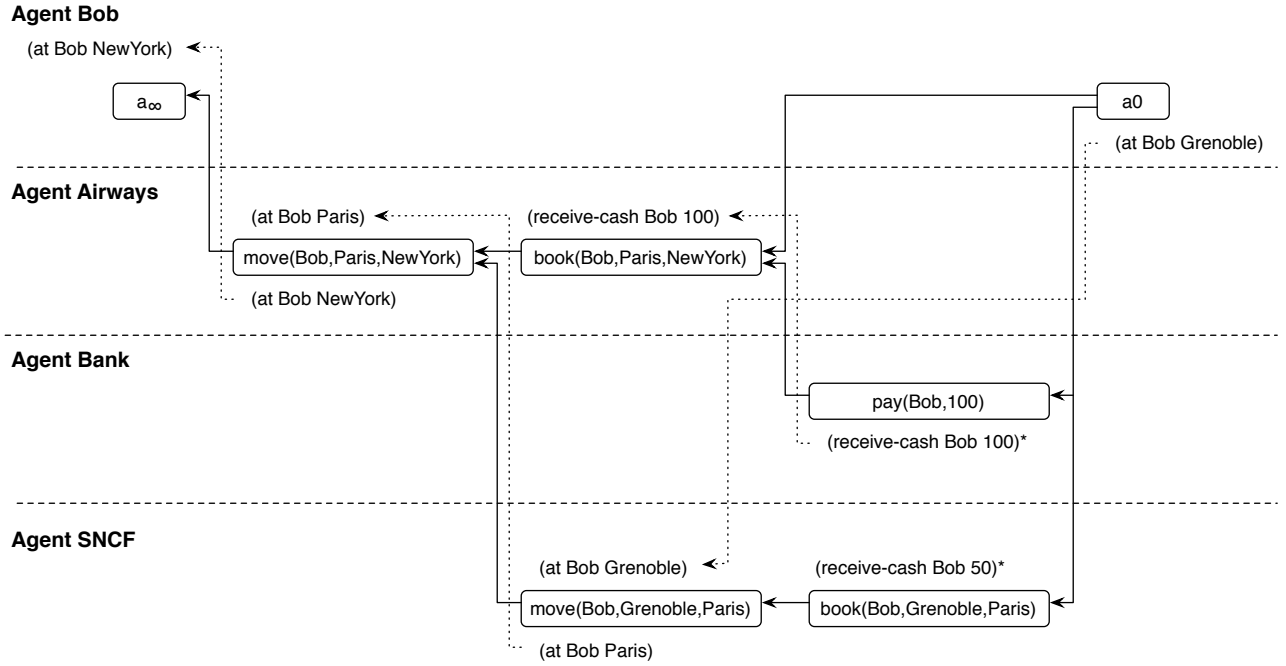

Figure 2. *Plan partiel initial de l'exemple 4.3 : les boîtes représentent les actions, les flèches pleines les contraintes d'ordre et les flèches pointillées les liens causaux entre les actions. Les contraintes d'instanciation sont décrites explicitement dans les actions et les propositions. Les propositions marquées d'un astérix sont des hypothèses*

interdisant aux agents d'introduire des cycles de dépendances. Rappelons également que les contraintes d'instanciation utilisées dans notre modèle sont de trois types : les contraintes unaires de la forme $?x = \mathsf{cst}$, $\mathsf{cst} \in D_{?x}$ et les contraintes binaires de la forme $?x = ?y$ et $?x \neq ?y$. Il faut donc également garantir qu'aucune des contraintes d'instanciation de $\mathcal{I}$ n'exprime de contradiction, par exemple :

$$\mathcal{I} = \{ ?x = \mathsf{c1}, ?x = ?y, ?y = \mathsf{c2}, ?z \neq ?x, ?z = \mathsf{c1} \}$$

En conclusion, nous donnons la définition d'un plan-solution :

Définition 4.9 (Plan-solution) Un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un *plan-solution* pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ si l'ensemble des contraintes d'ordre $\prec$ et l'ensemble des contraintes d'instanciation $\mathcal{I}$ sont cohérents et toutes les linéarisations $\lambda \in$ *completion*(π) définissent une séquence d'états cohérents $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ pour $0 \leq i \leq n$ tels que

– le but g est vérifié dans l'état s_n, *i.e.*, $g \subseteq s_n$;

– λ ne formule aucune hypothèse, *i.e.*, *assump*$(\lambda) = \emptyset$.

Malheureusement, la seconde partie de la définition, qui consiste à tester systématiquement pour chaque linéarisation d'un plan partiel si elle décrit une séquence d'états cohérents conduisant à un état but, ne définit pas une condition aisément calculable. Par conséquent, nous avons besoin de spécifier un ensemble de propriétés traduisant de façon pratique cette condition. Pour cela, nous la réexprimons en termes de *réfutations*. Bien qu'un plan partiel ne formule plus d'hypothèse, il peut ne pas être assez contraint pour garantir que toutes les séquences d'actions possibles définies par $\prec$ soient exemptes de conflit. En effet, un lien causal $a_i \xrightarrow{p} a_j$ n'interdit pas que d'autres actions soient exécutées entre a_i et a_j. Pour s'en persuader, considérons le plan partiel de la figure 3. Supposons que l'effet $\neg q$ soit produit par l'action a_k, et que q soit unifiable avec p. L'action a_k invalide potentiellement une précondition nécessaire à l'exécution de a_j. En l'absence de contrainte d'ordre entre a_k et les actions a_i et a_j, le plan partiel définit au moins une sous-séquence d'actions $\langle a_i, \ldots, a_k, \ldots, a_j \rangle$ invalide : la propriété du monde représentée par p n'est pas vérifiée dans l'état précédant l'exécution de a_j. Pour capturer cette condition et ainsi supprimer les séquences d'actions non valides, nous définissons ce que nous appelons une *réfutation*. Le concept de réfutation utilisé ici n'a rien avoir avec le concept de réfutation de la programmation logique. Il doit être rapproché de celui de *« clobber »* introduit par (Chapman, 1987).

Définition 4.10 (Réfutation) Une *réfutation* portant sur un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un couple $(a_k, a_i \xrightarrow{p} a_j)$ tel que : (i) a_k a pour effet $\neg q$ avec p et q unifiables ; (ii) les contraintes d'ordre $a_i \prec a_k$ et $a_k \prec a_j$ sont cohérentes avec $\prec$ et (iii) les contraintes d'instanciation résultant de l'unification de p et q sont cohérentes avec $\mathcal{I}$.

Par la suite, nous utiliserons le terme de *menace* pour caractériser l'ensemble des hypothèses et des réfutations d'un plan partiel.

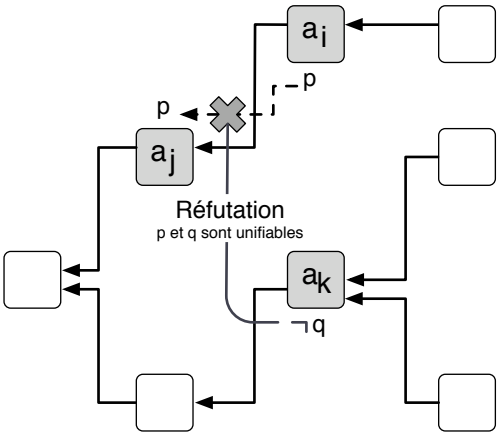

Figure 3. *Exemple de réfutation. p et q sont deux prédicats unifiables*

Proposition 4.1 Un plan partiel $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est un plan-solution pour un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, g)$ si les ensembles de contraintes d'ordre $\prec$ et d'instanciation $\mathcal{I}$ sont cohérents et π ne contient aucune menace.

Le lemme suivant permet de prouver la proposition 4.1 :

Lemme 4.1 Soit $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ un plan partiel et un lien causal $(a_i \xrightarrow{p} a_n) \in \mathcal{C}$. Nécessairement $p \in s_n$ s'il n'existe pas de réfutation $(a_k, a_i \xrightarrow{p} a_n)$.

Preuve 4.1 Preuve par induction sur la longueur de $\lambda \in$ *completion*(π) :

Cas de base : soit $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ avec $\mathcal{A} = \{a_0, a_\infty\}$. *completion*$(\pi) = \{\lambda\}$ et $\lambda = \langle a_0, a_\infty \rangle$. $s_0 = s_n$ et, par définition, il n'existe pas de réfutation possible $(\forall p \in s_0, p \in s_n)$.

Induction : Supposons que le lemme est vérifié pour π ayant n actions. Montrons qu'il est également vrai pour π composée de $n+1$ actions. Soit $\lambda \in$ *completion*(π) avec $\lambda = \langle a_0, \ldots, a_{n-1}, a_n \rangle$ $(a_n = a_\infty)$ et $\lambda' = \langle a_0, \ldots, a_{n-1} \rangle$. D'après l'hypothèse d'induction, $\forall (a_i \xrightarrow{p} a_{n-1})$ pour $0 \leq i < n-1$, $p \in s_{n-1}$ s'il n'existe pas de réfutation $(a_k, a_i \xrightarrow{p} a_{n-1})$ pour $0 \leq k < n-1$. Par définition, $s_n = ((s_{n-1} \cup$ *assump*$(a_{n-1}) -$ *effects*$^-(a_{n-1})) \cup$ *effects*$^+(a_{n-1})$. Par conséquent, $\forall p \in s_n$, soit $p \in$ *effects*$^+(a_{n-1})$, soit $p \in s_{n-1} \cup$ *assump*(a_{n-1}) et $p \notin$ *effects*$^-(a_{n-1})$. Dans le premier cas, p est produit par a_{n-1} et il n'y a pas de réfutation possible. Dans le second cas, p a été produit par λ' et n'est pas réfutée par a_{n-1}. Donc, dans tous les cas, le lemme est vérifié.

Preuve 4.2 Soit $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$. $\prec$ et $\mathcal{I}$ sont cohérents et il n'y a pas de menace dans π. Donc, $\forall \lambda \in$ *completion*(π), λ définit une séquence d'états $\langle s_0, \ldots, s_n \rangle$ telle que $s_i = (s_{i-1} - \textit{effects}^-(a_{i-1})) \cup \textit{effects}^+(a_{i-1})$ car *assump*$(\pi) = \emptyset$. Comme il n'y a pas de réfutation dans π, $g \subseteq s_n$. Cela se démontre par l'absurde. Supposons qu'il existe $p \in g$ et $p \notin s_n$. Comme p ne peut être une hypothèse ($\exists (a_i \xrightarrow{p} a_n)$), l'absence de p dans s_n est due à une réfutation d'après le lemme 4.1. Ceci est contradictoire avec l'absence de menace. Par conséquent, $g \subseteq s_n$ et π est un plan-solution.

5. Modèle de composition de services web par planification distribuée

La section précédent a permis d'entrevoir les mécanismes nécessaires à la mise en œuvre de la composition de services web au travers des définitions de plans partiels et de réfutations. Dans cette section, il s'agit maintenant de formaliser la dynamique du modèle de composition permettant à un groupe d'agents de raisonner conjointement à l'élaboration d'un plan-solution. D'une part, les agents doivent être capables d'interagir en respectant un certain nombre de règles qui garantissent le bien fondé du raisonnement produit et, d'autre part, ils doivent être capables de démontrer la validité des hypothèses formulées par les autres agents, de réfuter les plans partiels incorrects ou encore de les réparer lorsque ceux-ci ont été précédemment réfutés. Par conséquent, il faut distinguer deux types de mécanismes : les mécanismes qui spécifient quand un agent peut interagir, *e.g.*, pour réfuter un plan partiel, et les mécanismes qui servent de support à la production du contenu de l'interaction.

5.1. *Principe*

Chaque agent possède un tableau (Englemore *et al.*, 1988; Jagannathan *et al.*, 1989) dans lequel il enregistre les propositions des autres agents. D'un point de vue algorithmique, le tableau peut être vu comme un graphe orienté dont les nœuds représentent des plans partiels. Chaque arête sortant d'un nœud π est un opérateur qui transforme un plan partiel π en un plan partiel successeur π', traduisant ainsi les modifications proposées par les agents en termes de raffinement, de réfutation et de réparation. Par conséquent, le tableau définit l'état des interactions entre agents mais également l'espace de recherche co-construit par les agents.

Les différents actes de dialogue utilisés dans notre approche sont donnés par le tableau 2. Ils se regroupent en deux niveaux : *les actes de niveau informationnel* qui permettent aux agents d'échanger des informations sur les plans partiels contenus dans leurs tableaux et *les actes de niveau contextualisation* qui permettent de modifier le contexte de l'interaction. C'est par l'intermédiaire de ces actes que les agents vont pouvoir débuter ou encore suspendre l'élaboration d'un plan-solution.

Classiquement, le tableau de chaque agent est initialisé avec un plan partiel π_0 défini par :

Niveau	**Actes**
Informationnel	*refine*, *refute*, *repair*, *failure*
Contextualisation	*prop.solve*, *prop.failure*, *prop.success*, *ack.failure*, *ack.success*

Tableau 2. *Tableau des actes de dialogue classés par niveau*

– deux actions a_0, a_∞ telles que les préconditions de a_∞ représentent le but g soumis à l'ensemble des agents et les effets de a_0 l'état initial des croyances de l'agent ;

– une relation d'ordre $(a_0 \prec a_\infty)$;

– les contraintes d'instanciation relatives à la description des préconditions et des effets de a_0, a_∞ ;

– un ensemble de liens causaux vide.

π_0 n'est pas un plan solution car le but n'est soutenu par aucune relation causale. Les agents vont donc raffiner ce plan par l'ajout d'autres services (actions dans notre terminologie) dont les effets sont produits pour réaliser le but. Ces services peuvent eux-mêmes avoir des préconditions non soutenues causalement ce qui induit de nouveaux raffinements ou provoquer des réfutations avec d'autres services qui nécessitent des réparations. Nos algorithmes garantissent que ce processus converge vers un plan solution lorsqu'il existe.

5.2. *Les règles d'interactions*

La formalisation des règles d'interaction est donnée par le tableau 3. Pour les caractériser de manière formelle, nous définissons pour chaque acte du niveau informationnel trois sous-ensembles de règles :

1) *les règles de rationalité* qui définissent les propriétés que doit vérifier le contenu informationnel de l'acte pour être valide au sens du dialogue ;

2) *les règles du dialogue* qui spécifient les actes de dialogue autorisés pour garantir la cohérence du raisonnement produit par les agents ;

3) *les règles de mise à jour* qui précisent comment le tableau doit être modifié en fonction des actes de dialogue.

5.3. *Les règles de contextualisation*

Les règles du niveau informationnel ne suffisent pas à définir totalement les interactions entre agents. Il reste encore à préciser comment les agents débutent et clôturent la synthèse d'un plan. Nous proposons de formaliser l'ensemble des règles de contextualisation par un automate à états finis (*cf.* figure 4). Les états de l'automate

refine(ρ, p, π ***)*** où ρ est un raffinement et p l'hypothèse raffinée du plan partiel π.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau ; (ii) p est une hypothèse formulée par π ; (iii) ρ n'a pas déjà été proposé comme raffinement de p et (iv) le plan partiel résultat π' de l'application de l'opérateur de raffinement ρ est valide au sens où ses contraintes d'ordre et d'instanciation sont cohérentes.

Dialogue : les agents peuvent raffiner toutes les hypothèses de π' ou réfuter π'.

Mise à jour : ajouter π' comme raffinement de l'hypothèse p de π dans son tableau.

refute(ϕ, π ***)*** où ϕ est une réfutation et π le plan partiel réfuté.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau et (ii) ϕ n'a pas déjà été proposée comme réfutation à l'encontre de π.

Règles : les agents peuvent réparer π.

Mise à jour : ajouter ϕ comme réfutation de π dans le tableau.

repair(ψ, ϕ, π ***)*** où ψ est une réparation de π en réponse à la réfutation ϕ.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau ; (ii) ϕ est une réfutation formulée à l'encontre de π ; (iii) ψ n'a pas déjà été proposée comme réparation de π en réponse à la réfutation ϕ et (iv) le plan partiel résultat π' de l'application de ψ est valide au sens où ses contraintes d'ordre et d'instanciation sont cohérentes.

Règles : les autres agents peuvent raffiner toutes les hypothèses de π' ou réfuter π'.

Mise à jour : ajouter π' comme réparation de la réfutation ϕ de π.

failure(Φ,π ***)*** où Φ est une menace, *i.e.*, une hypothèse ou une réfutation de π.

Rationalité : l'agent doit vérifier que : (i) π est un plan partiel présent dans son tableau et (ii) Φ est une hypothèse ou une réfutation formulée à l'encontre de π.

Règles : $\emptyset$

Mise à jour : marquer Φ comme ne pouvant être résolue par l'agent qui a énoncé l'acte.

Tableau 3. *Règles d'interaction du niveau informationnel*

définissent les états des interactions et les arêtes les différents actes de dialogues du tableau 2. À chaque réception ou émission d'un acte par un autre agent, l'automate tient à jour l'état courant du dialogue. Nous utilisons la notation *?* et *!* devant un acte de dialogue pour indiquer que l'acte est respectivement reçu (*e.g.*, *?refine()*) ou émis (*e.g.*, *!refine()*) par un agent.

Essayons maintenant d'étudier plus en détail l'automate de contextualisation. Initialement, les agents sont dans un état **IDLE**. Cet état correspond à une attente des agents d'un but à résoudre. À la réception ou à l'émission de l'acte *prop.solve*, le dialogue passe dans l'état **Planning** dans lequel les agents échangent des raffinements, des réfutations ou encore des réparations qui sont stockés dans leur tableau en fonction des règles définies par le niveau informationnel (*cf.* tableau 3). Cet état repésente la phase de composition des différents services web. Notons que l'acte *prop.solve* peut être énoncé, soit par un opérateur humain, soit par un agent qui souhaite soumettre à un groupe d'agents un but à résoudre. La phase d'ouverture du dialogue est équivalente dans les deux cas. Le contenu informationnel de l'acte *prop.solve* représente les buts soumis groupe d'agents. Lorsque le dialogue est ouvert, les agents sont dans l'état **Planning**. La sortie de cet état peut se produire dans deux cas :

1) *sur proposition de l'agent.* Lorsque l'agent initie une proposition de sortie sur échec (resp. sur succès), l'agent énonce l'acte *prop.failure* (resp. *prop.success*). Le dialogue passe alors dans un état **Failure** (resp. **Success**) dans lequel l'agent attend les acquittements locaux des autres agents. Si tous les agents acquittent la proposition alors les conditions de terminaison sur échec (resp. sur succès) sont vérifiées ;

2) *sur proposition d'un autre agent,* c'est-à-dire lorsqu'un agent reçoit d'une proposition de sortie sur échec, *prop.failure*, ou respectivement à la réception d'une proposition de sortie sur succès, *prop.success*. La fermeture du dialogue sur proposition d'un autre agent nécessite un découpage en deux phases :

a) *une phase de vérification.* Lorsqu'un de ces deux actes est reçu, le dialogue passe en instance de sortie, représenté par les états **IF** (Instance Failure) et **IS** (Instance Success). Ces deux états du dialogue correspondent à la vérification locale des propriétés de sortie de dialogue. Dans le cas d'une proposition de sortie sur échec, l'agent doit vérifier qu'il n'existe pas localement de solution au problème initialement soumis au groupe d'agents et dans le cas d'une proposition de sortie sur succès que le plan partiel constituant le contenu informationnel de l'acte *prop.success* vérifie bien localement les propriétés d'un plan-solution définies par la proposition 4.1 ;

b) *une phase d'acquittement.* Si les propriétés locales de terminaison sur échec (resp. succès) sont vérifiées alors l'agent acquitte la proposition de sortie en énonçant *ack.failure* (resp. *ack.success*), et le dialogue passe dans l'état **Failure** (resp. **Success**). Les états **Failure** et **Success** correspondent à l'attente par un agent des acquittements des autres agents. Notons que le non-acquittement par un agent de la proposition de sortie ne se traduit pas par un acte spécifique du niveau de contextualisation. En effet, supposons qu'un agent soit dans l'état **IF** (*i.e.*, Instance Failure), si l'agent reçoit ou soumet un raffinement, une réfutation ou une réparation, cela signifie qu'un des agents est capable de poursuivre le processus de synthèse de plans. Les agents doivent

par conséquent réévaluer le tableau puisque celui-ci a été modifié. De manière similaire, lorsqu'un agent se trouve dans l'état **IS** (*i.e.*, Instance Success), si l'agent reçoit ou soumet une réfutation portant sur le plan partiel proposé comme solution, le plan partiel précédemment proposé comme solution est invalidé et l'agent doit à nouveau poursuivre le processus de synthèse de plans. Dans les deux cas, tous les agents impliqués dans le dialogue sont contraints à poursuivre le processus de synthèse de plans ce qui se traduit par un retour dans l'état **Planning** de l'automate de contextualisation.

L'automate de contextualisation garantit que la terminaison du processus de synthèse de plans ne peut avoir lieu que si l'ensemble des agents sont dans l'incapacité de faire progresser la co-construction d'un plan ou alors qu'un plan-solution est présent dans le tableau. Ceci revient à calculer un état global pour déterminer si l'une des conditions de sortie est vérifiée.

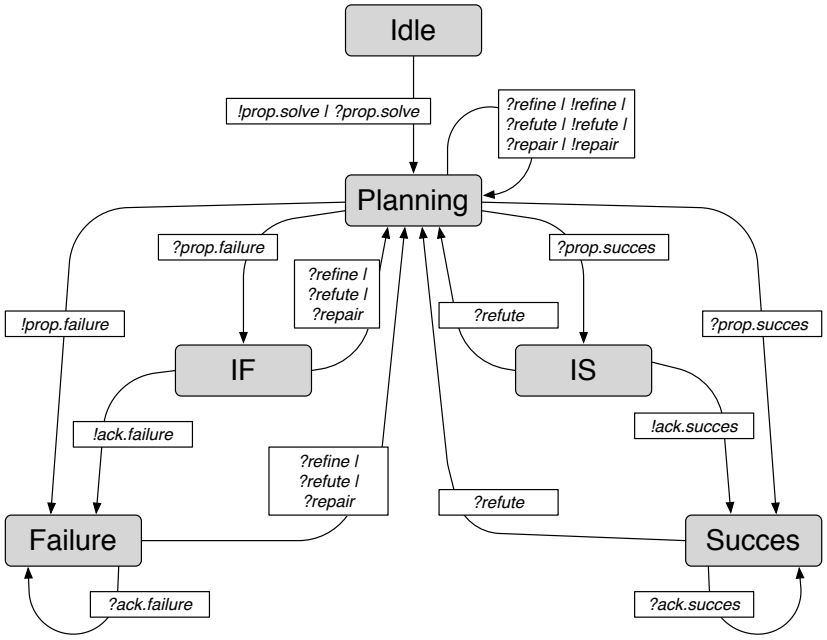

Figure 4. *Automate de contextualisation des interactions*

5.4. *La boucle de raisonnement*

La définition des règles qui régissent le dialogue est nécessaire mais ne spécifie pas les mécanismes qui guident l'interaction. Pour répondre à cette question, nous proposons d'introduire la boucle de raisonnement qui permet aux agents d'élaborer conjointement un plan. À chaque itération, un agent choisit un plan partiel déjà énoncé et cherche à lui appliquer un raffinement, une réfutation ou une réparation fai-

sant ainsi progresser collectivement la recherche d'un plan-solution. La procédure de raisonnement d'un agent peut être schématisée par l'exécution itérative des opérations suivantes :

1) Choix d'un plan partiel π non encore exploré dans son tableau ;

2) Choix de l'opération à appliquer à π, *i.e.*, raffiner, réfuter ou réparer ;

3) Calcul des modifications à apporter à π ;

4) Soumission des modifications aux autres agents.

Le raisonnement se poursuit tant que les tableaux des différents agents ne contiennent pas au moins un plan partiel qui respecte les propriétés d'un plan-solution (terminaison sur succès *cf.* proposition 4.1) ou alors qu'aucun agent ne peut plus modifier un plan partiel déjà proposé et ainsi faire progresser la synthèse de plans (terminaison sur échec). Lorsque l'une de ces deux conditions de terminaison est vérifiée, tous les agents mettent fin à leur raisonnement.

La boucle de raisonnement qui décrit le raisonnement d'un agent est donnée en pseudo-code par l'algorithme 1. La première opération effectuée par la boucle de raisonnement est de choisir un plan partiel à partir de son tableau. Cette sélection ne peut porter que sur les plans partiels pour lesquels l'agent peut encore proposer des raffinements, des réfutations ou des réparations. Si l'agent ne peut plus faire progresser la synthèse de plan (ligne 3), alors il doit soumettre aux autres agents une proposition de terminaison sur un échec et mettre fin à sa propre boucle de raisonnement (ligne 5). La proposition de terminaison est alors soumise aux autres agents pour acceptation.

Soulignons que le choix du plan partiel à explorer est important pour garantir de bonnes performances à l'algorithme de synthèse de plans. En effet, c'est de ce choix local que découle la politique globale d'exploration de l'espace de plans partiels. Pour plus de détails sur ces aspects, le lecteur peut se référer aux travaux de (Gerevini *et al.*, 1998).

Maintenant qu'un plan partiel a été sélectionné, l'agent doit déterminer si ce plan partiel est un plan-solution. Rappelons qu'un plan partiel π est un plan-solution (ligne 8) si π ne formule plus aucune hypothèse et ne peut pas être réfuté. La première propriété est vérifiée par la procédure `Assumptions(`π`)` qui calcule l'ensemble des hypothèses présentes dans un plan partiel. Cette procédure peut être implémentée de manière efficace en utilisant une table de hachage : à chaque ajout d'une nouvelle action a au plan partiel, les hypothèses de a sont ajoutées à la table et, à chaque ajout d'un lien causal au plan partiel, les hypothèses correspondantes sont supprimées de la table. La seconde propriété est vérifiée par la procédure `Refutations(`π`)` qui calcule l'ensemble des réfutations pouvant être formulées à l'encontre du plan partiel (*cf.* §6.2). Lorsqu'un plan partiel π vérifie localement ces deux propriétés (ligne 9), l'agent doit proposer à l'ensemble des autres agents de terminer le processus de synthèse de plans en soumettant π comme plan-solution. De manière symétrique aux mécanismes de terminaison sur échec, l'agent met fin à son raisonnement. Sinon l'agent doit pour-

suivre son raisonnement en proposant un nouveau raffinement ou encore une nouvelle réfutation ou réparation.

Considérons tout d'abord qu'un agent décide de proposer un nouveau raffinement portant une hypothèse de π (ligne 14). L'agent calcule les raffinements possibles par l'intermédiaire de la procédure `Refine(`ϕ, π`)` qui prend en paramètres l'hypothèse ϕ à raffiner ainsi que le plan partiel π. Si aucun raffinement ne peut être proposé, alors ϕ est marquée comme ne pouvant être résolue et l'agent propage son échec. Sinon, l'agent choisit un raffinement et le soumet à la délibération des autres agents. Le même mécanisme s'applique pour les réparations (ligne 24). Si ϕ est une réfutation qui a été précédemment formulée à l'encontre d'un plan partiel, π doit être réparée. L'agent calcule alors les réparations possibles pour la réfutation grâce à la procédure `Repair(`ϕ, π`)`. Si aucune réparation n'a été produite, l'agent marque la réfutation comme ne pouvant être résolue et l'agent propage son échec. Dans le cas contraire, l'agent informe les autres agents des réparations produites.

Finalement, si ϕ est une réfutation (ligne 35), l'agent soumet simplement la réfutation aux autres agents afin de leur indiquer que dorénavant le plan partiel π n'est pas correct et doit être réparé.

Proposition 5.1 La boucle de raisonnement est correcte et complète : chaque fois qu'il existe, du point de vue d'un agent, une solution pour $\mathcal{P} = (s_0, \mathcal{O}, g)$, cet agent termine sa boucle de raisonnement en faisant une proposition de succès avec un plan-solution.

Preuve 5.1 (Correction) Par définition, le plan partiel initial $\pi_0 = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ est défini par $\mathcal{A} = \{a_0, a_\infty\}$, $\prec$ un ensemble de contraintes d'ordre cohérent et $\mathcal{I}$ un ensemble de contraintes d'instanciation cohérent. Lorsque la boucle de raisonnement fait une proposition de succès, le plan partiel π proposé ne contient plus de menace et tous les opérateurs de raffinements et de réparation maintiennent $\prec$ et $\mathcal{I}$ cohérents. Ceci est garanti par les règles du dialogue. Par conséquent, d'après la proposition 4.1, π est un plan-solution.

Le raisonnement est également correct au niveau du groupe d'agents car la boucle de raisonnement ne s'arrête que si et seulement si un agent a initié une proposition de succès sur π (π vérifie localement les propriétés d'un plan-solution pour l'agent initiateur) et que π n'est pas réfutable par les autres agents. Par conséquent, π est un plan-solution pour $\mathcal{P}$ au niveau du groupe.

Preuve 5.2 (Complétude) Il faut montrer qu'au moins une des traces d'exécution de la procédure de raisonnement renvoie un plan-solution d'un point de vue d'un agent lorsqu'il existe. Preuve par induction, sur la longueur k d'un plan-solution.

Cas de base $(k = 0)$ *:* le plan vide π est solution de $\mathcal{P}$. Par conséquent, π ne comporte pas de menace et la procédure de raisonnement fait immédiatement une proposition de succès.

Algorithm 1: Boucle de raisonnement

```
1  while raisonne = true do
2      plans ← l'ensemble des plans partiels restant à explorer ;
3      if plans est vide then
4          Soumettre une proposition d'échec ;
5          raisonne ← false ;
6      else
7          Sélectionner un plan partiel π ∈ plans ;
8          menaces ← Assumptions(π) ∪ Refutations(π) ;
9          if menaces est vide then
10             Soumettre une proposition de succès portant sur π ;
11             raisonne ← false ;
12         else
13             Sélectionner une menace φ ∈ menaces ;
14             if φ est une hypothèse then
15                 raffinements ← Refine(φ, π) ;
16                 if raffinements est vide then
17                     Marquer φ comme ne pouvant être résolue ;
18                     Soumettre l'échec de raffinement portant sur φ ;
19                 else
20                     Sélectionner un raffinement ρ ∈ raffinements ;
21                     Soumettre ρ comme raffinement φ de π;
22             else if φ est une réfutation déjà formulée à l'encontre de π then
23                 reparations ← Repair(φ, π) ;
24                 if reparations est vide then
25                     Marquer φ comme ne pouvant être résolue ;
26                     Soumettre l'échec de réparation portant sur φ ;
27                 else
28                     Sélectionner une réparation ψ ∈ reparations ;
29                     Soumettre ψ comme réparation de φ de π;
30             else Soumettre φ comme nouvelle réfutation de π ;
```

Induction : supposons que la procédure de raisonnement soit complète pour un problème nécessitant une solution de longueur k. Soit $\mathcal{P} = (s_0, \mathcal{O}, g)$ nécessitant un plan-solution $\langle a_0, \ldots, a_k \rangle$ de longueur $k+1$. Comme a_k est nécessaire pour démontrer le but g, il existe au moins un agent qui proposera de manière non déterministe un opérateur de raffinement, de réfutation ou de réparation à appliquer au plan partiel initial π_0 contenant a_k. Soit $s_{k+1} = ((s_k \cup \mathit{assump}(a_k)) - \mathit{effects}^-(a_k)) \cup \mathit{effects}^+(a_k)$, l'état suivant l'exécution de a_k. Au prochain appel récursif de la procédure par les agents le plan partiel

π_1 résultat de l'opérateur contenant a_k est constitué d'au moins trois actions $\{a_0, a_k, a_\infty\}$. π_1 est équivalent au plan partiel initial d'un problème défini par l'état s_0 et $g = \textit{assump}(a_k) \cup \textit{assump}(a_\infty)$. Ce problème admet une solution $\langle a_0, \ldots, a_j \rangle$ de longueur $j \leq k$. Par hypothèse d'induction, les appels récursifs de la procédure sur le plan partiel π_1 produisent une trace qui trouve le plan-solution $\langle a_0, \ldots, a_j \rangle$ et la procédure propose de sortir sur succès.

6. Les opérateurs de raffinement, de réfutation et de réparation

Cette section présente les mécanismes nécessaires au calcul des opérateurs de raffinement, de réfutation et de réparation.

6.1. *Les mécanismes de raffinement*

L'opérateur de raffinement est utile pour démontrer qu'une hypothèse peut être vérifiée. Nous distinguons deux mécanismes de raffinement : les raffinements par ajout d'un lien causal et les raffinements par ajout d'un sous-plan partiel.

6.1.1. *Les raffinements par ajout d'un lien causal*

Le premier mécanisme de raffinement consiste à ajouter un lien causal. Cette technique de raffinement est la plus simple. S'il existe déjà au sein du plan partiel π une action a_i qui a pour effet p alors il suffit d'ajouter au plan partiel π un lien causal $(a_i \xrightarrow{p} a_j)$ indiquant dorénavant que l'action a_i démontre l'hypothèse p formulée par l'action a_j. Un exemple de ce type de raffinement est donné à la figure 5. Le raffinement solution est un tuple constitué d'un lien causal, d'une contrainte d'ordre, $(a_i \prec a_j)$, ainsi que des contraintes d'instanciation σ nécessaires à l'unification de p avec les effets de a_i.

L'intérêt de ce type de raffinement est de minimiser l'ajout de nouvelles actions au plan partiel et ainsi de réduire le nombre d'hypothèses introduites. Si l'on fait un rapprochement avec les travaux sur les interactions entre plans de (Martial, 1992), ce mécanisme peut être vu comme une technique de prise en compte des relations favorables[6] entre les actions proposées par les différents agents. En effet, le plan partiel construit est plus efficace puisque le raffinement conduit à la réutilisation d'une action déjà présente. On évite ainsi l'ajout d'actions redondantes au cours du processus de synthèse de plans.

6.1.2. *Les raffinements par ajout d'un sous-plan partiel*

Le second mécanisme de raffinement consiste à ajouter un sous-plan partiel pour démontrer la validité d'une hypothèse. Rappelons qu'une hypothèse p est considérée

6. Dans la mesure où la ressource est partageable et non consommable.

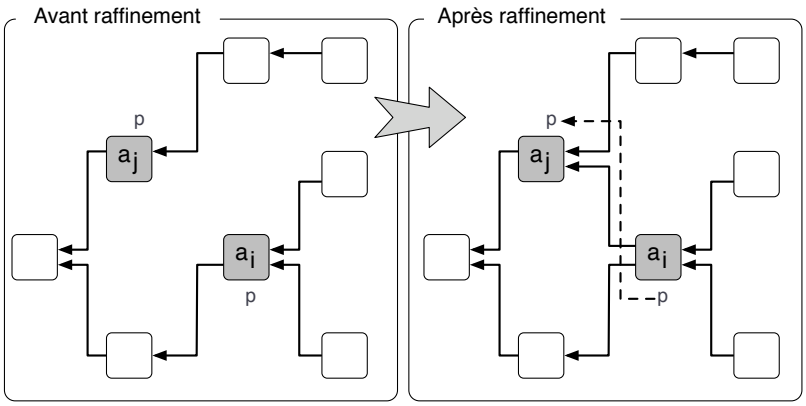

Figure 5. *Exemple de raffinement lorsqu'une action déjà contenue dans le plan partiel permet de démontrer une hypothèse p*

comme un sous-but à atteindre pour les autres agents. Le raffinement solution contient le sous-plan partiel produit π', un lien causal, $(a_i \xrightarrow{p} a_j)$ permettant de spécifier quelle action a_i de π' démontre l'hypothèse p, et une contrainte d'ordre $(a_i \prec a_j)$. La figure 6 montre un exemple de raffinement par ajout d'un sous-plan partiel pouvant lui-même contenir des hypothèses.

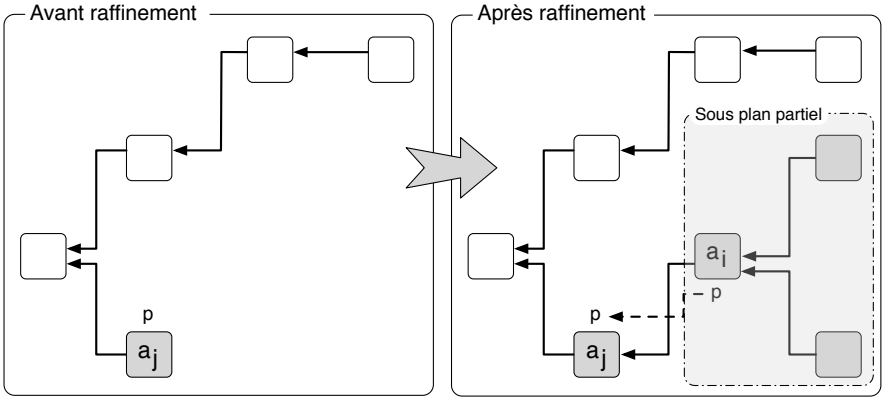

Figure 6. *Exemple de raffinement par ajout d'un sous-plan partiel*

L'algorithme mis en œuvre pour produire les raffinements, *i.e.*, les sous-plans partiels démontrant la validité d'une ou plusieurs hypothèses, repose sur une technique particulière de planification appelée *Hierarchical Task Network* (Nau *et al.*, 2003). En comparaison des techniques de planification classiques, le principal avantage de HTN réside dans l'expressivité de son langage et ses capacités d'extension, *e.g.*, en permettant l'ajout de procédures externes dans les préconditions des opérateurs, en ajoutant

un mécanisme d'inférence etc. Cette technique permet également de modéliser et de résoudre des problèmes que les planificateurs classiques ne peuvent pas résoudre (Erol *et al.*, 1994). En outre, avec une bonne modélisation pour guider la recherche d'un plan solution qui peut être soit partiellement soit totalement ordonné (cas considéré dans nos travaux), cette technique peut résoudre la plupart des problèmes classiques de planification beaucoup plus rapidement que des techniques telles que la recherche dans un espace d'états ou de plans. Le principal inconvénient de HTN réside dans la nécessité pour l'auteur d'un domaine de planification de spécifier non seulement les opérateurs de planification mais également les méthodes. Ce désavantage reste toutefois assez limité dans le contexte de la spécification de services web dans la mesure où les services sont déjà décrits comme la décomposition de tâches complexes.

Dans le cadre de planification HTN, le but à atteindre n'est plus un ensemble de propriétés du monde qui doivent être vérifiées mais une séquence d'actions primitives ou complexes à réaliser à partir de l'état initial de croyance des agents. La production d'un plan peut alors s'exprimer comme la décomposition récursive d'actions complexes en sous-actions jusqu'à ce que toutes les actions soient des actions *primitives*, *i.e.*, pouvant être exécutées en appliquant un opérateur (*cf.* définition 4.1). Pour passer de la représentation d'un but défini comme un ensemble de propositions à celle d'une actions à réaliser, il suffit de décrire une méthode particulière qui prend en paramètre le but à atteindre.

La définition d'un raffinement peut s'exprimer de la manière suivante.

Définition 6.1 (Raffinement) Soit un problème de planification $\mathcal{P} = (s_0, \mathcal{O}, \mathcal{M}, \langle \alpha_0, \ldots, \alpha_n \rangle)$. Un plan partiel π est un raffinement de $\mathcal{P}$ si toutes les linéarisations $\lambda \in \textit{completion}(\pi)$ raffinent $\mathcal{P}$. Soit $\lambda = \langle a_0, \ldots, a_k \rangle$ une séquence d'actions primitives, λ raffine $\mathcal{P}$ si l'une des conditions suivantes est vérifiée :

Cas 1 : la séquence d'actions à réaliser est vide alors λ est la séquence vide ;

Cas 2 : α_0 est primitive : α_0 est identique à a_0, α_0 est applicable à partir de s_0 et $\lambda = \langle a_1, \ldots, a_k \rangle$ raffine $(s_1, \mathcal{O}, \mathcal{M}, \langle \alpha_1, \ldots, \alpha_n \rangle)$;

Cas 3 : α_0 est complexe : il existe une réduction $\langle r_1, \ldots, r_j \rangle$ applicable pour réaliser α_0 à partir de s_0 et λ raffine $(s'_0, \mathcal{O}, \mathcal{M}, \langle r_1, \ldots, r_j, \alpha_1, \ldots, \alpha_n \rangle)$ avec $s'_0 = s_0 \cup \textit{assump}(\alpha_0)$.

La procédure qui produit les raffinements se découpe en deux phases : l'expansion de l'arbre de raffinements (détaillée dans la suite de cette section) et l'extraction d'un plan partiel qui consiste à parcourir une branche de l'arbre d'une feuille solution au nœud racine en initialisant les actions, les contraintes d'ordre, les contraintes d'instanciation et les liens causaux du raffinement. La construction de l'arbre de raffinements consiste à décomposer récursivement les actions complexes contenues dans les nœuds de l'arbre jusqu'à ce qu'une feuille soit produite, *i.e.*, un nœud possédant une séquence d'actions vide. Cette feuille représente l'état du monde qui sera atteint après l'exécution du raffinement.

L'algorithme d'expansion de l'arbre de raffinements s'appuie sur la définition 6.1. Initialement, la procédure d'expansion de l'arbre est exécutée avec le nœud représentant l'état initial des croyances de l'agent s_0 et la séquence des actions à réaliser $\langle \alpha_0, \ldots, \alpha_n \rangle$. La procédure commence par tester si $\langle \alpha_0, \ldots, \alpha_n \rangle$ est la séquence vide. Dans ce cas, une feuille de l'arbre est atteinte et le nœud n est retourné comme nœud solution (*cf.* cas 1 définition 6.1). Sinon la procédure essaie de réaliser la première action α_0. Deux cas sont à envisager selon que α_0 est :

– *une action primitive.* Pour chaque opérateur contenu dans $\mathcal{O}$, la procédure teste s'il peut réaliser l'action α_0. Si c'est le cas, la procédure calcule les substitutions unifiant les préconditions de l'opérateur avec l'état courant s contenu dans le nœud n. Finalement, pour chaque substitution σ, l'algorithme ajoute un nœud fils au nœud n (*cf.* cas 2 définition 6.1) ;

– *une action complexe.* La procédure repose sur le même principe que pour une action primitive. Cependant elle teste cette fois, non plus les opérateurs mais les méthodes qui permettent de réaliser α_0 et qui sont applicables dans s. Pour chacune de ces méthodes un nouveau nœud fils est ajouté au nœud n (*cf.* cas 3 définition 6.1).

Finalement, la procédure choisit de manière non déterministe un nouveau nœud parmi les nœuds fils de n et s'appelle récursivement avec comme paramètre le nouveau nœud sélectionné.

6.2. *Les mécanismes de réfutation*

En toute généralité, les mécanismes de réfutation cherchent à démontrer qu'un plan partiel est incorrect. Dans notre approche, un plan partiel est incorrect s'il contient des séquences d'actions non valides. Nous avons défini une réfutation (*cf.* définition 4.10) comme un couple $(a_k, a_i \xrightarrow{p} a_j)$ indiquant que l'action a_k a pour effet q tel que q supprime une précondition p nécessaire à l'exécution de a_j. Autrement dit, le calcul des réfutations consiste à déterminer les actions a_k qui invalident un lien causal $(a_i \xrightarrow{p} a_j)$ (*cf.* figure 3).

6.3. *Les mécanismes de réparation*

Les mécanismes de réparation calculent les modifications à apporter aux plans partiels lorsque ceux-ci ont été préalablement réfutés. Nous distinguons les réparations par ajout de contraintes d'ordre et par ajout d'un sous-plan partiel.

6.3.1. *Les réparations par ajout de contraintes d'ordre*

Le premier type de réparation consiste à introduire une contrainte d'ordre entre a_k et les actions a_i et a_j. En effet, si une réfutation a été formulée à l'encontre d'un plan partiel, c'est parce que a_k supprime une propriété du monde nécessaire à l'exécution de a_j. Il faut donc s'assurer que a_k ne puisse pas être exécutée entre les actions a_i et a_j, *i.e.*, $(a_i \prec a_k \prec a_j)$. Pour garantir que cela ne peut pas se produire, il faut ajouter soit une contrainte d'ordre $(a_j \prec a_k)$, imposant que l'exécution de a_k soit

réalisée après a_j, soit une contrainte d'ordre $(a_k \prec a_i)$, imposant que a_k s'exécute avant a_i (*cf.* figure 7). L'existence préalable de contraintes d'ordre dans le plan partiel entre l'action a_k et les actions a_j et a_i contraignent les choix possibles. Le tableau 4 récapitule les réparations par ajout de contraintes d'ordre en fonction des contraintes qui lient a_k avec les autres actions.

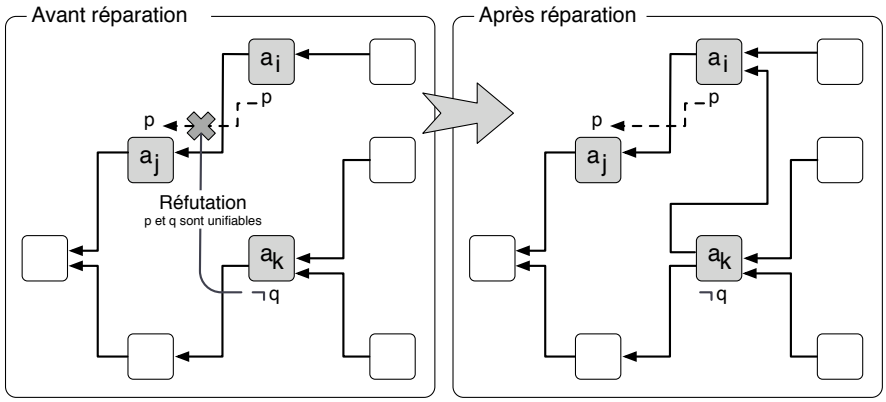

Figure 7. *Exemple de réparation par ajout de contraintes d'ordre*

Contraintes existantes sur a_k	**Contraintes à ajouter**
$\emptyset$	$a_j \prec a_k$ ou $a_k \prec a_i$
$a_k \prec a_j$	$a_k \prec a_i$
$a_i \prec a_k$	$a_j \prec a_k$
$a_i \prec a_k$ et $a_k \prec a_j$	pas de solution

Tableau 4. *Taxonomie des réparations par ajout d'une contrainte d'ordre*

6.3.2. *Les réparations par ajout d'un sous-plan partiel*

Étant donné que l'action a_k supprime une propriété p nécessaire à l'exécution de l'action a_j, la réparation consiste à ajouter un sous-plan partiel permettant de produire p après l'exécution de a_k. En ce sens, les réparations par ajout de sous-plans partiels peuvent être vues comme le raffinement d'une hypothèse implicite formulée par l'action a_k. En effet, en proposant l'action a_k, un agent suppose qu'un autre agent sera capable de recréer la propriété q. Par conséquent, ce type de réparation qui repose sur la production de raffinements vérifiant p (*cf.* §6.1.2) participe à la mise en œuvre de la coopération entre les agents.

Exemple 6.1 Considérons l'exemple donné par la figure 8. L'action a_k réfute le lien causal $(a_i \xrightarrow{p} a_j)$ car l'effet p est unifiable avec q. L'ajout du sous-plan partiel (ne contenant que l'action a_l) produit l'effet q après sa suppression par a_k. Par conséquent,

q est vérifiée dans l'état précédant a_j (a_j peut être exécutée) et a_k ne réfute plus ($a_i \xrightarrow{p} a_j$).

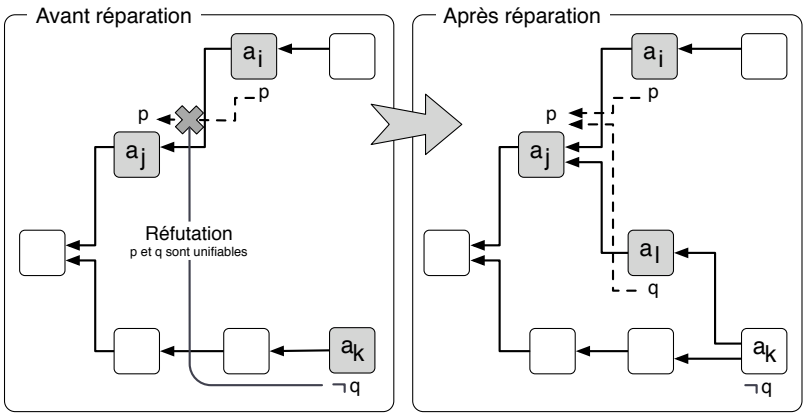

Figure 8. *Exemple de réparation par ajout d'un sous-plan partiel*

7. Conclusion

Dans cet article, nous avons présenté une étude sur une architecture totalement distribuée de composition automatique de services web par des techniques de planification. Cela se justifie par la nature intrinsèquement distribuée d'internet, la plate-forme d'exécution des services web. La problèmatique abordée ici est de comprendre les mécanismes qui permettent à un groupe d'agents autonomes de composer collectivement et dynamiquement un ensemble de services. La contribution de nos travaux repose sur la conception d'un modèle de planification entièrement distribué dans lequel les agents raisonnent conjointement sur leurs services respectifs pour atteindre un but commun prédéfini par l'utilisateur en intégrant leurs croyances partielles sur le monde. Dans cette perspective, nous proposons de considérer la synthèse de plans comme un raisonnement collectif et révisable fondé sur l'échange entre les agents de conjectures, *i.e.*, des plans qui peuvent être exécutés si certaines conditions sont vérifiées et de réfutations, *i.e.*, des objections quant à la réalisation du plan. Les interactions entre agents peuvent se définir comme un processus dialectique d'investigation dans lequel les agents proposent leurs services pour démontrer la validité de certaines hypothèses, réfuter les conjectures non réalisables, réparer lorsque cela est possible les conjectures précédemment réfutées et ainsi élaborer pas à pas un plan solution représentant une composition possible de leurs services.

L'étape suivante de notre étude est la mise en place à partir de services web déployés sur internet de critères de validation : en premier lieu le passage à l'échelle et le temps de réponse fonction du nombre des services composés. La complexité intrinsèque du processus de planification distribuée sur laquelle repose notre approche

de composition automatique la limite pour le moment à une classe de problèmes nécessitant une dizaine de services. Toutefois, il est possible d'envisager une généralisation de l'approche dans laquelle l'entité minimale ne serait plus un agent mais un groupe d'agents regroupés en fonction de structures organisationnelles statiques ou dynamiques, capables de composer des services de plus haut niveau d'abstraction. Cela aurait pour effet de contraindre les échanges entre agents. Enfin, il faut noter que les performances du processus global de composition peuvent être grandement améliorées en introduisant un certain nombre d'heuristiques que nous n'avons pas traitées dans cet article (Gerevini *et al.*, 1996; Nguyen *et al.*, 2001).

Un autre aspect important est la robustesse de l'architecture aux aléas d'exécution : est-il toujours possible de substituer des services fonctionnellement équivalents ? Quelle est l'incidence de la défaillance d'un service sur la composition calculée et l'éventuelle replanification à entreprendre ? À plus long terme, il s'agira de lever les hypothèses implicites des modèles de composition de services web en général. En particulier, rien ne garantit l'alignement ontologique (Euzenat *et al.*, 2007) des services web ; la composition est fondée sur des traitements syntaxiques et la description « sémantique » des services web implique que celle-ci soit partagée par tous les services invoqués. Un autre verrou fondamental des approches automatiques de composition est la capacité d'un expert à exprimer de façon simple et non ambiguë – ce qui est souvent contradictoire – l'objectif de cette composition. L'expression déclarative de la requête initiale doit être simple afin de décharger autant que faire ce peut l'utilisateur de la tâche cognitive de composition et rigoureuse afin de maximiser les performances des algorithmes de planification.

8. Bibliographie

Ankolekar A., Burstein M., Hobbs J., Lassila O., Martin D., McDermott D., McIlraith S., Narayanan S., Paolucci M., Payne T., Sycara K., « DAML-S : Web Service Description for the Semantic Web », *Proceedings of International Semantic Web Conference*, p. 348-363, 2002.

Benatallah B., Dumas M., Fauvet M., Rabhi F., « Towards patterns of web services composition », *Patterns and skeletons for parallel and distributed computing*, Springer Verlag, 2003.

Benatallah B., Dumas M., Sheng Q., « Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services », *Journal of Distributed and Parallel Databases*, vol. 17, n° 1, p. 5-37, 2005.

Bourdon J., *« Multi-agent systems for the automatic composition of semantic web services in dynamic environments »*, Rapport de master, École des Mines de Saint Etienne - G2I & Université Joseph Fourier, 2007.

Chapman D., « Planning for conjunctive goals », *Artificial Intelligence*, vol. 32, n° 3, p. 333-377, 1987.

Englemore R., Morgan T., *Blackboard Systems*, Addison Wesley, 1988.

Erol K., Hendler J., Nau D., « HTN Planning : Complexity and Expressivity », *Proceedings of the National Conference on Artificial Intelligence*, p. 1123-1128, 1994.

Euzenat J., Shvaiko P., *Ontology Matching*, Springer-Verlag, 2007.

Gerevini A., Schubert L., « Accelerating Partial-Order Planners : Some Techniques for Effective Search Control and Pruning », *Journal of Artificial Intelligence Research*, vol. 5, n° 1, p. 95-137, 1996.

Gerevini A., Schubert L., « Inferring State Constraints for Domain-Independent Planning », *Proceedings of the National Conference on Artificial Intelligence*, p. 905-912, 1998.

Ghallab M., Nau D., Traverso P., *Automated Planning, Theory and Practice*, Morgan Kaufmann, 2004.

Giacomo G. D., Lespérance Y., Levesque H., « ConGolog, a concurrent programming language based on the situation calculus », *Artificial Intelligence*, vol. 121, n° 1–2, p. 109-169, 2000.

Guitton J.*, « Planification multi-agent pour la composition dynamique de services web »*, Rapport de master, Université Joseph Fourier, 2006.

Hamadi R., Benatallah B., « Patterns and skeletons for parallel and distributed computing », *Proceedings of the Australasian Conference on Database*, p. 191-200, 2003.

Jagannathan V., Dodhiawala R., Baum L., *Blackboard Architectures and Applications*, Academic Press, 1989.

Jamal S., Environnement de procédé extensible pour l'orchestration - Application aux services web, Thèse de doctorat, Université Joseph Fourier, 2005.

Jayadev M., William C., « Computation Orchestration : A Basis for Wide-area Computing », *Journal of Software and Systems Modeling*, vol. 6, n° 1, p. 83-110, 2007.

Klusch M., Gerber M., M.Schmidt, « Semantic web service composition planning with OWLS-XPlan », *Proceedings of the International AAAI Fall Symposium on Agents and the Semantic Web*, 2006.

Levesque H., Reiter R., Lespérance Y., Lin F., Scherl R., « GOLOG : a logic programming language for dynamic domains », *Journal of Logic Programming*, vol. 31, n° 1–3, p. 59-84, 1997.

Martial F. V., *Coordinating plans for autonomous agents*, Springer Verlag, 1992.

Martin D., Burstein M., Hobbs J., Lassila O., McDermott D., McIlraith S., Narayanan S., Paolucci M., Parsia B., Payne T., Sirin E., Srini-vasan N., Sycara K., OWL-S : Semantic markup for web services, Technical report, France Telecom, MINDL Maryland, NIST, Nokia, 2004.

McCarthy J., Hayes P. J., *Readings in nonmonotonic reasoning*, Morgan Kaufmann, chapter Some philosophical problems from the standpoint of artificial intelligence, p. 26-45, 1987.

McIlraith S., Son T., « Adapting Golog for composition of semantic web services », *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, p. 482-496, 2002.

Medjahed B., Bouguettaya A., Elmagarmid A., « Composing Web Services on the Semantic Web », *The International Journal on Very Large Data Bases*, vol. 13, n° 2, p. 333-351, 2003.

Milanovic N., Malek M., « Current Solutions for Web Service Composition », *IEEE Internet Computing*, vol. 8, n° 6, p. 51-59, 2004.

Narayanan S., McIlraith S., « Simulation, verification and automated composition of web services », *Proceedings of the International Conference on World Wide Web*, p. 77-88, 2002.

Nau D., Au T., Ilghami O., Kuter U., Murdock W., Wu D., Yaman Y., « SHOP2 : An HTN planning system », *Journal of Artificial Intelligence Research*, vol. 20, n° 1, p. 379-404, 2003.

Nguyen X., Kambhampati S., « Reviving Partial Order Planning », *Proceedings of the International Conference on Artificial Intelligence*, p. 459-466, 2001.

Peer J., « A PDDL Based Tool for Automatic Web Service Composition », *Proceedings of the International Workshop on Principles and Practice of Semantic Web Reasoning*, p. 149-163, 2004.

Peer J., « A POP-Based Replanning Agent for Automatic Web Service Composition », *Proceedings of the European Conference on Semantic Web*, p. 47-61, 2005.

Peltz C., Web Services Orchestration - a review of emerging technologies, tools, and standards, Technical report, Hewlett Packard, 2003.

Penberthy J., Weld D., « UCPO : A sound, complete, partial order planner for ADL », *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, p. 103-114, 1992.

Pistore M., Barbon F., Bertoli P., Shaparau D., Traverso P., « Planning and Monitoring Web Service Composition », *Proceedings of the International Conference on Artificial Intelligence, Methodology, Systems, and Applications*, p. 106-115, 2004.

Pistore M., Marconi A., Bertoli P., Traverso P., « Automated Composition of Web Services by Planning at the Knowledge Level », *Proceedings of the International Joint Conference on Artificial Intelligence*, p. 1252-1259, 2005a.

Pistore M., Traverso P., Bertoli P., « Automated Composition of Web Services by Planning in Asynchronous Domains », *Proceedings of the International Conference on Planning and Scheduling*, p. 2-11, 2005b.

Singh M., Huhns M., *Service-Oriented Computing : Semantics, Processes, Agents*, Wiley and Sons, 2005.

Sirin E., Hendler J., Parsia B., « Semi-automatic Composition of Web Services using Semantic Descriptions », *Proceedings of the International Workshop on Web Services, Modeling, Architecture and Infrastructure (ICEIS 02)*, 2002.

Sirin E., Parsia B., WU D., Hendler J., Nau D., « HTN planning for web service composition using SHOP2 », *Journal of Web Semantics*, vol. 1, n° 4, p. 377-396, 2004.

Sycara K., Paolucci M., Ankolekar A., Srinivasan N., « Automated discovery, interaction and composition of Semantic Web services », *Journal of Web Semantics : Science, Services and Agents on the World Wide Web*, vol. 1, n° 1, p. 27-46, 2003.

Wolverton T., « Amazon opens web services shop », *CNET News*, 2002.

Wu D., Parsia B., Sirin E., Nau D., « Automating DAML-S Web Services Composition Using SHOP2 », *Proceedings of International Semantic Web Conference*, p. 195-210, 2003.