De l'intégration de données à la composition de services Web

Olivier Perrin

Nancy-Université
Université Nancy 2

École doctorale IAEM Lorraine
Département de formation doctorale en informatique

De l'intégration de données à la composition de services Web

Mémoire

présenté et soutenu publiquement le 23 juin 2010

pour l'obtention de l'

Habilitation à Diriger des Recherches
Nancy-Université – Université Nancy 2
Discipline Informatique

par

Olivier Perrin

Composition du jury

Rapporteurs :	Christine Collet	Professeur à l'Institut Polytechnique de Grenoble
	Djamal Benslimane	Professeur à l'Université Claude Bernard, Lyon 1
	Yamine Aït-Ameur	Professeur à l'Université de Poitiers
Examinateurs :	Marlon Dumas	Professeur à l'Université de Tartu, Estonie
	Farouk Toumani	Professeur à l'Université Blaise Pascal, Clermont-Ferrand
	Jean-François Mari	Professeur à l'Université de Nancy 2
	Claude Godart	Professeur à l'Université Henri Poincaré, Nancy

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Table des matières

Table des figures

Chapitre 1

Curriculum vitæ

1.1 État civil

Date et lieu de naissance : 13 juin 1966 à Nancy (Meurthe et Moselle)
Nationalité : française
Situation de famille : vie maritale, 2 enfants
Adresse professionnelle : LORIA-INRIA Lorraine, Technopôle de Nancy Brabois
615, rue du Jardin Botanique, 54602 Villers-lès-Nancy, France
Tel : +33 (0) 383.59.30.82, Fax : +33 (0) 383.41.30.79
Mél : `Olivier.Perrin@loria.fr` **Web** : `http://www.loria.fr/~operrin`

1.2 Statut actuel

▷ Maître de Conférences en Informatique à l'Université Nancy 2, IUT Nancy-Charlemagne, département Informatique nommé au 1er septembre 1996.

▷ Chercheur au LORIA dans le projet ECOO (Environnement et COOpération) sous la direction de Claude Godart, professeur à l'Université Henri Poincaré, Nancy I.

1.3 Emplois universitaires

▷ **Septembre 1999 – à présent**
Maître de Conférences à l'Université de Nancy 2. IUT de Nancy-Charlemagne, département Informatique.
Enseignements en Master SDR, DEA et DESS à l'Université Henry Poincaré, Nancy I.
Délégation à l'INRIA au sein du projet ECOO, dirigé par Claude Godart, de septembre 2002 à août 2004.

▷ **Septembre 1995 – Août 1999**
Maître de Conférences à l'Université Nancy 2. IUT de Nancy-Charlemagne, département Services et Réseaux de Communications (localisé à Verdun).

▷ **Septembre 1994 – Août 1995**
ATER à l'Université Henri Poincaré Nancy I (demi-poste), département Informatique. Enseignements complémentaires en Maîtrise Informatique et DEUG Sciences de la Matière.

▷ **Septembre 1991 – Août 1994**
Moniteur à l'Université Henri Poincaré Nancy I. Enseignement en Licence Informatique et en DEUG Nature et Vie.

1.4 Titres universitaires

Nov. 1994	Doctorat de l'Université de Nancy I. Sujet de thèse : *Un modèle d'intégration d'outils dans les environnements de développement de logiciels.* Directeurs de thèse : N. Boudjlida & J.C. Derniame Mention : Très Honorable.
Septembre 1990	DEA informatique à l'Université Henri Poincaré – Nancy I. Sujet de stage : *Un langage de programmation pour les structures d'accueil orientées objet*
Juin 1989	Maîtrise d'informatique à l'Université Henri Poincaré – Nancy I.
Juin 1988	Licence d'informatique à l'Université Henri Poincaré – Nancy I.
Juin 1987	DEUG Sciences de la Matière — filière mathématique obtenu à l'Université Henri Poincaré – Nancy I.

1.5 Activités de recherche

Depuis maintenant une dizaine d'années, il est devenu naturel pour les entreprises d'utiliser et de faire coexister des systèmes d'informations, des services ou des processus différents. Cela permet à ces entreprises de mener à bien des projets communs, ou de réaliser des coopérations afin de pouvoir continuer à progresser dans un marché de plus en plus compétitif. Dans ce contexte, l'*interopérabilité* est devenue de plus en plus importante afin de pouvoir satisfaire à la fois les besoins des clients et ceux des entreprises, tout en rentabilisant les investissements consentis dans des systèmes généralement assez coûteux.

L'interopérabilité représente la capacité qu'ont deux ou plusieurs composants (applications, sources de données, services mais aussi processus métier) de communiquer et de coopérer en dépit des différences (c'est ce que l'on nomme l'hétérogénéité) dues au langage d'implantation (Java, C++, ...), à l'environnement d'exécution (système d'exploitation, applicatif métier) ou au modèle d'abstraction choisi (niveau d'abstraction choisi pour représenter une information par exemple). Depuis longtemps, les problèmes liés à l'interopérabilité des systèmes sont nombreux [Weg96, Hei95, Man95]. Ces problèmes ont reçu une attention plus particulière ces vingt

dernières années au fur et à mesure que les entreprises passaient d'architectures centralisées à des architectures plus distribuées. Les trois propriétés principales de systèmes interopérables sont la prise en compte de la *distribution*, de *l'hétérogénéité*, et du *respect de l'autonomie* de chaque composant.

Au cours de ces 15 dernières années, je me suis respectivement intéressé à différents problèmes autour de l'interopérabilité, en étudiant respectivement les problèmes liés à *l'intégration de données* afin de prendre compte l'hétérogénéité au niveau données, les problèmes liés à la distribution et à l'autonomie dans le contexte des *processus interentreprises*, et enfin les problèmes liés la *fiabilité des compositions de services* dans les architectures orientées services. C'est ce fil conducteur – données, processus et services – que je vais présenter dans cette section, et qui m'amènera à discuter dans la section suivante des perspectives de mes travaux de recherche.

1.5.1 Intégration de données

Lors de ma thèse, j'ai commencé par travailler sur les aspects données. Ces aspects sont importants, et on constate qu'ils ressurgissent actuellement. Ceci n'est pas forcément étonnant puisque les données sont la clé de voûte de tout système informatique.

L'intégration de données a pour objectif de permettre un accès transparent à différentes sources de données hétérogènes. Elle donne l'illusion à l'utilisateur (ou à l'application) d'avoir accès à un système unique et homogène. Le problème auquel s'adresse l'intégration dans ce contexte est donc le suivant [1] : ≪fournir un *accès* (requêtes, éventuellement mises à jour) *uniforme* (les sources sont transparentes à l'utilisateur) à des *sources* (pas seulement des BD) *multiples* (même 2 est un problème) *autonomes* (sans affecter le comportement des sources) *hétérogènes* (différents modèles de données, schémas) *structurées* (ou au moins semi-structurées)≫.

Le problème auquel je me suis intéressé dans ma thèse [1] concernait l'intégration de données basée sur la conversion et la restructuration des données. Le contexte d'application de mes travaux concernait les Ateliers de Développement de Logiciel, dans lesquels le problème consistait à permettre à un outil d'accéder aux données d'un autre outil, indépendamment de leurs différences et des conflits (*structurels*, *sémantiques*, ou *intentionnels*). Pour cela, l'approche que nous avons proposée reposait à la fois sur l'intégration des données et sur leur transformation. Pour cela, nous avons défini un niveau intermédiaire qui autorisait chaque outil à déclarer sa propre structure de données (son modèle), et un ensemble de règles permettant de transformer un modèle dans un autre modèle à la fois au niveau de la structure et des données contenues dans ce modèle. Cette définition d'un adaptateur générique basé sur la description des sources de données représentait une avancée par rapport à la méthode habituelle qui était ad-hoc, et donc difficilement réutilisable. Notre proposition permettait d'avoir une architecture générique qui évitait d'avoir plusieurs adaptateurs, et qui réunissait le médiateur et l'adaptateur générique.

Notre proposition contenait 2 composants. Le premier était le modèle de représentation. Le deuxième était un ensemble de primitives permettant de passer d'un modèle à un autre.

Pour capturer les données provenant d'une application, on utilise un modèle de représentation structuré sous forme d'arbre étiqueté et ordonné. Un format de données était donc modélisé comme un arbre, un peu comme un document XML est actuellement modélisé par un XSD ou une DTD. Par rapport à XML, notre modèle était entièrement structuré, et non semi-structuré.

1. `http://www-poleia.lip6.fr/~doucet/`

Un des avantages du modèle est qu'il permet de passer d'une collection d'éléments à un ensemble de valeurs, et réciproquement, d'un ensemble de valeurs à une collection ordonnée d'éléments.

Le deuxième composant est un ensemble d'opérateurs permettant de manipuler un modèle afin d'obtenir un nouveau modèle. Cet ensemble d'opérateurs permet de définir une transformation pour passer d'un modèle de représentation r_1 à un modèle de représentation r_2. Une transformation est une composition de règles qui permettent d'agir à la fois sur la structure avec la possibilité de ré-ordonner ou de restructurer l'arbre de départ, ou d'agir sur le contenu des nœuds de manière à gérer l'hétérogénéité intentionnelle (domaines et contraintes d'intégrité).

Pour gérer ces deux composants, nous avons proposé un médiateur qui réalise l'intégration d'un modèle à partir d'un autre modèle, en respectant les étapes suivantes : (1) importation des deux modèles (source et cible) dans le médiateur, (2) comparaison des deux modèles et transformation du modèle source dans le modèle cible, et (3) exportation des données du modèle source vers des données compatibles avec le modèle cible.

1.5.2 Intégration par les processus

Une évolution naturelle m'a ensuite amené à m'intéresser aux problèmes liés aux processus métier dans le contexte de ce que l'on nomme les entreprises virtuelles.

Pour répondre aux problèmes liés au support des processus interentreprises créatifs, nous avons proposé l'approche PointSynchro [Bit03] qui est principalement basée sur deux composants. Le premier composant est un méta-modèle permettant de représenter les entités d'une entreprise virtuelle et les liens entre ces entités. Ce méta-modèle est important puisqu'il permet à chaque partenaire de pouvoir se positionner par rapport à une sémantique commune de l'entreprise virtuelle, de son organisation, de ses composants et de ses artefacts. Cette vision commune permet de diminuer les problèmes liés aux différences de représentation qui existent entre les différents partenaires. Le second composant est le *point de synchronisation* qui permet de gérer la coordination des processus partagés par les partenaires de l'entreprise virtuelle, et qui offre la gestion des partenaires, de leurs processus, et de leurs ressources.

L'opportunité qui m'a été offerte de m'intéresser aux processus m'a permis de tenir compte, en plus des données, des activités qui manipulent ces données. Dans le contexte des processus interentreprises, cette dualité entre activités et données est très importante, et amène à raisonner non seulement en termes d'abstraction au niveau des activités, mais également au niveau des données. Les formalismes utilisés pour la représentation des processus classiques sont alors insuffisants pour modéliser correctement ce nouveau types de processus. De même, les formalismes qui permettent de raisonner au niveau données sont insuffisants pour raisonner sur les activités. L'objectif des travaux que j'ai menés concernait donc la possibilité de trouver un équilibre entre la dimension activités et la dimension données. Le fait d'avoir eu comme support des exemples concrets de processus interentreprises a été une grande chance, car les problèmes sont plus faciles à cerner. En particulier, il est nécessaire d'avoir, comme dans le cas des données, une compréhension commune des artefacts manipulés (grâce à des méta-modèles, ou des ontologies). Les méta-modèles de PointSynchro répondent à ce besoin. De même, l'aspect coordination, l'intimité (*privacy*) et l'autonomie sont critiques pour les partenaires. Les services processus et le point de synchronisation en tant que composant sont une proposition pour prendre en compte ces aspects. La flexibilité, l'absence de modèle très structuré, et le sup-

port d'activités non nécessairement prévues sont des caractéristiques qui n'existent pas dans les processus classiques et qui doivent être proposées dans le cadre des processus interentreprises. Le cycle *Contrôler-Décider-Déployer* répond à ces besoins. Enfin, le modèle de contrat permet de vérifier que le processus (en prenant en compte à la fois les activités et les données) se déroulent conformément aux limites fixées par les différentes partenaires.

1.5.3 Intégration par les services

De part la technologie utilisée dans l'approche PointSynchro, j'ai été amené à étudier la technologie basée sur les services. L'approche service apparaît comme une proposition intéressante permettant de faciliter la prise en compte de certains problèmes liés à l'interopérabilité entre systèmes.

Le problème auquel je me suis particulièrement intéressé concerne principalement la composition et la coordination de services Web, avec comme objectif d'obtenir lors de l'exécution un résultat fiable. Je me suis donc intéressé aux aspects transactionnels, et également aux couches basses, c'est-à-dire tout ce qui concerne les requêtes, les réponses, et la coordination. L'idée est de conserver l'aspect autonome des services. De plus, on souhaite également rester dans un cadre *peer-to-peer*, c'est-à-dire que l'on ne souhaite pas avoir un orchestrateur qui prenne le contrôle des différents services (pas de centralisation de la coordination). Dans le même temps, puisque les normes actuelles sont assez pauvres en terme de sémantique, et en particulier de sémantique opérationnelle, nous avons rajouté aux définitions de services des propriétés transactionnelles, ce qui rajoute un peu d'information de manière à connaître un peu mieux un service, de ce qu'il offre, et de la garantie qu'il est susceptible de d'assurer. Nous avons travaillé avec Sami Bhiri sur un algorithme qui, étant données les propriétés transactionnelles de plusieurs services et étant donnée une composition, est capable de déduire les propriétés transactionnelles offertes par la composition. Nous avons également travaillé sur un autre algorithme qui cette fois part d'une composition pour laquelle on fixe une propriété transactionnelle à satisfaire, et à partir de celle-ci, choisir les services qui peuvent garantir cette propriété. Nous avons implanté un prototype qui réalise ces deux algorithmes, et gère également l'exécution de la composition obtenue. En terme de coordination, nous nous sommes inspirés pour commencer des modèles de coordination issus des workflows, et nous avons ensuite travailler sur des scénarios dont la coordination est plus évoluée. Nous avons ainsi obtenu ce que nous avons appelé des patrons transactionnels. Enfin, en ce qui concerne la coordination, nous avons travaillons sur une approche mixte, c'est-à-dire une coordination basée à la fois sur le contrôle (plutôt orientée workflow) et sur les données (cohérence transactionnelle).

Nous avons commencé à travailler sur un autre aspect important dans le contexte de la composition de services Web. Il s'agit de prendre en compte l'aspect temporel. Le problème est le suivant : étant donné un ensemble de services Web, un service abstrait (le service but) décrivant le résultat que l'on souhaite obtenir, comment composer cet ensemble de services de telle sorte que le service but soit satisfait ? Pour ce faire, on essaie de coordonner les services Web capables d'échanger des messages. Cependant, si la coordination ne satisfait pas le service but, nous essayons alors de générer un nouveau service, appelé *Médiateur*. Le rôle de ce dernier est d'essayer de générer les flux de messages manquants à la composition, i.e., lorsque un service indispensable à la composition attend un message pour s'exécuter et qu'il ne le reçoit pas, alors le médiateur va tenter de générer ce message. Ces travaux servent maintenant de base à

l'intégration de contraintes temporelles au niveau des services Web considérés. Le problème est le suivant : on suppose qu'un service peut associer à une des fonctionnalités qu'il est capable d'assurer à la fois une limite de validité exprimée comme un intervalle, mais également un temps d'exécution. De ce fait, certaines instances de la composition calculée dans le cadre générale ne sont plus réalisables. Par exemple, si un service bancaire délivre un numéro de carte bleue dont la période de validité est fixée à une heure, le service auquel vous donnez ce numéro devra initier la demande de virement dans cet intervalle de validité. Nous avons proposé de modéliser les compositions de services par des systèmes à transitions (*State Transition System*) STS. Nous nous sommes basés sur le modèle présenté dans [KPP06] pour étendre le formalisme WSTTS (*Web Services Timed Transition Systems*), et ce pour considérer d'autres formes de contraintes temporelles plus expressives. La validation de la composition se fait ensuite en analysant la compatibilité des traces pouvant être produites par la composition par rapport aux propriétés des services.

1.6 Projet de recherche

Mes perspectives de recherche concernent plusieurs aspects autour de la composition de services Web. Les services Web sémantiques peuvent être considérés comme une solution aux problèmes de l'intéropérabilité. En effet, sous cette appellation, on trouve à la fois l'aspect données et la représentation de ces données, les aspects fonctionnels et opérationnels, et les aspects non-fonctionnels comme le temps, la fiabilité, la qualité de service, la sécurité. Un de mes objectifs est donc de m'intéresser à la problématique de la composition de services, en considérant simultanément les aspects suivants : la représentation formelle d'une composition et la capacité à raisonner sur les propriétés attendues, la description sémantique des services et des données qu'ils manipulent, et la prise en compte d'aspects non fonctionnels ou non opérationnels. Plus particulièrement, nous envisageons trois approches possibles : les aspects temporels, la sécurité dans les compositions de services, et les données.

1.6.1 Les aspects temporels d'une composition

Un premier objectif a trait à l'intégration du temps. Actuellement, ce critère est peu pris en compte dans les propositions de modèles de composition, or il existe de nombreux exemples dans lesquels le temps intervient : un service est accessible pour une durée donnée (référence absolue), ce qui peut avoir un impact sur la composition. Un autre exemple concerne l'accessibilité d'un service pendant une durée donnée à partir de la fin d'un autre service (référence relative), ce qui là encore a un impact sur la composition. Introduire le temps dans la composition n'est pas une chose triviale, et pour cela, nous pensons que nous avons besoin de nous intéresser à une formalisation qui permettra de raisonner sur la composition de manière automatique. Nous avons déjà initié ce genre de travaux dans [13, 14, 18]. Dans [Tou05, PBCT07], on trouve des propositions pour la notion d'analyse de compatibilité et de remplaçabilité pour des protocoles avec des aspects temporels.

La composition de plusieurs services Web devient un protocole contraint par des propriétés temporelles. Dans nos précédents travaux, nous avons abordé les aspects transactionnels des services que nous souhaitions composer. Les résultats sont encourageants, mais nous nous sommes

posés la question de savoir si par exemple un service pouvait voir certaines de ses propriétés changer avec le temps. Par exemple, un service est rejouable pendant 3 jours, puis il devient pivot. On peut avoir ce genre de situation dans le cas d'achats sur Internet, avec un service qui est annulable pendant une période donnée (7 jours), puis qui devient non modifiable ensuite. Dans ce cas, on voit bien que le temps joue un rôle non négligeable sur la composition obtenue, et qu'il peut être nécessaire d'intégrer ce concept de temps dans la composition elle-même. De la même façon, un service peut éventuellement être remplacé par un autre service mais dans un intervalle de temps donné. Par exemple, on peut supposer qu'un service sera accessible dans une plage de temps définie, ce qui là encore a une implication sur la composition.

Les conséquences de ces contraintes liées au temps sont doubles. D'une part, il faut pouvoir être capable d'exprimer les contraintes temporelles des services que l'on souhaite composer, et il faut être également en mesure de raisonner sur la composition en termes de respect des contraintes temporelles. Pour cela, il faut pouvoir travailler à un niveau d'abstraction qui permette de raisonner sur la composition. Nous avons donc commencé des travaux qui permettent d'exprimer certaines contraintes sur les services et qui permettent de spécifier dans une logique temporelle ces contraintes. Le problème est alors de pouvoir se ramener à une logique du premier ordre afin de pouvoir raisonner. Nous avons ainsi travaillé sur des contraintes sur les services en termes de contractualisation. Nous avons ensuite transformer ces contraintes en engagements, puis nous avons exprimé ces engagements dans le calcul d'événements. Tout ce processus est actuellement manuel, et ce que nous souhaitons faire maintenant est de pouvoir raisonner de manière automatique. Pour cela, nous avons commencé à travailler avec le projet CASSIS afin de pouvoir coder une logique temporelle (calcul d'événement discret, calcul de situation,...) dans une logique du premier ordre et de vérifier les propriétés attendues au moyen d'un système de *Model Checking* tel que UPPAAL . Ce système est capable de gérer efficacement des systèmes modélisés comme des automates temporisés, avec en plus un support pour des types de données simples tels que les entiers ou les tableaux.

Si la partie raisonnement sur la composition *a priori* est primordiale, la partie surveillance et vérification *a posteriori* des compositions obtenues est également importante. Nous comptons donc travailler sur la surveillance de l'exécution du service précédemment composé. Cet aspect sera abordé grâce aux travaux menés précédemment dans le projet ECOO sur la découverte de procédés. L'idée est de pouvoir vérifier l'exécution réelle d'une composition à partir des traces d'exécution, en prenant en compte les aspects temporels pris en compte dans la première partie. L'exécution telle qu'elle s'est réellement produite sera ensuite comparée à celle qui était prévue dans la composition de départ. Des différences peuvent en effet survenir (service non disponible à un instant donné, ré-exécution multiple d'un service,...). Ces travaux devrait également permettre de savoir s'il n'y a pas eu de problèmes concernant la sécurité (par exemple DoS sur un service...).

1.6.2 La sécurité et les services Web

Un deuxième objectif concerne toujours la composition de services, mais cette fois en présence de contraintes supplémentaires qui ne sont pas des contraintes fonctionnelles, à savoir des contraintes concernant la sécurité. Nous avons commencé à travailler sur ces problèmes dans les projets COPS et COWS. L'idée est de s'intéresser à la synthèse de services Web en prenant en compte les aspects authentification, autorisation,... Il s'agit de contraintes non-fonctionnelles

qui ont un impact sur le résultat de la composition, et sur son exécution. Comme en ce qui concerne les aspects temporels, le protocole issu de la composition devra respecter toutes les contraintes fixées pour le service composé et toutes les contraintes des services composants. Une fois encore, le problème est de pouvoir trouver un modèle (un niveau d'abstraction) qui permette d'exprimer ces contraintes de manière précise, et qui permette ensuite de raisonner sur la composition obtenue, afin de calculer la validité de la composition par rapport aux contraintes initiales, et de vérifier à l'exécution que la composition ne viole pas ces contraintes.

Il existe actuellement encore peu de travaux en ce qui concerne la sécurité des interactions entre différents services lorsque l'on désire composer ces derniers. En particulier, les trois questions suivantes sont actuellement sans réponse :

- comment définir un modèle de contrôle d'accès qui soit compatible avec les besoins et le fonctionnement des architectures orientées service (*Service Oriented Architecture*),
- comment savoir si une politique de sécurité est valide et décidable étant donné un modèle de composition de services,
- comment vérifier que l'exécution d'une politique de sécurité est correcte.

De plus, si l'on se place dans le contexte des services Web, plusieurs modèles de sécurité différents (issus des systèmes que les services sont censés abstraire) coexistent, et une composition de services prend en compte ces différents modèles. Le problème posé est donc celui qui consiste à être sûr que malgré ces différences, la composition des services ne sera pas exposée à des problèmes liés à la sécurité.

Actuellement, ce n'est pas possible. D'une part, il existe beaucoup de standards dans la pile WS, et certains ont des intersections non vides, d'autres sont concurrents. D'autre part, il n'y pas de modèle formel sous-jacent permettant de calculer si les propositions d'un service client en terme de sécurité (au sens large) sont compatibles avec celles d'un service fournisseur. Une deuxième catégorie de problème concerne l'audit, c'est-à-dire la vérification a posteriori (cf. 1.6.1) que les obligations des uns et des autres se sont déroulées comme prévu. Cela doit permettre de pointer les endroits où il y a eu des problèmes. Pour tenter de répondre à ces deux problèmes, on envisage de définir un modèle qui permette d'unifier la notion de sécurité en termes d'authentification, d'autorisation, d'obligations, d'intégrité, d'intimité,... Si l'on arrive à définir un modèle (formel) qui prenne en compte tous ces aspects, on aura fait un grand pas, puisque l'on aura la possibilité de raisonner sur des concepts bien définis. Ensuite, il s'agirait de décliner chaque WS-spécification dans le modèle. Ensuite, on pourrait proposer des procédures permettant d'analyser et de gérer la sécurité en se servant de la proposition du point précédent. On pourrait ainsi calculer la compatibilité entre deux services d'un point de vue sécurité, calculer la remplaçabilité (similarité et différences entre deux services), et analyser la cohérence d'un ensemble de spécifications liées à la sécurité pour un service donné. Bien sur, on devra prendre en compte l'aspect multi-niveaux (intimité, contrôle d'accès, confiance,...). Les questions auxquelles il faudra répondre sont les suivantes :

- quel type de formalisme : assertions/contraintes, règles,... ? sur quels objets ? pour quel aspect de la sécurité ?
- comment coder les politiques de sécurité et le protocole (interactions multiples) ?
- comment surveiller ? cohérence avec le formalisme choisi précédemment ?
- correspondance entre une politique de sécurité et un ensemble de services ?

L'objectif des travaux que je compte mener est donc de proposer un modèle de composition qui permette d'obtenir un protocole permettant l'interaction entre les différents services appar-

tenant à la composition et paramétré par les politiques de sécurité de chaque service et par la politique de sécurité attendue.

1.6.3 L'aspect données

Jusqu'à présent, peu de travaux s'intéresse aux données lorsque l'on parle des services Web. Pourtant, ces dernières sont présentes en permanence, mais elles ne sont pas mises en avant. Avec nos travaux sur les aspects transactionnels des services, nous avons commencé à nous intéresser à cet aspect, mais il reste encore beaucoup de travail. Pourquoi doit-on s'intéresser aux données dans le contexte des architectures orientées services ? Les données sont un point crucial de ces architectures. En effet, lorsque l'on parle de services, on parle d'échanges de messages qui contiennent des données, c'est-à-dire des représentations des objets des systèmes dont les services exposent une partie des fonctionnalités. Cependant, jusqu'à présent, très peu de travaux se sont intéressés à la gestion de ces données. Pourtant, quelques problèmes existent.

Le premier problème concerne le fait que, très souvent, les données ont des dépendances les unes avec les autres. C'est un problème que le couplage faible et l'autonomie des services font oublier. Par exemple, si on crée au sein de la même organisation deux services CommandeClient et GestionClient, ces deux services ne sont pas censés avoir des échanges de messages, et pourtant, on peut imaginer qu'ils collectent souvent leurs données de la même base de données. De même, ces deux services peuvent avoir besoin de données gérées par l'autre service. Cela a un impact, car si on imagine la situation dans laquelle on souhaite récupérer toutes les commandes faites par un client (pour lui accorder une remise par exemple), comment doit-on procéder, comment est-on sûr que les différentes commandes ont été enregistrées avec une référence valide vers ce client pour le service GestionClient, comment gère-t-on les contrôles qui peuvent être demandés concernant la solvabilité d'un client, comment gère-t-on les accès concurrents aux mêmes données si un service échoue,... Ces problèmes sont encore plus difficiles si on suppose deux services qui ne sont pas issus de la même entreprise. On constate donc sur cet exemple pourtant simple qu'il faut s'intéresser aux données lorsque l'on souhaite comprendre les interactions entre plusieurs services.

Un problème corrélé au problème précédent concerne la sémantique des données manipulées par les services, c'est-à-dire non plus seulement l'information brute (son type), mais également la connaissance sur ces informations. Comme indiqué dans le chapitre sur l'intégration de données, l'interprétation d'une donnée ne peut se contenter d'une annotation dite sémantique ou d'une ontologie. La connaissance dépend également des relations de la donnée avec d'autres données, du contexte d'utilisation de la donnée en intégrant des propriétés temporelles par exemple, ou de la possibilité de pouvoir transformer une donnée [PLC+08, Wie92, MGB+08].

Un troisième problème concerne le fait que les données, une fois extraites du système (ERP, CRM, base de données, WfMS,...), peuvent éventuellement être invalidées, périmées,... Supposons à nouveau un service qui permette de passer une commande. On peut préparer son bon de commande, puis le fournir en attachement à l'appel SOAP du service. Le bon de commande est alors analysé par le système abstrait par le service. Le bon de commande est ensuite renvoyé au client afin que ce dernier l'accepte. Or, il se peut que entre l'envoi du bon de commande à confirmer par le client et la réponse de celui-ci, certaines données soient invalidées. Par exemple, l'article n'est plus disponible, son prix a été modifié, la date de livraison a été mise à jour, la remise potentielle n'est plus valide,... En fait, tout ce concerne la logique métier des données

manipulées par les services ne sont pas répercutées. Cet exemple montre le problème qui se pose une fois que les données extraites ne sont plus sous contrôle de l'application qui les manipule en temps normal. Cela pose donc le problème de pouvoir contractualiser les services et surtout les données qui sont associées à ces services. Ce problème peut être relié au problème cité en 1.6.1 sur l'inclusion du temps dans les compositions (et les instances des compositions – les protocoles) de services.

Enfin, une autre catégorie de problèmes concerne l'intimité des données (*privacy*). Là encore, une fois les données extraites de la base de données, du système ERP, ou du WfMS, puis utilisées en dehors des frontières de l'organisation qui les gère, il faut garantir qu'elles ne seront pas utilisées d'une manière non prévue, et que les règles de propriété sur lesquelles se sont entendus les services seront respectées. Ce point peut d'ailleurs être relié aux aspects sécurité du paragraphe 1.6.2.

On constate donc que les problèmes relatifs aux données dans les architectures orientées service sont difficiles. Pour les aborder, il faut dépasser le stade actuel de la recherche sur les services afin de réfléchir aux liens entre les objets métiers, les données, les systèmes gérant ces données et les services (et leurs interactions). Ainsi, les problèmes concernant l'identité des données, la fédération de données, la réplication, les transactions, la sémantique des données, l'intimité des données, la gestion de l'état des données dans le temps, les services gérant de grande masse de données, la sécurité des données,... sont autant de problèmes qui sont relativement peu abordés par les spécifications actuelles, et sont des problèmes qui me motivent énormément.

1.7 Projets de recherche contractuels

▷ **OTAN PCIS, 1999-2001**.
Ce projet était financé par le ministère de la défense pour la définition d'une plate-forme de développement logiciel basée sur une structure d'accueil à base d'objets. L'objectif était de favoriser l'interopérabilité entre les outils de développement.

▷ **AEE, 1999-2001**.
Le but du projet national AEE (Architecture Electronique Embarquée) était de concevoir et de valider un processus rapide et sûr pour la définition de l'architecture système et le développement des logiciels associés, embarqués à bord des moyens de transport, notamment de l'automobile. L'INRIA a participé, dans le cadre de ce projet, à la définition du langage AIL (*Architecture Implementation Language*), à la définition de méthodes et d'outils de validation et de sûreté de fonctionnement, à la définition de méthodes de placement (projection optimisée de l'architecture logicielle sur l'architecture matérielle) et à la définition de composants logiciels et matériels réutilisables, et à la définition d'un processus de développement multi-intervenant. Le projet ECOO, en tant qu'équipe INRIA, a participé dans à la dernière tâche, celle de la définition d'un processus de développement multi-intervenant.

▷ **ITEA EAST-EEA, 2001-2004**.
Le projet EAST-EEA (*Embedded Electronic Architecture for the European Automotive Industry*) avait pour objectif majeur de permettre une intégration poussée des différents composants électroniques d'une automobile en définissant une architecture ouverte, et en fournissant les

outils nécessaires à l'interopérabilité des différents composants. L'idée était de proposé un langage de haut niveau adapté aux architectures électroniques embarquées afin que les véhicules de demain supportent l'implantation de nouvelles fonctions électroniques, ou l'adaptation de caractéristiques existantes. Ma participation dans ce projet était principalement dévolue à la plate-forme support permettant aux constructeurs et aux équipementiers de travailler sur les architectures embarquées de manière coopérative. Dans le cadre de ce projet, le LORIA-INRIA Lorraine et l'équipe ECOO ont obtenu le prix ≪*Achievement Award for Outstanding contribution to the ITEA program*≫, le 7 octobre 2004 lors du workshop ITEA de Valencia en Espagne.

▷ **FP7 NoE INTEROP, 2004-2007**.
Le réseau d'excellence INTEROP est supporté par la Commission Européenne pour une période de trois ans. INTEROP a pour but de réunir et de créer une coopération entre différents laboratoires universitaires européens sur le thème de l'interopérabilité pour les applications d'entreprises (*Interoperability for Enterprise Applications and Software*). INTEROP s'intéresse à plusieurs dimensions : les architectures et les technologies associées, la modélisation d'entreprise, et le domaine des ontologies afin de savoir comment la sémantique permet une meilleure interopérabilité. Ma contribution dans le réseau se situait à deux niveaux. Premièrement, j'ai participé au groupe de travail sur la Knowledge Map qui permettait de préciser quels étaient les thèmes qui intéressaient les différents partenaires du réseau. J'ai également participé au groupe de travail sur les architectures orientées services comme support de l'intéropérabilité.

▷ **QSL COWS, 2006-2007**.
J'ai coordonné avec Laurent Vigneron (CASSIS) cette opération QSL commune à CASSIS et ECOO qui visait à étudier l'utilisation du raisonnement par contraintes pour la conception sûre de services Web. Les objectifs étaient multiples. Un premier objectif consistait à spécifier formellement la composition de services Web. Un deuxième concernait la coordination d'un ensemble de services, en garantissant que l'exécution est sûre, tout en étant conforme à ce qui est attendu par toutes les parties. Enfin, un troisième objectif concernait les mécanismes permettant de surveiller la manière dont une composition précédemment spécifiée s'exécute, et de savoir réagir lorsque survient une exception. Il est clair que les objectifs étaient étroitement liés, et que nous visions la définition d'un modèle permettant de composer, de coordonner, et de surveiller.

▷ **GDR I3 GT WS, 2006-2007**.
J'ai participé au Groupe de Travail sur les Services Web du GDR I3 sur la modélisation, la découverte et la composition de services. Les animateurs étaient Marie-Christine Fauvet (CLIPS, Grenoble), Bernd Amann (LIP6, Paris), Mohand-Said Hacid (LIRIS, Lyon), Farouk Toumani (LIMOS, Clermont-Ferrand), Alain Leger (France Télécom R&D), Claude Godart (LORIA, Nancy) et moi-même. Les thèmes de recherche concernaient la modélisation des services et leur lien avec le web sémantique (exploration des formalismes permettant une description adéquate des comportements des services et leurs possibilités), la découverte des services (comment localiser et comparer des services dans la perspective de sélectionner un ou plusieurs services entrant dans la composition d'un autre, à la fois lors de la conception et lors de l'exécution), et la composition de services (compréhension des propriétés fondamentales de la composition des services).

▷ **ARA COPS, 2006-2008**.
Le projet Cops était un projet de l'action de recherche amont « Sécurité, système embarqués et intelligence ambiante » financé par l'«Agence nationale de la recherche» (ANR). Il regroupait des chercheurs de l'IRIT de Toulouse, du LIF de Marseille, du LORIA de Nancy (ECOO et CASSIS) et du laboratoire d'informatique de l'université de Franche-Comté. Le thème était la composition des politiques et des services. Il y avait quatre parties principales dans ce projet. En premier lieu, il fallait définir la famille des propriétés de sécurité qui peuvent être exigées des protocoles utilisés par les services et étudier la faisabilité de la vérification automatique des propriétés de ces protocoles. En second lieu, il fallait formaliser le type des contraintes pouvant être imposées à l'accès aux ressources par les politiques. En troisième lieu, il était nécessaire d'examiner les problèmes soulevés par l'usage concomitants des protocoles et des politiques et en particulier les questions de décidabilité et de complexité des nombreux problèmes d'accessibilité que cet usage concomitant implique. Enfin, l'élaboration d'un modèle formel de la composition des services et des algorithmes capables de calculer, étant donnés des services composants et la spécification d'un objectif de service, un service composé.

▷ **INRIA-CONICYT CoreWeb, 2007**-.
Je suis impliqué dans le projet CoreWeb qui a pour cadre le raisonnement par contraintes pour la composition de services Web. Ses responsables sont respectivement Eric Monfroy (côté chilien) et Michael Rusinowitch (côté français). Les équipes du LORIA impliquées sont ECOO et CASSIS.

▷ **INRIA ADT Galaxy, 2008-2010**.
L'action de développement technologique Galaxy est un projet qui regroupe plusieurs EPI INRIA et dont l'objectif est de produire une plate-forme agile aux niveaux architecture, intergiciels et métiers. Nous intervenons au niveau métier en ce qui concerne la gestion des processus (BPM) et au niveau surveillance de la plate-forme lors de l'exécution des processus métier.

▷ **Équipe associée INRIA VanaWeb, 2008**-.
Une équipe associée INRIA permet depuis 2008 de consolider de façon significative notre collaboration avec l'UTFSM autour des contraintes sous stratégies et leurs applications aux problèmes de composition pour le Web. Les équipes du LORIA impliquées sont ECOO, CASSIS et PAREO.

1.8 Activités d'encadrement

Co-encadrement de la thèse de Ehtesham Zahoor 2008-2011 (50%)

Je co-encadre avec Claude Godart la thèse d'Ehtesham Zahoor. L'objectif de la thèse est de proposer une approche déclarative qui permette dans un cadre commun de proposer la composition de services, et leur surveillance. Dans ce cadre, nous souhaitons aborder non seulement les aspects gestion des opérations (le flux de contrôle) mais également la gestion des données. De même, nous souhaitons intégrer des aspects non-fonctionnels tels que le temps et la sécurité. L'idée consiste à proposer un cadre dans lequel on pourra exprimer grâce à un ensemble de règles les contraintes qui définissent la composition des services. Ces règles permettront de définir le flux de contrôle, le flux de données, et les aspects non-fonctionnels (en l'occurrence le temps et certaines propriétés de sécurité). La résolution des règles aboutira à une composition exécutable.

L'avantage d'utiliser des règles est qu'avec la même abstraction, on pourra surveiller l'exécution de la composition calculée, de manière à vérifier que celle-ci se déroule conformément à sa définition. Une première ébauche de ce travail sera présentée à ICWS 2009 [10] et à WISE 2009 [9].

Co-encadrement de la thèse de Nawal Guermouche 2007-2010 (50%)

Je co-encadre avec Claude Godart la thèse de Nawal Guermouche. L'objectif de la thèse est de proposer un modèle de composition qui permette de modéliser, d'analyser, et de simuler un protocole permettant l'interaction entre les plusieurs services Web. Dans sa thèse, Nawal s'intéresse à l'étude des propriétés temporelles dans le cadre de la composition de services Web. En effet, le comportement des services Web dans des applications réelles ne dépend pas seulement des messages, mais dépend aussi d'autres propriétés comme les propriétés temporelles. Ainsi, la construction d'une composition correcte nécessite d'examiner les séquences des messages augmentées de propriétés temporelles. Des travaux préliminaires d'analyse de compatibilité de services Web ont été proposés ces dernières années, mais ces travaux ne considèrent que des services Web synchrones. Cependant, la nature des systèmes distribués en général et en particulier les services Web est asynchrone. Par conséquent, le domaine d'application de ces approches reste très limité. En outre, les travaux qui considèrent les propriétés temporelles ne permettent pas la découverte de certains conflits temporels qui sont étudiés dans cette thèse. Les problèmes auxquels Nawal s'intéresse concernent donc la définition d'un modèle formel qui prend en considération les abstractions nécessaires telles que les propriétés temporelles, la proposition d'un cadre d'analyse de la compatibilité d'un ensemble de services Web temporisés synchrones et asynchrones, et la proposition d'un cadre de composition (orchestration) qui permette de satisfaire le besoin des clients et ce en considérant les propriétés temporelles [13, 14].

Co-encadrement de la thèse de Sami Bhiri 2002-2005 (50%)

J'ai co-encadré avec Claude Godart la thèse de Sami Bhiri. L'objectif de la thèse était d'assurer la cohérence transactionnelle d'une composition d'un ensemble de services Web. Les services Web émergent actuellement comme une technologie intéressante pour automatiser les interactions entre entreprises. Ces dernières peuvent ainsi externaliser une partie de leurs procédés métier et les présenter comme des services Web. Ensuite, elles peuvent combiner à la volée les services existants pour offrir de nouveaux services composés à forte valeur ajoutée. Cependant, un des problèmes non résolu à l'heure actuelle est d'assurer la fiabilité des exécutions de ces services composés conformément aux besoins des partenaires. Dans la thèse, Sami Bhiri a présenté une approche transactionnelle pour assurer la semi atomicité des services composés exigée par les partenaires. Il a introduit un modèle de composition de services Web permettant de combiner la flexibilité des workflows et la fiabilité des modèles transactionnels. Il a proposé également un ensemble de techniques pour assurer la fiabilité d'exécution des services composés conformément aux besoins des partenaires. La thèse a fait l'objet de publications dans deux journaux d'audience internationale (Journal of Data Management 2009 [4], Journal of Integrated Design & Process Science 2004 [7]), un journal d'audience nationale (Technique et Science Informatiques 2009 [8]), et dans des conférences internationales avec comité de lecture (ICWE 2006 [16], WWW 2005 [21], EEE 2005 [22], SCC'04 [23], IDPT'03 [26]).

Co-encadrement de la thèse de Julia Bitcheva 2000-2004 (50%)

J'ai co-encadré avec Claude Godart, professeur à l'Université Henri Poincaré, Nancy 1, la thèse de Julia Bitcheva (2000-2004). Le sujet de la thèse était « PointSynchro : un service pour la coordination de procédés interentreprises coopératifs ». L'objectif était de proposer une approche originale pour la modélisation de procédés d'entreprises, et de leur intégration au sein de procédés interentreprises dans la perspective de créer une entreprise virtuelle dont les activités sont orchestrées par le biais de services accessibles par Internet. Dans cette thèse, Julia Bitcheva a proposé un ensemble de méta-modèles fixant un cadre pour la définition d'une entreprise virtuelle, et elle a également défini le concept de point de synchronisation pour réaliser le contrat régissant les interactions entre les partenaires de l'entreprise virtuelle. La thèse donné lieu à une publication dans un journal d'audience internationale (Data and Knowledge Engineering 2004 [5]) et de nombreuses publications dans des conférences internationales avec comité de lecture (BPM'03 [24], ICWS'03 [25], SEA'02 [27], AI'02 [28]).

Encadrement du stage de Master de Aymen Baouab 2009 (100%)

J'encadre le stage de Master de Aymen qui consiste à étudier l'apport de méta-services de sécurité dans une architecture orientée services. Le stage s'est déroulé en association avec le Centre de Recherche Gabriel Lippman à Luxembourg.

Encadrement du stage de Master de Ehtesham Zahoor 2008 (100%)

J'ai encadré le stage de Master d'Ehtesham Zahoor qui consistait à étudier les services de mashup existants (Yahoo Pipes, API Google,...) et de voir comment les patrons de workflow développés précédemment dans le projet ECOO pouvaient être réutilisés pour créer de nouveaux mashups.

Co-encadrement du stage de Master de Mounir Tlili 2006 (50%)

J'ai co-encadré ce sujet avec Christophe Ringeissen du projet CASSIS ce sujet de DEA qui consistait à étudier les modèles de composition de services Web en prenant en compte les aspects temporels.

Co-encadrement du stage de DEA de Moshen Rouched 2005 (50%)

J'ai co-encadré avec Claude Godart le stage de DEA de Moshen Rouached. Le sujet du DEA consistait à étudier la gestion de contrats en se basant sur la gestion d'événements complexes.

Co-encadrement du stage de DEA de Sami Bhiri 2001 (50%)

J'ai co-encadré avec Claude Godart le stage de DEA de Sami Bhiri intitulé « Les services électroniques » et dont l'objectif était de comprendre les technologies actuelles autour des services Web (SOAP, WSDL, UDDI, ebXML, BizTalk...) et de montrer leurs limites dans le contexte de la composition de services.

Encadrement de la thèse CNAM de Franck Wynen 2003 (100%)

J'ai encadré la thèse CNAM de Franck Wynen. L'objectif était de proposer une plate-forme de coopération inter-organisationnelle. Les idées de la thèse sont issues des travaux de la thèse de Julia Bitcheva. Franck a réalisé une implantation du point de synchronisation. La thèse a fait l'objet d'une publication dans une conférence internationale (BPM'03), et Franck Wynen a obtenu pour ce travail la mention Très bien. Il est actuellement chef de projet chez Sagem à Paris.

Parallèlement à ces stages, j'ai également encadré Marcos Orfila, étudiant urugayen en stage Internship de 6 mois à l'INRIA Lorraine en 2004, Tomas Laurenzo, étudiant urugayen en stage Internship de 6 mois à l'INRIA Lorraine en 2006, deux stagiaires en troisième année d'ingénieur de l'ESSTIN en 2001, et plusieurs projets tutorés en deuxième année d'IUT et année de Licence Professionnelle à l'IUT Nancy-Charlemagne (1995-2009).

1.9 Responsabilités collectives

▷ Chargé de mission pour les admissions à l'IUT Nancy-Charlemagne pour les campagnes 2000-2001 et 2001-2002.

▷ Responsable des projets tutorés S3/S4 au département Informatique de l'IUT Nancy-Charlemagne de 2007 à 2009.

▷ Membre de la commission de spécialiste de l'Université de Nancy 2 en 2006.

▷ Responsable informatique au département SRC de l'IUT Nancy-Charlemagne localisé à Verdun de 1996 à 1999.

1.10 Animation scientifique

1.10.1 Organisation de conférence

▷ Organisateur de la troisième conférence internationale BPM 2005 à Nancy.

▷ Co-organisateur avec Farouk Toumani du workshop ISWS dans le cadre de la troisième conférence internationale Wetice 2007 à Paris.

▷ Co-organisateur avec Farouk Toumani et Marie-Christine Fauvet du workshop SISW dans le cadre de la conférence Inforsid 2006 à Hammamet (Tunisie).

▷ Membre du comité d'organisation de la huitième conférence internationale WISE 2007 à Nancy.

1.10.2 Membre de comité de programme

Je suis ou je fus membre des comités de programme suivants :

BPM, Business Process Management : 2009, 2008, 2007, 2006

WIDTS, IEEE Web-Based Information Technologies and Distributed Systems : 2009, 2008

I3E, IFIP Conference on e-Business, e-Services, and e-Society : 2009

WISE, Web Information Systems Engineering : 2007

TCoB (ICEIS), Technologies for Context-Aware BPM : 2010, 2009, 2008, 2007

GRCIS (CAiSE), Governance, Risk and Compliance in Information Systems : 2010, 2008

GRC (WISE), Governance, Risk and Compliance in Web Information Systems : 2007

ISWS (Wetice), Information Systems & Web Services : 2007

WSCO (BPM), Web Services Choreography and Orchestration : 2005

INTEROP (OTM), On the Move : 2005

ENEI (BPM), Enterprise and Networked Enterprises Interoperability : 2005

Inforsid : 2009

BDA, Bases de Données Avancées : 2007

WBPMO (Notere), Business Process Management for Outsourcing : 2007

SISW (Inforsid), Systèmes d'Information et Services Web : 2006

Majest'IC : 2005

1.10.3 Rapports de lecture

J'ai eu ou j'ai la responsabilité de rédiger des rapports de lecture pour les principales revues et conférences de mon domaine. Dans les listes qui suivent on précise pour chaque revue le nombre de rapports, et pour chaque conférence l'année ou les années concernées.

Revues

IEEE Transactions on Services Computing 2009

Journal of Data Management : 2008

Annals of telecommunications : 2008

Information Systems Frontiers : 2008

Software and Systems Modeling (SoSyM) : 2008

Information and Software Technology Journal : 2006

Conférences

ICWS : 2010

BPM, Business Process Management : 2009, 2008, 2007, 2006, 2005, 2004

CAiSE, Conference on Advanced Information Systems : 2006, 2004, 2003

ASIAN, Asian Computing Science Conference : 2006

ADVIS, Advances in Information Systems : 2006

CoopIS, Cooperative Information Systems : 2004, 2003

ENEI, Enterprise and Networked Enterprises Interoperability : 2006

IBERAMIA/SBIA, Ibero-american Conference on AI : 2006

Saint, Symposium on Applications and the Internet : 2004

WEC, IEEE International Workshop on Electronic Contracting : 2004

Inforsid : 2009, 2003

ICPADS, International Conference on Parallel and Distributed Systems : 2001

BDA, Bases de Données Avancées : 2007, 2003, 2001

1.10.4 Tutoriel

Collaborative Business Processes for Virtual Enterprises, tutoriel lors des journées Spiral, 7 juin 2005, Centre Henri Tudor, Luxembourg.

1.10.5 Jury de thèse

J'ai été membre du jury de la thèse de Fahima Cheikh en 2009 (Université Paul Sabatier – Toulouse III).

1.11 Activités pédagogiques

Depuis ma nomination en tant que Maître de conférences à l'IUT Nancy-Charlemagne, au département SRC délocalisé à Verdun d'abord, puis au département Informatique depuis septembre 1999, j'ai eu l'occasion de participer à de nombreux enseignements, et d'encadrer des projets tutorés et des stages en entreprise.

1.11.1 Département SRC

Au département SRC, j'ai participé avec Norbert Herstchuh à l'élaboration des différents plans de cours de première et deuxième années. En particulier, j'ai enseigné les matières suivantes : architecture des ordinateurs, système, fonctionnement d'Internet, algorithmique, programmation multimédia, langage Java, et concepts des SGBD. La plupart des modules étaient des modules de 30h, exceptés les modules liés à Internet, programmation multimédia et SGBD qui étaient des modules de 50h. L'objectif des cours était de donner aux étudiants les connaissances générales des concepts de l'informatique en présentant leur intégration dans les produits multimédia (sites Web, CD-ROM, bornes interactives), de leur apprendre une méthodologie de conception permettant d'intégrer les différentes facettes de l'univers multimédia (graphisme, son, communication, interaction), de leur permettre de maîtriser la majeure partie des étapes de développement d'un produit multimédia, et enfin d'être capable de mener à bien la conception d'un projet réel. Dans un département SRC, l'informatique est un outil. Il a donc fallu non seulement présenter la matière de manière différente, et également adapter le contenu des cours à un public dont les connaissances étaient assez hétérogènes. En plus des cours, j'ai assuré l'encadrement de projets tutorés et de stages en entreprise.

Au département SRC, j'ai été responsable du parc informatique, ce qui impliquait non seulement le parc machines, mais aussi un contact étroit avec le CIRIL pour toute la partie réseau.

1.11.2 Département Informatique

À mon arrivée au département Informatique de l'IUT, j'ai eu la responsabilité du cours de bases de données en deuxième année. J'ai fait la majorité de mon service dans cette matière, et le reste en enseignant l'ACSI (Analyse et Conception des Systèmes Informatique).

Avant d'obtenir ma délégation en septembre 2002, j'ai assuré pendant deux années la responsabilité de chargé de mission des admissions. Lors de cette mission, j'ai supprimé le dossier papier qui nécessitait beaucoup de saisies manuelles pour le remplacer par une candidature Minitel puis Internet. J'ai pour cela travaillé en relation avec le service Scolarité et avec les 5 départements de l'IUT. Cela a permis à l'IUT d'être plus réactif au niveau des inscriptions et des appels sur liste complémentaire.

Je suis depuis 2007 responsable de l'organisation des projets tuteurés en S3/S4 et S3bis/S4bis.

Bases de données (50h, DUT S3)

Je suis responsable du module SGBD en deuxième année. Dans ce module (50 heures), j'aborde la conception de bases de données, l'utilisation d'un langage L4G, ainsi que l'étude des mécanismes internes aux SGBD actuels (les transactions, l'optimisation de requêtes, les triggers, la confidentialité et les contraintes d'intégrité). La mise en pratique se fait sur le SGBD Oracle. J'aborde également les moyens qui existent à l'heure actuelle pour accéder aux SGBD les plus classiques (Oracle, Sybase, PostgreSQL). En particulier, on étudie l'API JDBC proposée avec le JDK Java, et PearDB qui est une librairie PHP permettant l'abstraction pour l'accès aux SGBD. L'idée du module SGBD en deuxième année est de sensibiliser les étudiants à la conception de bases de données, et également de leur montrer comment fonctionnent les architectures 3 tiers, en se concentrant sur le tiers métier (la logique applicative) et le tiers stockage et accès aux données.

Autour des bases de données (20h, DUT S4)

Je suis responsable du module ≪Autour des bases de données≫ en deuxième année. Dans ce module de 20h, j'aborde trois problématiques autour des bases de données, à savoir comment échanger des données extraites à partir d'un système de gestion de base de données, quels sont les principes qui permettent s'assurer la sécurité des données stockées dans une base de données, et quelles sont les bases de données disponibles dans le monde du logiciel libre. Dans la première partie, je présente ainsi les technologies liées à XML, ainsi que le langage XSLT. Dans la deuxième partie, j'essaie de sensibiliser les étudiants aux aspects liés à la sécurité des données. Enfin, dans la troisième partie, nous essayons d'analyser les points forts et les points faibles des solutions actuelles dans le monde du logiciel libre. Les étudiants sont souvent très intéressés par ce cours.

Java Avancé (45h, LPro Cisii)

Je suis responsable du module Java Avancé en Licence Pro Cisii. Ce module a pour objectif de montrer aux étudiants les concepts et deux technologies permettant de faire de la distribution. J'ai donc abordé dans ce module de 45 heures le fonctionnement des architectures distribuées, ainsi que les besoin particuliers associés à la distribution (données, exécution, utilisateurs). Je

détaille les problèmes tels que l'hétérogénéité, l'ouverture, la sécurité, le passage à l'échelle, la tolérance aux pannes, la concurrence, la transparence, et les problèmes de protocoles et de nommage. En ce qui concerne la conception, j'aborde également la notion de patrons de conception (*Design Patterns*) puisque beaucoup d'architectures distribuées font appel à ces modèles. La mise en œuvre se fait grâce à deux technologies : la première est *Java Enterprise Edition* (JEE), la seconde est celle des services Web (SOAP, WSDL, REST). Enfin, je présente quelques éléments de conception, en présentant et en détaillant les patrons de conception *Proxy*, *Factory*, *Front Controler*, *Wrapper*, et *Interceptor*.

XML/XSLT (30h, LPro Cisii)

En Licence Pro Cisii, j'ai monté en 2001 le module de Publication en relation avec les technologies permettant la publication d'informations sur Internet, à savoir, les CSS (*Cascading Stylesheets*), XML (*eXtensible Markup Language*) et XSLT (*eXtensible Stylesheet Language Transformations*). Il s'agit d'un module de 30 heures qui permet de montrer aux étudiants les nouvelles technologies pour la publication d'informations sur le Web. Depuis 2005, j'assure à nouveau ce cours dont le responsable est désormais Yann Boniface.

Administration Bases de données (16h, LPro Asrall)

Je suis responsable du module Administration de bases de données en Licence Pro Asrall. Ce module consiste à aborder les aspects liés à l'administration des bases de données pour assurer que les utilisateurs puisse bénéficier d'une garantie maximale de service. J'aborde en particulier les thèmes autour de la sécurité, des mécanismes de sauvegarde/restauration, de la gestion des performances, et de la gestion du changement. La mise en œuvre est ensuite abordée en utilisant le SGBD PosgreSQL (installation, configuration, optimisation).

Interface (30h, LPro Cisii)

Sous la responsabilité de Slim Ouni, je participe au module Interfaces utilisateur pour le Web. Ce module de 30h permet de réfléchir aux aspects non plus techniques du Web, mais aux interactions avec les utilisateurs en prenant compte l'utilisabilité, l'expérience de l'utilisateur sur le Web, les problèmes classiques. La deuxième partie du module concerne la mise en œuvre d'interface en utilisant les technologies autour d'AJAX.

1.11.3 Master Systèmes Distribués et Réseaux – SDR

J'assure depuis 2006 10h de cours dans le Master SDR sur la partie Services Web. J'aborde à la fois les standards autour des services Web, mais également les problèmes de recherche liés à la composition de services.

Bibliographie

[1] O. Perrin. Un modèle d'intégration d'outils dans les environnements de développement de logiciels. Thèse de doctorat de l'Université de Nancy I, Novembre 1994.

Chapitres de livre

[2] C. Godart, O. Perrin Les processus métiers : concepts, modèles et systèmes. *Éditeur*. Lavoisier, Hermès, 2009.

[3] B. Benatallah, O. Perrin, F. Rabhi, C. Godart. Web Service Computing : Overview and Directions. *Book chapter*. In Handbook of Innovative Computing. Editor : Albert Y. Zomaya. Springer, 2005.

Journaux internationaux

[4] S. Bhiri, W. Gaaloul, C. Godart, O. Perrin, M. Zaremba. Ensuring Customised Reliability of Composite Web Services. *Journal of Data Management*, À paraître, 2009.

[5] O. Perrin, C. Godart. A model to support collaborative work in virtual enterprises. *Data & Knowledge Engineering*, 50(1) : 63-86, July 2004.

[6] C. Godart, P. Molli, G. Oster, O. Perrin, H. Skaf-Molli, P. Ray, F. Rabhi. The ToxicFarm Integrated Cooperation Framework for Virtual Teams. *Distributed and Parallel Databases*, 15(1) : 67-88, 2004.

[7] S. Bhiri, O. Perrin, W. Gaaloul, C. Godart. An Object-oriented metamodel for inter-entreprises cooperative processes based on Web services. *Journal of Integrated Design and Process Science*, 8(2) : 37-55, June 2004.

Journaux nationaux

[8] S. Bhiri, C. Godart, O. Perrin. Patrons Transactionnels pour Assurer des Compositions Fiables de Services Web. *Technique et Science Informatiques*, 28(3) : 301-330, 2009.

Conférences internationales avec comités de lecture

[9] E. Zahoor, O. Perrin, C. Godart. An Integrated Declarative Approach to Web Services Composition and Monitoring , in *Proceedings of WISE 2009*, Poznan, 2009.

[10] E. Zahoor, O. Perrin, C. Godart. Rule-based semi automatic Web services composition, in *Proceedings of ICWS 2009*, Los Angeles, 2009.

[11] E. Zahoor, O. Perrin, C. Godart. Mashup Model and Verification using Mashup Processing Network, in *Proceedings of CollaborateCom 2008*, New York, 2008.

[12] E. Monfroy, O. Perrin, C. Ringeissen. Dynamic Web Services Provisioning with Constraints, in *Proceedings of CoopIS 2008*, Mexico, 2008.

[13] N. Guermouche, O. Perrin, C. Ringeissen. A Mediator Based Approach For Services Composition, in *Proceedings of the 6th Software Engineering Research, Management and Applications Conference, SERA 2008*, Prague, 2008.

[14] N. Guermouche, O. Perrin, C. Ringeissen. Timed Specification For Web Services Compatibility Analysis, in *Electr. Notes Theor. Comput. Sci. 200(3) : 155-170*, 2008.

[15] S. Bhiri, O. Perrin, C. Godart. Extending workflow patterns with transactional dependencies to define reliable composite Web services, in *Proceedings of Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)*, February 2006.

[16] S. Bhiri, O. Perrin, C. Godart. Transactional patterns for reliable web services compositions, in *Proceedings of the 6th International Conference on Web Engineering, ICWE 2006*, Palo Alto, 2006.

[17] U. Yildiz, O. Perrin, and C. Godart. On automating networked enterprise management, in *Proceedings of the 3th International Conference on Business Process Management, BPM 2005*, Nancy, 2005.

[18] M. Rouached, O. Perrin, C. Godart. A Contract-based Approach for Monitoring Collaborative Web Services using Commitments in the Event Calculus, in *Proceedings of WISE 2005*, New-York, November 2005.

[19] S. Bhiri, K. Gaaloul, C. Godart, O. Perrin. Transactional Patterns : Combining Workflow Flexibility and Transactional Reliability, in *Proceedings of BPM 2005*, Nancy, September 2005.

[20] M. Rouached, O. Perrin, C. Godart. A Contract Layered Architecture for regulating cross-organisational Business Processes, in *Proceedings of BPM 2005*, Nancy, September 2005.

[21] S. Bhiri, C. Godart, O. Perrin. Ensuring Required Failure Atomicity of Composite Web services, in *Proceedings of 14th International Conference in WWW 2005*, Japan, May 2005.

[22] S. Bhiri, C. Godart, O. Perrin. Reliable Web services composition using a transactional approach. *IEEE International Conference on e-Technology, e-Commerce and e-Service - EEE 05*, Hong Kong, March 2005.

[23] S. Bhiri, C. Godart, O. Perrin. A Transaction-oriented Framework for Composing Transactional Web Services. *IEEE International Conference on Services Computing - SCC 2004*, Shanghai, China, IEEE digital library, 654-663, Sep 2004.

[24] O. Perrin, F. Wynen, J. Bitcheva, C. Godart. A Model to Support Collaborative Work in Virtual Enterprises. *International Conference on Business Process Management, BPM'03*, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings. Lecture Notes in Computer Science 2678, Springer.

[25] J. Bitcheva, O. Perrin, C. Godart. Coordination of Cooperative Processes with the Synchronization Point Concept. *Proceedings of the International Conference on Web Services, ICWS '03*, Las Vegas, Nevada, USA. CSREA Press, June 23 - 26, 2003.

[26] S. Bhiri, O. Perrin, W. Gaaloul, C. Godart. An Object Oriented Metamodel for Inter-entreprises Cooperative Processes based on Web Services. *7th World Conference on Integrated Design and Process Technology 2003 - IDPT 2003*, Austin, Texas, Etats Unis, Dec 2003.

[27] O. Perrin, J. Bitcheva, C. Godart. Cooperative Process Coordination. *SEA 2002, 6th IASTED International Conference Software Engineering and Applications*, Cambridge, USA. November 4-6, 2002.

[28] O. Perrin, C. Godart. A Mail Based and XML based Protocol To Support Workflow Interoperability. *20th IASTED International Multi-Conference Applied Informatics (AI 2002)*, February 18-21, 2002, Innsbruck, Austria.

[29] C. Godart, O. Perrin, H. Skaf-Molli. Cooperative Workflows to Coordinate Cooperative Asynchronous Applications in a simple Way. *International Conference on Parallel and Distributed Systems (ICPADS '2000)*. Iwate Prefectural University, Iwate, Japan. July 4 (Tue.) – 7 (Fri.), 2000.

[30] C. Godart, O. Perrin. The *coo* operator to support and improve the flexibility of adaptive workflows. *Proceedings of IEEE Globecom'99 (Global telecommunications conference)*, Rio de Janeiro, Brésil, 5-9 Décembre 1999.

[31] C. Godart, O. Perrin, H. Skaf. *coo* : a workflow operator to improve cooperation modeling in virtual processes. *Proceedings of the Ninth IEEE International Workshop on Research Issues on Data Engineering RIDE-VE '99*, March 22-25, Sydney, 1999.

[32] N. Boudjlida, O. Perrin. Database Versioning and Restructuring. *XII Brazilian Symposium on Database Systems - SBBD '97*,1997, October 13-15, Fortaleza, CE - Brazil.

[33] N. Boudjlida, M. Bouneffa, O. Perrin. Object and Schema Versioning and Restructuring in Databases. *Proceedings of 16th DATASEM*, 1996, October, pages 159–169, Brno (CZ).

[34] J.C. Derniame, O. Perrin, P.F. Tiako. Control Integration like Verification Dimension in Software Engineering Environment. *Proceedings of 3rd African Symposium on Research in Computer Science*, CARI'96, 1996, October, pages 195–204, Libreville.

[35] O. Perrin, N. Boudjlida. An Open Approach for Data Integration. *Proceedings of the 2nd International Conference on Object-Oriented Information Systems*, OOIS '95, 1995, December, pages 94–98, Dublin.

[36] O. Perrin, N. Boudjlida. A Formal Framework and a Procedural Approach for Data Interchange. *Proceedings of the 3rd International Conference on System Integration*, ICSI'94 (IEEE), 1994, August, pages 476–485, São Paulo, Brazil.

[37] O. Perrin, N. Boudjlida. Towards a Model for Persistent Data Integration. *CAiSE'93 – The Fifth Conference on Advanced Information Systems Engineering – LNCS 685*, 1993, June, pages 93–117, Paris.

[38] O. Perrin, N. Boudjlida. Tool Integration in Integrated Software-Engineering Environments : data dimension. *Fifth International Conference on Software Engineering and its applications*, 1992, December, pages 95–105, Toulouse.

Workshops internationaux avec comité de lecture et publication des actes

[39] O. Perrin, F. Toumani. Final report, in *Information Systems and Web Services Workshop*, 2007.

[40] O. Perrin, C. Godart. An approach to implement contracts as trusted intermediaries. *IEEE Conference on E-Commerce Technology, International Workshop on Electronic Contracting (WEC)*. July 6-9, 2004, San Diego, California, USA.

[41] O. Perrin, C. Godart. Contract Model to Deploy and Control Cooperative Processes. *Technologies for E-Services, 4th International Workshop, TES 2003*, Berlin, Germany, September 8, 2003, Proceedings. Lecture Notes in Computer Science 2819, Springer.

[42] C. Godart, J.C. Bignon, C. Bouthier, P. Canalda, F. Charoy, G. Halin, O. Malcurat, P. Molli, O. Perrin, H. Saliou. Asynchronous Coordination of Virtual Teams in Creative Applications (co-design or co-engineering). *Information Technology for Virtual Enterprises (ITVE 2001 workshop)*, Gold Coast, Australia. January, 29 - February 1, 2001.

[43] C. Godart, P. Molli, O. Perrin. Modelling and enacting Processes. Some difficulties. *International Process Technology Workshop (IPTW)*. In cooperation with SIGSOFT. Grenoble. France. September 1-3, 1999.

Conférence nationales avec comité de lecture et publications des actes

[44] N. Boudjlida, O. Perrin. Un cadre formel et une approche procédurale pour l'échange de données. *Actes des journées "Ingénierie des bases de données : migration, intégration, évolution" (AFCET, Comité Technique Bases de Données)*, 1994, Septembre, pages 95–105, Paris.

Rapports internes

[45] P. Castelpietra, F. Simonot-Lion, O. Perrin. Objects - Models - Meta models in EAST Project, Rapport interne LORIA A02-R-414, Décembre 2002.

[46] O. Perrin, C. Godart. AEE process : inter-organizational process model v. 1.5 and AEE inter-organizational communication model v. 1.1. AEE project, December 2001.

[47] O. Perrin, J. Bitcheva. La gestion de procédés coopératifs inter-entreprises. Rapport de recherche, LORIA-France Telecom Recherche, Mars 2001.

[48] O. Perrin, F. Charoy, D. Grigori. The plan in a cooperative process : a pragmatic point of view. Research Report, LORIA, 2000.

[49] O. Perrin. PCIS2 Architecture Specification, Version 1.0, Framework Administration chapter, pages 87-106. January 31, 1998.

[50] O. Perrin, M. Kassab. Trading and Type management in SEE. Research Report, LORIA, 1998.

[51] O. Perrin, W. O'Mullane. Experiences with CORBA and Interoperability. Research Report 94-R-021, CRIN, 1994, April.

Chapitre 2

Synthèse des travaux

2.1 Introduction

Depuis maintenant une dizaine d'années, il est devenu naturel pour les entreprises d'utiliser et de faire coexister des systèmes d'informations, des processus différents ou des services. Cela permet à ces entreprises de mener à bien des projets communs, ou de réaliser des coopérations afin de pouvoir continuer à progresser dans un marché de plus en plus compétitif. Dans ce contexte, l'*interopérabilité* est devenue de plus en plus importante afin de pouvoir satisfaire à la fois les besoins des clients et ceux des entreprises, tout en rentabilisant les investissements consentis dans des systèmes généralement assez coûteux.

Qu'est ce que l'interopérabilité ? L'interopérabilité représente la capacité qu'ont deux ou plusieurs composants (applications, sources de données, services mais aussi processus métier) de communiquer et de coopérer en dépit des différences dues au langage d'implantation (Java, C++, ...), à l'environnement d'exécution (système d'exploitation, applicatif métier) ou au modèle d'abstraction choisi (niveau d'abstraction choisi pour représenter une information par exemple). Il s'agit donc d'un objectif à atteindre et d'une qualité que les utilisateurs réclament pour un environnement afin de bénéficier d'un ensemble d'applications interopérables, de bases de données interopérables ou de services interopérables. Depuis longtemps, les problèmes liés à l'interopérabilité des sytèmes sont nombreux [Weg96, Hei95, Man95]. Ces problèmes ont reçu une attention plus particulière ces vingt dernières années au fur et à mesure que les entreprises passaient d'architectures centralisées à des architectures plus distribuées. Les trois propriétés principales de systèmes interopérables sont la prise en compte de la *distribution*, de *l'hétérogénéité*, et du *respect de l'autonomie* de chaque composant.

En ce qui concerne la *distribution*, on s'intéresse à la fois aux données, aux procesus et aux services. Ceux-ci peuvent en effet exister sur des supports répartis géographiquement. Il se pose alors le problème de la localisation, de la disponibilité, et de l'accès à ces sources de données, à ces processus ou à ces services. La deuxième propriété prend en compte *l'hétérogénéité* : données, modèles, langages, API,... Lorsque l'on parle d'hétérogénéité, deux problèmes se posent rapidement : le problème structurel et le problème sémantique. Nous reviendrons sur ces problèmes un peu plus loin. Enfin, la dernière propriété a trait à *l'autonomie*. L'idée est que l'interopérabilité ne doit pas remettre en cause l'autonomie que possède chaque entreprise sur la gestion de ses sources de données, de ses processus ou de ses services. On privilégie ainsi un couplage faible

entre les différents systèmes et/ou applications, par opposition à un couplage fort.

À l'origine, on a d'abord vu apparaître de nombreux travaux autour de l'intégration de données, avec en particulier tous les travaux sur l'intégration de données dans les SGBD. Sont venus ensuite les travaux autour de l'interopérablité autour du contrôle. En effet, les activités d'un système sont basées sur les règles explicites de fonctionnement du système – c'est ce que l'on appelle les processus. Enfin, le découpage des activités en unités plus petites, et censées être plus modulables a permis de parler de services (ou de composants suivant le contexte d'utilisation à l'intérieur d'une organisation ou au-delà d'une organisation). Permettre à plusieurs services de fonctionner de concert est devenu un problème important. La terminologie employée – « de concert » – n'est pas innocente puisque l'on parle assez justement d'orchestration de services.

En règle générale, l'intégration de plusieurs sources d'information ou de plusieurs services mène à combiner ces différentes sources ou ces différents services de manière à ce qu'ils forment une vue uniforme pour les utilisateurs, leur donnant l'illusion de n'interagir qu'avec un seul système. C'est le cas dans plusieurs exemples de notre vie courante. Ainsi, lorsque l'on passe une commande sur un site Web, nous ne percevons pas necéssairement les différentes applications qui sont utilisées pour satisfaire notre requête : système de gestion de la commande, système de gestion du stock, portail Internet, système de transfert bancaire, système de gestion de la livraison, etc. Werner Vogels, CTO de Amazon, précisait ainsi dans une communication sur ACMQueue que chaque accès sur le site Amazon.com mettait en jeu une centaine de services pour collecter les données et afficher la page d'accueil [1].

La raison d'être de l'intégration est donc double : d'une part faciliter l'accès à l'information quelque soit le nombre de systèmes/applications grâce à la création d'une vue unique, et étant donnée une requête ou un appel de fonction, faire en sorte que la combinaison de plusieurs sources de données ou systèmes permette de répondre à cette requête ou à cet appel d'autre part.

À travers l'exemple de la commande via un site Web, on voit qu'il existe de nombreuses retombées liées à l'intégration de plusieurs systèmes d'informations, et on recense un grand nombre d'applications dans de nombreux domaines de notre vie de tous lesjours. Un premier domaine concerne par exemple ce que l'on appelle désormais l'intelligence métier (*Business Intelligence* – BI). Les informations issues de l'intégration concernent les activités métiers, et peuvent être utilisées à des fins statistiques (*Online Analytical Processing* – OLAP) ou à des fins de fouille de données (*data mining*) de façon à donner aux personnes compétentes les moyens de mieux décider ou prévoir une stratégie par exemple. Dans le domaine de la relation client (*Customer Relationship Management* – CRM), on utilise également l'intégration de données afin d'améliorer les services offerts aux clients. Enfin, dans tout ce que l'on appelle le commerce électronique (*e-commerce*), l'intégration de l'information permet et facilite les transactions entre clients et entreprises (*B2C*) ou entre plusieurs entreprises (*B2B*).

De la même façon que l'on discute de l'intégration d'information, les applications et les services peuvent aussi être intégrés, de façon à fournir un seul point d'accès (un service est composé de plusieurs services), ou de façon à adapter au mieux les services aux besoins. C'est par exemple le cas pour tout ce qui relève du travail collaboratif interorganisationnel ou de la réingénérie de processus (*Business Process Reengineering* - BPR) pour lesquels l'intégration des services et applications qui supportent les processus permet de réduire les délais de mise sur

1. `http://queue.acm.org/detail.cfm?id=1142065`. URL vérifié le 24 juin 2009

le marché (*time-to-market* – T2M) ou de fournir des services à plus forte valeur ajoutée. De même, intégrer des services provenant de différentes organisations permet de mettre en place des collaborations qui cassent les frontières traditionnelles des entreprises. C'est ce que l'on a appellé les *entreprises virtuelles*, virtuelles dans le sens où elles sont distribuées à travers le temps, l'espace et que les processus peuvent être distribués dans plusieurs organisations. Cela signifie par exemple qu'un processus peut être scindé en plusieurs parties (un processus coopératif dépasse les frontières d'une entreprise) et que chaque organisation ou chaque entreprise participante implante une partie seulement du processus (sous-traitance). On peut également avoir un scénario dans lequel la même activité peut être implantée par plusieurs partenaires, et il faut alors gérer la synchronisation des données produites (co-développement). On distingue alors une entreprise virtuelle par le fait qu'elle est issue d'une alliance temporaire et qu'elle est ainsi amenée à composer certains services, à partager certaines ressources et connaissances provenant d'organisations diverses sans que ces dernières perdent leur autonomie. Une telle entreprise nécessite donc des mécanismes d'intégration (dans le sens composition) relativement flexibles (modification de sous-traitant, arrivée de nouveaux partenaires, modification des participants à une activité donnée, capitalisation de projets précédents avec certaines modifications mineures ou majeures, etc.) et il est maintenant établi que les architectures centralisées issues du monde des systèmes d'information ne sont pas adaptées au support de ce modèle d'entreprise.

Enfin, dans le contexte propre de l'entreprise, le problème de l'intégration est défini comme la capacité à intégrer des informations et des fonctionnalités provenant des différents systèmes de l'entreprise, et se présente sous deux formes, l'EII (*Enterprise Information Integration*) et l'EAI (*Enterprise Application Integration*). L'EII concerne l'intégration des données et des informations. L'EAI considère l'intégration au niveau de la logique des applications, avec un contrôle centralisé, d'un nombre relativement réduit de processus et de participants.

Au cours de ces 15 dernières années, je me suis respectivement intéressé à différents problèmes autour de l'interopérabilité, en étudiant respectivement les problèmes liés à *l'intégration de* données afin de prendre en compte l'hétérogénéité au niveau données, les problèmes liés à la distribution et à l'autonomie dans le contexte des *processus interentreprises*, et enfin les problèmes liés à la *fiabilité des compositions de services* dans les architectures orientées services. Lors de ma thèse, j'ai commencé par travailler sur les aspects données. Ces aspects sont importants, et on constate qu'ils ressurgissent actuellement. Ceci n'est pas forcément étonnant puisque les données sont la clé de voute de tout système informatique. Une évolution naturelle m'a ensuite amené à m'intéresser aux problèmes liés aux processus métiers dans un contexte d'entreprise virtuelle. Puis j'ai étudié la dimension service qui apparait comme une proposition permettant de faciliter la prise en compte de certaines problèmes liés à l'interopérabilité entre systèmes. C'est ce fil conducteur – données, processus et services – que je vais présenter dans ce document, et qui m'amènera à discuter des perspectives de mes travaux puisqu'un bon nombre de problèmes sont toujours d'actualité, en dépit d'avancées importantes sur les aspects essentiellement architecturaux.

2.2 Le problème de l'interopérabilité

L'interopérabilité entre plusieurs systèmes d'information, plusieurs applications, plusieurs processus, plusieurs services constitue le cœur de nos travaux. Cela consiste généralement à combiner (composer) ces systèmes, applications, processus, services de façon à former un nouveau

système, une nouvelle application, un nouveau processus ou un nouveau service qui vont donner l'illusion aux utilisateurs qu'ils ont à faire à une seule et même entité. Dans le contexte des systèmes d'informations ou des bases de données, cela signifie que l'utilisateur manipulera les données grâce à une vue logique homogène, même si derrière, les données sont physiquement distribuées et qu'elles proviennent de sources de données hétérogènes. Cette hétérogénéité peut provenir de deux causes principales : une *structure* différente, ou une *sémantique* différente. Pour masquer cette hétérogénéité, toutes les données doivent être représentées en utilisant le même niveau d'abstraction, à savoir un modèle de données global, et une sémantique unifiée. Cela nécessite alors de détecter et de résoudre à la fois les conflits structurels et les conflits sémantiques. Ces problèmes de différences structurelles et de différences sémantiques sont les mêmes lorsque l'on parle d'applications ou de services. On verra tout au long de ce manuscrit que ces deux problèmes (structure et sémantique) sont des problèmes qui sont toujours d'actualité.

Nativement et à de rares exceptions, un système ou une application ne sont pas nécessairement conçus pour être intégrés. En effet, au cours des 25 dernières années, la mise en place des systèmes d'information dans les entreprises a mis en évidence une segmentation verticale des applications (gestion administrative, conception, recherche, etc.), sans considération particulière pour l'intégration de ces applications. Ces systèmes d'information se sont constitués progressivement par agrégation de composants initialement conçus pour fonctionner de façon autonome. Ce processus de construction par agrégation a conduit à un niveau minimal d'intégration, insuffisant pour atteindre les nouveaux objectifs et les nouvelles contraintes assignés aux entreprises, à savoir :

- une maîtrise voire une réduction des coûts par des gains d'efficacité, en particulier grâce à une rationalisation des informations (de façon à éviter les multiples versions) et à une réutilisation de celles-ci dans des contextes multiples (gestion financière, audit qualité, recherche),
- des contraintes de qualité globales au niveau de l'entreprise (services offerts, satisfaction des utilisateurs du système),
- des contraintes de sécurité/confidentialité globales,
- une vision globale des informations, indépendante des conditions matérielles de recueil de ces données,
- une intégration des données au sein de véritables documents multimédia, associant données textuelles, données structurées, images,...

Ce postulat sur le peu d'attention quant à l'intégration est éventuellement un peu moins vrai en qui concerne les services ou les composants, mais cela dépend fortement du développeur et du contexte d'utilisation. Dès lors, lorsque l'on veut intégrer des systèmes, des applications, des processus ou des services, on se heurte rapidement à la nécessité d'adapter et de réconcilier des données, des fonctionnalités, des activités. Il faut noter que lorsque l'on parle d'intégration, on parle en fait d'un ensemble de problèmes à résoudre pour parvenir à une vue unifiée et homogène. Cela va de l'architecture à l'interface en passant par les données, les fonctions, les ressources. Il est également important de noter que ce n'est pas parce que l'on va optimiser chaque composant que l'on obtiendra une optimisation globale. Il est donc nécessaire de repenser globalement la fonction du système et de ses acteurs pour en déduire une meilleure définition des composants et de leurs articulations. Bien entendu, ces mutations ne peuvent se faire du jour au lendemain compte tenu des investissements très importants consentis dans la mise au point et le déploiement des systèmes actuels, tant par les entreprises que par les développeurs d'applications.

Avec les bases de données et les systèmes d'information, les environnements de développement de logiciels furent parmi les premiers à s'intéresser au problème de l'intégration. L'analyse qui suit concerne donc principalement le domaine des Ateliers de Développement de Logiciel, même si nous verrons que les concepts et les approches que nous décrivons trouvent des applications au delà de ce domaine. Nous rappelons tout d'abord les motivations qui sous-tendent l'effort d'intégration et nous analysons différentes approches pour y parvenir, en nous focalisant sur le plan technique (d'autres niveaux d'étude – niveau social, niveau économique, etc. – seraient certainement également pertinents et importants, mais sortent du cadre discuté ici).

Nous avons adopté un point de vue à la fois théorique et pragmatique vis à vis du problème ; les communautés des utilisateurs et des industriels du domaine sont forcément plus sensibles à une vision pragmatique, mais il nous a semblé important de ne pas perdre de vue l'aspect théorique de la question. Nous détaillons différentes approches de l'intégration, en soulignant les possibilités et les limites de chacune, ainsi que leur complémentarité.

2.3 Les différentes dimensions

2.3.1 À l'origine : les Ateliers de Développement de Logiciel

Au début de ma thèse (début des années 1990), les travaux autour de l'intégration et de l'interopérabilité ont eu pour domaine d'application les environnements de développement de logiciels. L'objectif était de fournir un support le plus automatisé possible pour gérer les différentes activités du processus de développement d'un logiciel. Les différents outils, systèmes et environnements qui en résultèrent avaient l'intention de rendre plus prévisible le processus, de le rendre moins coûteux, plus facile à gérer et de fournir au final des produits de meilleure qualité. Les environnements qui furent alors proposés, les Ateliers de Développement de Logiciels (*Software Engineering Environments – SEE*), n'eurent pas le succès escompté. Il y a sans doute plusieurs raisons à cela, mais une des principales fut sans doute la confusion entre *agrégation* et *intégration*. En effet, il s'agit là de deux concepts très différents, et sous le terme intégration se cache un certain nombre de propriétés que l'on retrouvera par la suite lorsque l'on parlera *d'intéropérabilité* et de coopération, à savoir :

- le passage à l'échelle – la taille et la complexité d'un ADL posent des problèmes de gestion, de contrôle et de cohérence.
- la diversité – cela va de la diversité des usages, en fonction des utilisateurs (programmeurs, chefs de projet, développeurs d'outils, etc.) jusqu'aux différentes structures des organisations qui utilisent ces ADL (différence de taille, ressources, habitudes et méthodes), en passant par les domaines d'applications ou métiers (finance, données, temps réel, etc.) et les données,
- la technologie – un ADL essaie de fédérer un certain nombre de technologies pour obtenir un système cohérent. Ainsi, on a vu tour à tour les ADLs comme des systèmes d'exploitation étendus, des bases de données complexes, des langages de programmation de haut niveau, des interfaces utilisateur étendues, ou comme un combinaison de tout cela.

De plus, lorsque l'on combine les besoins en termes d'intégration et ceux liés aux ADL, la complexité augmente encore. En effet, il est facile de voir les conflits qui peuvent exister entre les besoins d'ouverture, de paramétrisation ou d'extensibilité d'un côté, et les besoins de cohérence et de prévisibilité d'autre part. Et ce qui est acquis à un niveau local (une entreprise

par exemple) devient moins vrai quand on commence à s'intéresser à des environnements répartis (entreprises virtuelles par exemple). Les concepts, les modèles, les approches, les techniques et les mécanismes doivent alors être analysés et éventuellement adaptés.

Pour résoudre les différents objectifs dédiés aux ADL, Wasserman et Thomas [Was89, TN92] avaient proposé d'aborder le problème de l'interopérabilité et de l'intégration des applications dans un ADL en tenant compte à la fois des différents composants internes d'une application, des services utilisés par cette application ou de l'utilisation possible de l'application. Cette vision peut en fait être généralisée à tout contexte, et nous verrons que l'on peut trouver dans [DJ00] une version un peu différente de la classification de Wasserman.

La classification de Wasserman recense cinq possibilités pour intégrer des applications :
- *l'intégration par la plate-forme*, c'est-à-dire l'ensemble des services systèmes permettant d'offrir de masquer le réseau et le système d'exploitation utilisés par les applications,
- *l'intégration par la présentation*, c'est-à-dire le fait que les applications partagent une interface (*look and feel*) commune du point de vue de l'utilisateur,
- *l'intégration par les données*, c'est-à-dire le support du partage de données entre applications,
- *l'intégration par le contrôle*, c'est-à-dire le support de la notification d'événements entre les applications, et la possibilité d'activer certaines applications à partir d'autres,
- *l'intégration par le processus*, pour contrôler des processus de développement définis.

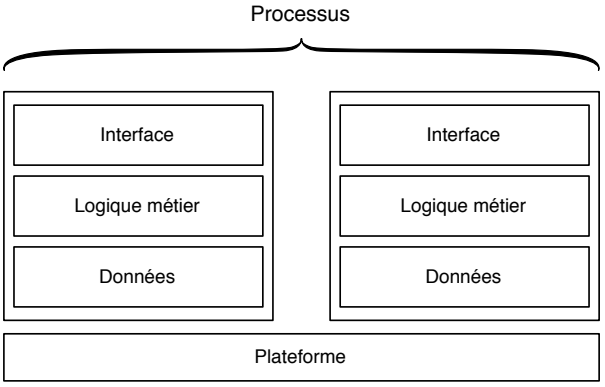


FIGURE 2.1 – Modèle d'application

La figure 2.1 représente un modèle d'intégration de deux applications conforme à la vision de Wasserman. Chaque application est composée de trois parties :
- une couche de gestion des données, pour organiser, enregistrer et accéder aux données que l'application utilise,
- une couche de gestion de la présentation, pour gérer l'interface utilisateur, c'est-à dire les interactions entre l'application et l'utilisateur,
- et une couche de gestion de la logique métier de l'application.

On peut ajouter deux autres parties, externes aux applications, qui sont :
- la plate-forme, dont les applications peuvent utiliser les services,
- le processus qui permet de gérer l'enchaînement des différentes applications selon un ordre précis.

Cette vision synthétique d'une application est bien sûr très caricaturale, mais on va voir qu'elle aide à comprendre les évolutions successives des architectures des systèmes. C'est cette évolution que nous allons maintenant présenter.

2.3.2 L'évolution des applications

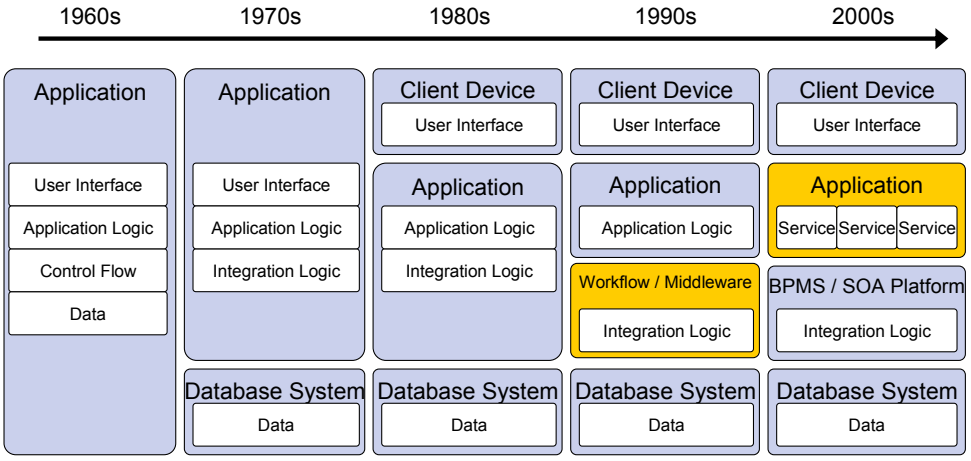


FIGURE 2.2 – Évolution

La figure 2.2 (proposée par M. Rosemann [2]) présente l'évolution successive du modèle tel qu'imaginé par Wasserman. La première version du modèle est conforme à ce que nous avons présenté juste avant. Une application regroupe à la fois les dimensions présentation, logique métier, contrôle et données. L'étape suivante a consisté à séparer la partie dédiée aux données. Plutôt que chaque application utilise ses propres mécanismes de persistance, on factorise ceux-ci dans un système de gestion de bases de données. La deuxième évolution consiste à séparer la partie dédiée à l'interface utilisateur. La troisième étape s'est intéressé à la possibilité de fédérer plusieurs applications. Les moteurs de gestion de workflow (WfMS – *Workflow Management Systems*) permettent de faire travailler ensemble plusieurs applications en organisant le flux de contrôle de ces applications grâce à un processus métier. Les intergiciels permettent de faire cohabiter plusieurs applications, mais sans necéssiter de processus (par exemple CORBA – *Common Object Request Broker Architecture*). La dernière itération consiste à orchestrer plusieurs services afin d'obtenir la fonctionnalité désirée.

Si on reprend la structure d'une application grâce à une structure en couches, comme le montre la figure 2.3. La couche la plus haute concerne les utilisateurs et l'accès aux données et aux services, via par exemple des interfaces offertes par les applications (API par exemple). Les applications peuvent quant à elles utiliser un intergiciel (middleware) permettant d'accèder aux données via la couche données. Il dissimule la complexité de l'infrastructure sous-jacente, il présente une interface commode aux développeurs d'applications, et il fournit un ensemble de services communs (données, transactions,...).

2. Evolution of Information System Architectures, ©Michael Rosemann, QUT.

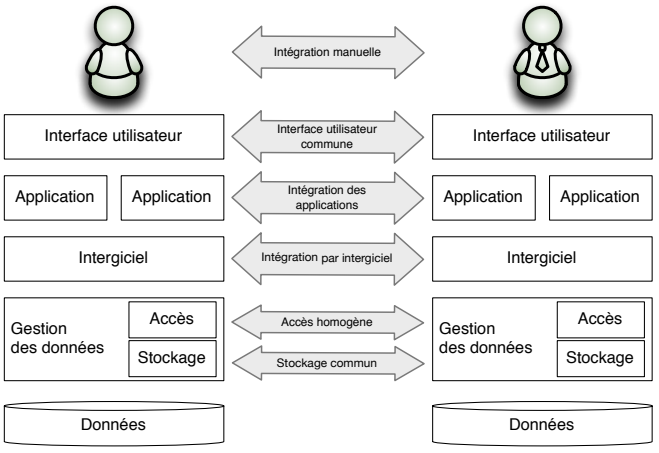


FIGURE 2.3 – Niveaux d'intégration

Le problème de l'intégration peut donc être abordé différemment en fonction de la couche à laquelle on s'adresse. On recense six grandes approches :

- l'intégration manuelle par l'utilisateur
- l'intégration par une interface utilisateur commune,
- l'intégration par les applications,
- l'intégration par l'intergiciel,
- l'intégration par accès uniforme aux données,
- l'intégration par espace de stockage commun.

Nous allons présenter maintenant de manière synthétique les mécanismes d'intégration manuelle, l'intégration par l'interface utilisateur, et l'intégration par l'intergiciel. L'intégration par les données sera développée dans le chapitre 3, et celle par les applications dans les chapitres 4 et 5.

2.4 L'intégration manuelle

L'intégration manuelle permet à l'utilisateur d'interagir avec toutes les informations nécessaires, et de sélectionner celles qui lui paraissent importantes. C'est l'utilisateur qui va se charger de gérer les différences, que cela concerne les interfaces ou les langages de requêtes par exemple. De plus, c'est l'utilisateur qui doit connaître l'emplacement des données, leur représentation et leur sémantique.

Exemple : la gestion de modèles (*Model management*) prend en compte des opérations de haut-niveau sur des modèles : schémas de bases de données, diagrammes UML, etc. L'intégration à ce niveau est clairement une intégration manuelle, et inclut des opérations de transformation : comparaison, fusion, sélection, combinaison [BHP00]. Généralement, on utilise pour gérer l'intégration des algèbres spécifiques aux schémas manipulés, ce qui permet de réduire un peu le travail manuel, mais à l'heure actuelle, cela reste majoritairement une activité d'intégration manuelle.

2.5 L'intégration par l'interface utilisateur

L'idée consiste à ce que toutes les applications adoptent une apparence commune et un comportement semblable du point de vue de l'interface (niveau conceptuel), ou bien partagent un ensemble commun de routines chargées de l'interface utilisateur afin d'assurer une homogénéité d'utilisation (niveau physique). À l'époque à laquelle Wasserman a proposé son modèle, un grand nombre de définitions d'interface étaient disponibles (Motif, OpenLook ou l'interface du Macintosh) et pouvaient offrir une intégration du point de vue de l'interface utilisateur. Les avantages attendus, en bénéficiant d'une intégration au niveau de la présentation des applications, consistaient à avoir *une apparence et un comportement standard*, c'est-à-dire à garantir que les applications se comportent de manière similaire dans des situations identiques, *une cohérence renforcée pour l'utilisation des outils*, *une efficacité accrue* pour l'utilisateur dans son interaction avec l'environnement, et *un renforcement* de la capacité à utiliser dans leur totalité un outil ou un ensemble d'outils.

Les inconvénients générés par les interfaces utilisateurs de plus en plus sophistiquées concernaient le fait qu'il devenait nécessaire de réécrire entièrement l'application spécialement pour le système d'interface choisi, et également la trop grande difficulté à accéder aux fonctionnalités de l'application autrement que par l'intermédiaire de son interface utilisateur.

Aucune de ces interfaces citées précédemment n'a pu s'imposer. Par contre, avec l'avènement du Web [3], l'interface qui est devenue un standard de fait est celle qui repose sur un des standards du Web, à savoir le langage HTML. Au départ, HTML était utilisé dans sous sa forme statique, et désormais, ce sont de véritables applications qui peuvent développées grâce aux évolutions du langage lui-même, ou des évolutions côté serveur ou côté client. On citera les applications Google (*Google Apps*), ou nombre d'applications dont l'interface est celle accessible avec un navigateur. Ainsi, quelque soit l'application sous-jacente, l'utilisateur accède aux fonctionnalités de celle-ci via son navigateur.

L'intégration via une **interface utilisateur commune** (par exemple un navigateur Web) permet donc désormais de ne pas avoir à se soucier de l'emplacement et de la représentation des données (cela est masqué par le navigateur à travers une interface uniforme). Cependant, l'utilisateur doit malgré tout s'occuper de l'homogénéisation et de l'intégration des données.

Exemple : les portails sont un exemple d'interface permettant un accès uniforme aux données. Grâce au portail, chaque utilisateur a accès aux données auxquelles il souhaite accéder, en fonction de ses besoins, et quel que soit le format des données, où et comment celles-ci sont disponibles. La localisation, le format, l'accès aux données sont entièrement masqués et l'intégration des sources de données est faite par le portail qui se charge d'uniformiser, en fonction de la requête, les différences issues de l'intégration.

2.6 L'intégration par le contrôle

Souvent, lorsque l'on analyse les fonctionnalités des différentes applications présentes dans un environnement, on constate que certaines utilisent des fonctions communes. **L'intégration utilisant un intergiciel** [4] permet de répondre à certains problèmes liés à l'intégration, tels que

3. `http://www.w3.org`
4. Intergiciel et Construction d ?Applications Réparties, S. Krakowiak & al.

la persistance par exemple. Ainsi, les applications n'ont pas à se préoccuper de cette fonctionnalité car elle sera prise en charge par une autre application qui fournira cette fonctionnalité, ou par l'intergiciel. Comme la figure 2.1 le montre, il peut être intéressant de réutiliser ou de partager des fonctionnalités communes.

Au niveau conceptuel, l'intégration de la logique métier de plusieurs applications (appelée également intégration du point de vue contrôle) assure que les mêmes services sont utilisés par les applications assurant les mêmes fonctionnalités. Au niveau physique, les applications partagent certains fragments de code. Cette option est particulièrement intéressante, en particulier dans les deux cas suivants : les fonctionnalités partagées sont assurées par une application particulière (par exemple, un service de gestion de versions), ou un service d'un outil donné peut être accédé de manière transparente par un autre outil. Pour bénéficier de ces fonctionnalités, les applications doivent se conformer à une architecture ouverte qui permet à un outil de proposer tout ou partie de ses services (c'est la fin de l'application ≪boîte noire≫ entre données et utilisateur).

De manière générale, l'utilisation du paradigme d'appel de procédure à distance (*Remote Procedure Call*) [BN84] est la solution la plus simple et la plus classique à ce problème. Ce paradigme est basé sur une hypothèse implicite : les procédures d'une application serveur offrent exactement les fonctionnalités que celles requises par une application cliente.

Polylith [PA91, Pur94] est un exemple de cette approche. L'intégration de composants hétérogènes dans le système Polylith est réalisée grâce à NIMBLE, un langage de génération de passerelles d'interfaces. Un programmeur décrit alors avec des directives NIMBLE, la correspondance entre les paramètres d'un service requis et offert. Le système génère alors automatiquement le code qui réalise les adaptations au moment de l'exécution. Il faut néanmoins modifier le style, s'il n'existe pas de correspondance une à une entre les interfaces.

[Len96] présente une évolution qui contient une plate-forme qui permet l'interopérabilité entre plusieurs applications : *Distributed Computing Environment* (DCE). DCE comporte un service d'appel de procédure à distance, un système réparti de gestion de fichiers, un service de gestion des horloges, et un service de gestion de la sécurité. On voit bien dans DCE les problèmes qui sont encore actuellement au cœur de certaines recherches : l'appel de fonctionnalité de manière distante, la répartition des données, la gestion du temps et les aspects liés à la sécurité.

En 1989, l'*Object Management Group* (OMG) est créé et propose en 1995 l'architecture CORBA (*Common Object Request Broker Architecture*). CORBA est une approche de standardisation des interfaces. Grâce à CORBA, les services proposés par un objet application sont spécifiés à l'aide du langage appelé IDL (*Interface Definition Language*) et stockés dans un répertoire particulier (*Interface Repository*). La spécification IDL sert de cible pour les requêtes émanant des clients, et de source pour le serveur. Dans CORBA, l'ORB (*Object REquest Broker*) gère la communication entre les différents objets applications, objets services (appelés également objets systèmes) et les objets bibliothèques (*Common Facilities*). CORBA génère automatiquement les modules de code clients et serveurs (appelés des talons ou stubs) d'invocation des interfaces. Il fournit également un pont entre le client et l'ORB, et un pont entre l'ORB et l'objet serveur. Les clients peuvent invoquer un service par l'intermédiaire d'un stub statique ou par le biais d'une invocation dynamique créé à partir de la spécification IDL à l'exécution. L'ORB valide la requête du client par rapport à l'interface IDL et envoie cette dernière au serveur. Au niveau de l'objet serveur, les arguments sont désérialisés (*unmarshalled*), les méthodes sont exécutées, et les résultats sont retournés.

La définition du langage Java par Sun en 1995 ouvre la voie à plusieurs intergiciels, dont *Java Remote Method Invocation* (RMI) [WRW] et les *Enterprise JavaBeans* (EJB) [MHB06]. Ces systèmes, et d'autres, sont intégrés dans une plate-forme commune, *Java Enterprise Edition*[5].

Microsoft développe à la même époque le *Distributed Component Object Model* (DCOM) [Gri97], un intergiciel définissant des objets répartis composables, dont une version améliorée est COM+. L'évolution actuelle de DCOM, puis de COM+ est la plate-forme .NET[6], plate-forme intergicielle pour la construction d'applications réparties et de services pour le Web.

Une autre approche consiste non pas à générer un module d'adaptation des interfaces à la volée, mais au contraire de standardiser l'accès aux interfaces. Ainsi, à partir des interfaces générées dans l'environnement d'exécution local, on accède aux services disponibles dans d'autres environnements. C'est la cas de SLI [WWRT91], qui définit la compatibilité des types en spécifiant les propriétés des objets et en cachant les différences de représentation. Les types partagés par des programmes qui interopèrent sont décrits en respectant un modèle de type unifié (*Unified Type Model* ou UTM). L'implémentation d'un nouveau service avec SLI passe par la spécification d'une interface de service avec UTM et fournit de nouvelles définitions de type et un langage de liaison. Une fois ce travail fait, la génération automatique de l'interface est réalisée, dans le langage d'implémentation spécifié.

L'inconvénient majeur de SLI et CORBA est que l'interface du service doit être incluse dans le code des clients. En cas de modification des interfaces du serveur, il faut modifier tous les clients. La résolution des incompatibilités de style au niveau des interfaces passe alors par l'utilisation de passerelles d'interfaces ou par la standardisation des interfaces [Kon93]. L'essor de la programmation orientée objet a entraîné des recherches d'interopérabilité non plus au niveau des procédures, mais directement au niveau des objets et à l'encapsulation des constructions (*wrapper construction*) [YS94].

Cependant, il reste toujours certains aspects à implanter pour les applications. De plus, avec le recul, on s'est aperçu que la combinaison de plusieurs intergiciels différents posaient à nouveau des problèmes d'intégration. Par exemple, en CORBA 1.2, certains ORB n'étaient pas intéropérables. C'est ce problème que CORBA 2.0 était censé résoudre avec le protocole UNO (*Universal Network Object*) et ses déclinaisons GIOP (*General Inter-ORB Protocol*) et ESIOP (*Environment Specific Inter-Orb Protocol*).

Les deux problèmes importants de ce genre d'approche sont donc d'une part le besoin de réécrire les outils afin qu'ils s'adaptent à la technologie choisie, et d'autre part, le problème du choix entre plusieurs propositions, telles celles citées, qui divergent peu mais restent incompatibles entre elles.

2.7 Autres exemples

D'autres exemples d'intégration sont plus compliqués à classer :

– l'intégration mise en œuvre dans les réseaux pair-à-pair (*Peer-to-Peer* P2P) est une approche décentralisée de pairs distribués et autonomes dans lesquels les données peuvent être mutuellement partagées et intégrées grâce à des transformations entre les schémas locaux des pairs. L'intégration peut être alors vue, en fonction du degré de fonctionnalités

5. `http://java.sun.com/javaee/`.
6. `http://msdn.microsoft.com/fr-fr/netframework/default.aspx`.

offertes, soit comme une approche homogène d'accès aux données ou comme une approche basée sur l'intégration des applications (plus ou moins manuelle ou automatique).
- l'intégration collaborative [MDV+03] est une forme d'intégration manuelle, basée sur l'idée que les utilisateurs participent à l'intégration des données dont ils ont besoin. Les transformations entre les différents schémas sont soumises aux utilisateurs. En fonction des réponses, les transformations sont affinées. La tâche (difficile) de mise au point des transformations est ainsi distribuée aux utilisateurs.
- dans les systèmes dataspace [FHM05], la co-existence des données (strucutrées ou non) est asurée. À la base, un dataspace permet la recherche de données au-dessus de plusieurs sources de données indépendamment de leur degré d'intégration. Lorsqu'un besoin plus poussé d'intégration est nécessaire, il faut alors revenir aux techniques décrites précédemment.

2.8 Organisation du document

Le document suit la progression de mes différentes contributions. Le chapitre 3 présente mes travaux liés à l'intégration de données. De part mon changement d'équipe en 1999, je me suis intéressé ensuite aux modèles de processus, et plus particulièrement à la modélisation des processus interentreprises. Ma contribution dans ce domaine est présentée dans le chapitre 4. Enfin, le chapitre 5 est consacré à l'étude des services, et en particulier de la façon d'assurer que les compositions de services soient fiables étant donnés un certain nombre de critères de fiabilité. Avant de conclure et de présenter la façon dont nous envisageons la suite de nos travaux, nous donnons un aperçu de nos travaux actuels autour de la prise en compte des aspects non-fonctionnels tels que les propriétés temporelles ou les propriétés autour de la sécurité dans la composition de services.

Note *Les travaux exposés ici ont toujours été menés au sein de groupes ou d'équipes, avec des contributions importantes venant des stagiaires, doctorants, ingénieurs et collègues, qui ont naturellement influencés ma perception des problèmes, et par conséquent les directions que j'ai suivies. Cette influence subjective, bien qu'importante, est difficile à expliciter. Pourtant, nombre des résultats dont je parle dans ce mémoire sont partagés par toutes ces personnes, et cette coopération transparaît dans le document par l'usage répété de la première personne du pluriel (*nous*) qui désigne de manière générique toutes les personnes avec lesquelles j'ai collaboré. Enfin, lorsque la coopération est plus facilement indentifiable, je mentionnerai explicitement les noms des personnes qui m'ont permis de mener à bien ces travaux.*

Chapitre 3

L'intégration de données

3.1 Introduction

L'intégration de données a pour objectif de permettre un accès transparent à différentes sources de données hétérogènes. Elle donne l'illusion à l'utilisateur (ou à l'application) d'avoir accès à un système unique et homogène. Le problème auquel s'adresse l'intégration dans ce contexte est donc le suivant : « fournir un *accès* (requêtes, éventuellement mises àjour) *uniforme* (les sources sont transparentes à l'utilisateur) à des *sources* (pas seulement des BD) *multiples* (même 2 est un problème) *autonomes* (sans affecter le comportement des sources) *hétérogènes* (différents modèles de données, schémas) *structurées* (ou au moins semi-structurées)» (Doucet).

D'après Sheth [SK93], les conflits peuvent être de natures différentes : *structurels*, *sémantiques*, ou *intentionnels*. La différence entre les conflits *structurels* et les conflits *sémantiques* sont caractérisés par les différences entre la structure (*comment les données sont logiquement organisées*) et l'interprétation des données (*que représente une donnée ?*). La distinction entre les 2 n'est pas toujours claire car quelque fois, l'organisation logique véhicule de la sémantique. Une troisième catégorie de conflits – appelés *intentionnels* – concernent les différences ayant trait au contenu des informations. Le problème provient du fait qu'il existe deux mondes distincts lorsque l'on modélise un problème afin de pouvoir proposer une solution informatique. Ces deux mondes sont le *monde réel*, et le *monde modélisé*, qui est une représentation du monde réel. Cela signifie que si l'on considère une entité dans le monde réel, on peut la représenter de diverses manières dans le monde modélisé, et il est alors difficile de définir la sémantique (c'est-à-dire la relation entre un symbole et ce qu'il représente) de cette entité dans le monde modélisé. Il est donc important d'analyser si, dans le monde modélisé, deux entités sont *sémantiquement proches*, *sémantiquement éloignées* ou *totalement différentes sémantiquement*, et ensuite seulement de se préoccuper des différences de représentations.

L'hétérogénéité structurelle permet de distinguer 4 catégories de conflits :
- les types de données : cela concerne les systèmes de types disponibles pour représenter des valeurs.
- le nommage des attributs : cela concerne les attributs qui sont synonymes, homonymes ou antonymes.
- l'aggrégation : cela concerne les choix utilisés pour regrouper certaines informations.
- la généralisation : cela concerne l'existence d'un lien de subsomption entre différents types.

L'hétérogénéité sémantique permet de séparer 3 catégories de conflits :
- les conflits de nommage : cela consiste à identifier les synonymes et homonymes au niveau des valeurs des attributs.
- les conflits de précision et d'échelle : cela fait référence aux unités de mesures ou d'échelle utilisées pour les valeurs des attributs, avec l'utilisation de référentiels différents pour mesurer une valeur.
- les conflits de confusion : cela fait référence à des concepts qui paraissent avoir une signification identique mais qui sont malgré tout différents.

Enfin, l'hétérogénéité intentionnelle permet de séparer 2 catégories de conflits :
- les conflits de domaine : cela concerne les différences de perception au niveau de l'univers du discours.
- les conflits de contrainte d'intégrité : cela concerne les différences portant sur les contraintes associées aux valeurs.

Deux approches sont possibles pour répondre au problème de l'intégration de données. La première approche est celle qui utilise un entrepôt de données (on l'appelle également l'approche matérialisée). L'idée est que l'utilisateur interroge une base de données réelle (l'entrepôt) contenant une copie des données pertinentes des différentes sources considérées. Il s'agit dans ce cas de proposer à l'utilisateur un espace de stockage commun. La seconde approche consiste à utiliser un médiateur (on l'appelle également l'approche virtuelle). L'idée est alors que l'utilisateur interroge ce médiateur chargé de traduire la requête pour interroger les différentes sources et de combiner les résultats obtenus. C'est une intégration virtuelle pour l'accès homogène aux données. Nous détaillons maintenant ces deux approches.

3.1.1 Intégration par accès homogène aux données

L'intégration par accès homogène aux données permet d'avoir un intégration logique au niveau de l'accès aux données. Dans ce cas, l'intégration logique des données se fait au niveau accès aux données. Au niveau global, les applications ont une vue uniforme des données physiquement distribuées, grâce à une présentation virtuelle des données. Au niveau local, les systèmes d'information conserve leur autonomie et peuvent permettre d'accèder aux données via d'autres applications. L'accès aux données, l'homogénéisation, et l'intégration sont faites à l'exécution.

Dans cette approche, le composant principal est le *médiateur* [WWC92]. C'est le composant qui fait le lien entre l'application et les sources de données et qui va permettre de masquer l'hétérogénéité des différentes sources de données. L'intégration de données dans ce contexte peut être divisée en deux catégories (en fait, il en existe d'autres) en fonction de la façon d'intégrer les sources : GAV ou LAV [Len02, RB01, Hal01].

Formellement, Lenzerini (cite Lenzerini) définit un système d'intégration de données comme un triplet $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ où $\mathcal{G}$ représente le schéma global, $\mathcal{S}$ représente le schéma de la source, et $\mathcal{M}$ représente la transformation entre $\mathcal{G}$ et $\mathcal{S}$. L'intégration *Global-as-view* (GAV) consiste à définir le schéma global en fonction du schéma des sources à intégrer : les relations du schéma global sont des vues basées sur les schémas des sources de données. Les données restent autonomes puisque contenues dans les sources. Une requête q sur le schéma global est traduite en termes de schémas des sources (dépliement de la requête), puis la requête dépliée est évaluée sur les sources. Cette approche présente l'avantage d'être très simple (reformulation de la requête), mais elle souffre

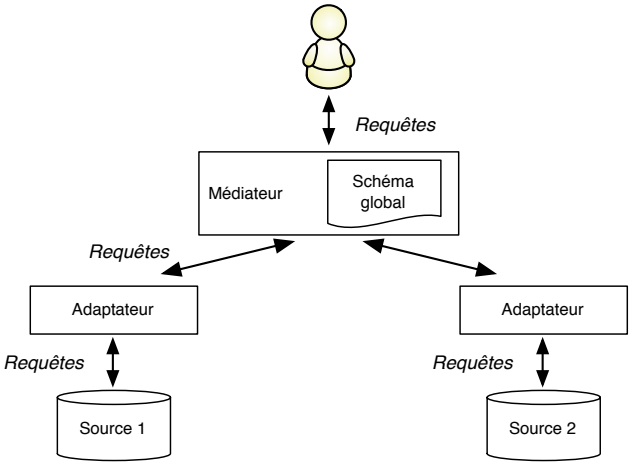


FIGURE 3.1 – Approche virtuelle

d'un manque de modularité puisque l'addition d'une nouvelle source de données nécessite de redéfinir le schéma global.

Dans l'approche GAV, la transformation $\mathcal{M}$ associe à chaque élément g de $\mathcal{G}$ une requête $q_{\mathcal{S}}$ sur $\mathcal{S}$. Une transformation GAV est donc un ensemble d'assertions, une pour chaque élément g de $\mathcal{G}$ qui a la forme $g \rightsquigarrow q_{\mathcal{S}}$.

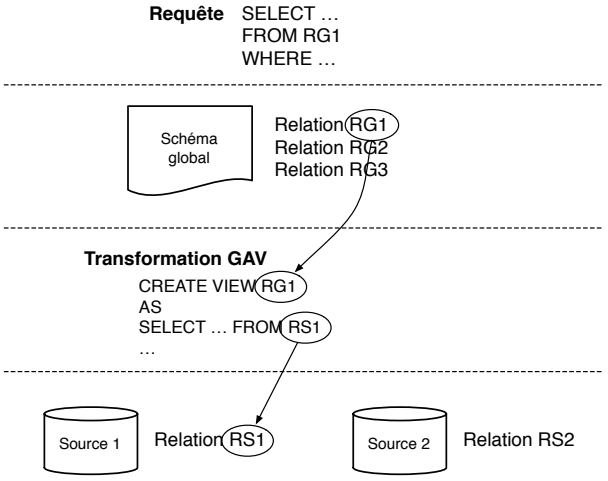


FIGURE 3.2 – Transformation GAV

L'intégration *Local-as-view* (LAV) consiste à définir les schémas locaux en fonction du schéma global. Ainsi, les relations des schémas locaux sont définies comme des requêtes (vues) sur le schéma global, i.e. les sources sont des vues. Les données sont également autonomes et restent dans les sources. Une requête q sur le schéma global doit être traduite en termes des schémas locaux (réécriture des requêtes). Cette approche présente l'avantage d'avoir une bonne modularité

puisque l'addition de nouvelles sources est simple. Chaque source reste autonome. Par contre, la reformulation de la requête peut être complexe. En effet, on utilise un mécanisme d'inférence qui permet d'exprimer les composants du schéma global en termes de composants des sources.

Dans l'approche LAV, la transformation $\mathcal{M}$ associe à chaque élément s de $\mathcal{S}$ une requête $q_{\mathcal{G}}$ sur $\mathcal{G}$. Une transformation LAV est donc un ensemble d'assertions, une pour chaque élément s de $\mathcal{S}$ qui a la forme $s \rightsquigarrow q_{\mathcal{G}}$.

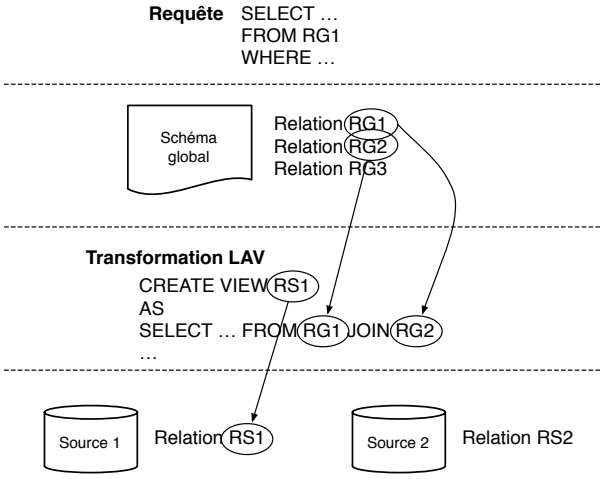


FIGURE 3.3 – Transformation LAV

Avec la montée en puissance du Web, de nombreuses applications doivent désormais intégrer des sources de données hétérogènes. Le problème de l'intégration de ces sources de données est donc très important. Pour résoudre ce problème, on peut agir à trois niveaux : au niveau des requêtes, au niveau du schéma global, au niveau de la transformation schéma global/schéma des sources de données ou enfin au niveau des sources de données.

Exemple : les systèmes de médiation sont un exemple classique d'accès uniforme à de multiples sources données. Ils représentent une solution basée sur un point d'accès unique permettant d'interroger plusieurs sources de données. Le rôle du médiateur est de traiter une requête globale, et d'envoyer des sous-requêtes aux sources de données locales. Les résultats obtenus sont alors combinés/intégrés et présentés à l'utilisateur.

Les systèmes de gestion de bases de données fédérées (*Federated database systems* – FDBMS [SL90]) sont également un exemple de ce type d'intégration. Elles offrent une solution à l'accès uniforme aux données en intégrant logiquement les données appartenant aux systèmes de gestion de bases de données locaux. Cependant, un tel système est un SGBD à part entière puisqu'il possède son propre modèle de données, supporte les requêtes, les transactions, et gère le contrôle d'accès au niveau global.

3.1.2 Intégration via un stockage commun

L'intégration par stockage commun des données est la deuxième approche possible (appelée également approche matérialisée) pour intégrer des sources données hétérogènes. Elle permet de réaliser une intégration physique des données. Dans ce contexte, l'intégration des

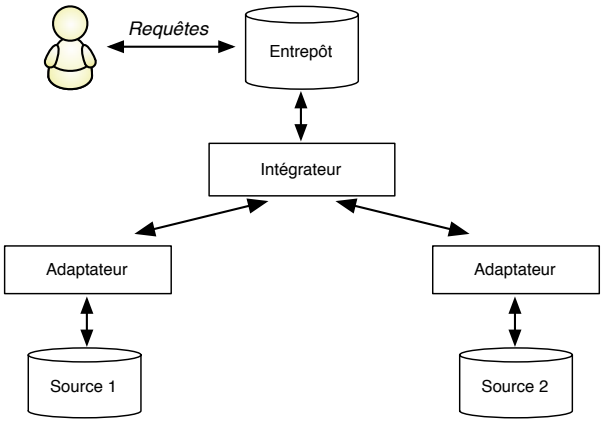


FIGURE 3.4 – Approche matérialisée

données est faite en transférant les données vers un nouvel espace de stockage. On appelle ce processus de construction le processus ETL pour Extraction - Transformation - Chargement (*Extract - Transform - Load*). Les sources de données locales peuvent alors être soit supprimées, soit conservées. En général, ce mode d'intégration fournit un accès rapide aux données puisque ces dernières sont disponibles dans une seule et même source. Cependant, si les sources de données locales sont supprimées, les applications qui manipulent les données distantes doivent être soit transférées également, soit bénéficier de débits suffisants. De plus, le problème de la mise àjour des données de l'entrepôt est difficile. En effet, si les données locales sont conservées, une mise àjour régulière et adaptée de l'entrepôt doit être envisagée, ce qui nécessite d'exécuter à nouveau le processus ETL. Enfin, la définition du schéma de l'entrepôt n'est pas une tâche facile, car il faut être capable de construire un schéma alors que l'on n'a pas forcément toutes les informations disponibles sur les sources de données.

Exemple : les entrepôts de données (*data warehouse*) sont des solutions qui réalisent ce type d'intégration. Dans le cas des entrepôts de données, les données provenant de plusieurs sources sont extraites (*on-line transaction processing systems*, OLTP), transformées et conservées dans l'entrepôt. Une analyse est alors possible sur les données intégrées (*online analytical processing* – OLAP).

Un autre exemple de stockage commun est le service Amazon S3 (*Simple Storage Service*) qui est un espace commun de stockage en ligne offert par Amazon. Amazon S3 ne fonctionne pas comme les autres espaces de stockage en ligne : c'est un service Web qui propose un stockage illimité, tout en faisant une abstraction totale de la couche physique. L'utilisateur n'a pas à se soucier de l'adresse du serveur, de sa connectivité, de problématiques liés à l'emplacement géographique, ou à l'espace disque. Il peut alors interagir avec ses données (regroupées dans des *buckets*) en utilisant les protocoles REST ou SOAP.

3.2 Contribution

Le problème auquel je me suis intéressé dans ma thèse [1] concernait l'intégration de données basée sur la conversion et la restructuration des données. Le contexte d'application de mes tra-

vaux concernait les Ateliers de Développement de Logiciel, dans lesquels le problème consistait à permettre à un outil d'accéder aux données d'un autre outil, indépendamment de leurs différences et des conflits (*structurels*, *sémantiques*, ou *intentionnels*). Pour cela, l'approche que nous avons proposée reposait à la fois sur l'intégration des données et sur leur transformation. Pour cela, nous avons défini un niveau intermédiaire qui autorisait chaque outil à déclarer sa propre structure de données (son modèle), et un ensemble de règles permettant de transformer un modèle dans un autre modèle à la fois au niveau de la structure et des données contenues dans ce modèle. Cette définition d'un adaptateur générique basé sur la description des sources de données représentait une avancée par rapport à la méthode habituelle qui était ad-hoc, et donc difficilement réutilisable. Notre proposition permettait d'avoir une architecture générique qui évitait d'avoir plusieurs adaptateurs, et qui réunissait le médiateur et l'adaptateur générique.

Notre proposition contenait 2 composants. Le premier était le modèle de représentation. Le deuxième était un ensemble de primitives permettant de passer d'un modèle à un autre.

Pour capturer les données provenant d'une application, on utilisait un modèle de représentation structuré sous forme d'arbre étiqueté et ordonné. Un format de données était donc modélisé comme un arbre, un peu comme un document XML est actuellement modélisé par un XSD ou une DTD. Par rapport à XML, notre modèle était entièrement structuré, et non semi-structuré. Un des avantages du modèle était qu'il permettait de passer d'une collection d'éléments à un ensemble de valeurs, et réciproquement, d'un ensemble de valeurs à une collection ordonnée d'éléments.

Le deuxième composant était un ensemble d'opérateurs permettant de manipuler un modèle afin d'obtenir un nouveau modèle. Cet ensemble d'opérateurs permettait de définir une transformation pour passer d'un modèle de représentation r_1 à un modèle de représentation r_2. Une transformation est une composition de règles qui permettent d'agir à la fois sur la structure avec la possibilité de ré-ordonner ou de restructurer l'arbre de départ, ou d'agir sur le contenu des nœuds de manière à gérer l'hétérogénéité intentionnelle (domaines et contraintes d'intégrité).

Pour gérer ces deux composants, nous avions proposé un médiateur qui réalisait l'intégration d'un modèle à partir d'un autre modèle, en respectant les étapes suivantes : (1) importation des deux modèles (source et cible) dans le médiateur, (2) comparaison des deux modèles et transformation du modèle source dans le modèle cible, et (3) exportation des données du modèle source vers des données compatibles avec le modèle cible.

3.3 Synthèse

En dépit des remarquables avancées qui ont été faites sur la problématique de l'intégration de données, des concepts aux algorithmes, en passant par les systèmes commerciaux, cela reste un problème difficile. Les raisons sont multiples. Premièrement, l'intégration de données reste un problème important du fait que de plus d'entreprises partagent désormais certaines de leurs sources de données, dans le cadre de coopération ou de projets communs. On souhaite donc désormais pouvoir intégrer les données provenant de différents sources, ou des informations de différents types. Du coup, on doit manipuler des modèles différents, des interfaces différentes. De plus, on se heurte souvent au problème de l'information disponible sur les données à intégrer : leur sémantique, le lien entre les données, le fait que le savoir à propos des données est principalement maîtrisé par les personnes qui en ont la charge,... Il faut donc être en mesure de :

- *comprendre* les données à intégrer, c'est-à-dire être capable de donner un sens aux données, ainsi qu'à leurs relations,
- *standardiser* éventuellement les données, que cela soit au niveau du schéma par exemple, au niveau de la sémantique, ou au niveau de la qualité des données : cohérence, complétude, précision, validité,...
- *exécuter* l'outil qui permettra de rendre l'intégration effective. Concernant les outils, on sélectionnera soit un outil ETL, ou soit un outil qui gère la fédération des sources de données en fonction de la méthode choisie, et de critères tels que les besoins (performance, volume), les types de données (hétérogènes, majoritairement relationnelles), leur disponibilité, ou leur réplication.

L'intégration de données reste également un problème important du fait des problèmes sémantiques. En effet, le problème de l'intégration est plus qu'un problème structurel et technique. Nous venons de montrer qu'intégrer différents modèles de données est compliqué. Intégrer différents modèles dont les sémantiques sont hétérogènes est encore plus compliqué. C'est ce qu'avait souligné très tôt Sheth dans un papier assez fondateur [SK93]. Intégrer deux sources de données dans lesquelles certaines données sont proches structurellement, mais complètement hétérogènes sémantiquement conduit à des résultats désastreux. Être en mesure d'exprimer de façon explicite et précise la sémantique des données à intégrer est donc essentiel si l'on veut que l'intégration soit la plus probante possible. Il faut noter qu'aucune solution actuelle à l'intégration ne répond complètement à ce challenge. Les ontologies constituent un modèle de données représentatif d'un ensemble de concepts dans un domaine, ainsi que les relations entre ces concepts. Elles peuvent donc contribuer à améliorer le problème de l'hétérogénéité sémantique. Comparées avec d'autres modèles de classification (taxonomies, thesaurus, mots clés), les ontologies permettent de modéliser plus précisément et de manière plus complète des domaines [HS97].

Articles

Object and Schema Versioning and Restructuring in Databases
Nacer Boudjlida, Mourad Bouneffa, Olivier Perrin.
Proceedings of 16th DATASEM, 1996, October, Brno (CZ).

An Open Approach for Data Integration
Olivier Perrin, Nacer Boudjlida.
Proceedings of the 2nd International Conference on Object-Oriented Information Systems, OOIS' 95, 1995, December, Dublin.

A Formal Framework and a Procedural Approach for Data Interchange
Olivier Perrin, Nacer Boudjlida.
Proceedings of the 3rd International Conference on System Integration, ICSI'94 (IEEE), 1994, August, São Paulo, Brazil.

Towards a Model for Persistent Data Integration
Olivier Perrin, Nacer Boudjlida.
CAiSE'93 The Fifth Conference on Advanced Information Systems Engineering, LNCS 685, 1993, June, Paris.

Chapitre 4

L'intégration par les processus

4.1 Introduction

Les systèmes de gestion de processus (*Workflow management systems* – WfMS) permettent de définir et d'exécuter des *processus métier*. Un processus métier est un ensemble de procédures et d'activités liées par un flot de contrôle et/ou par un flot de données afin de réaliser un objectif métier, en général au sein d'une structure organisationnelle définissant des rôles et des relations fonctionnelles. Chaque activité peut être exécutée par une application particulière ou par un utilisateur. Un processus représente donc une approche basée sur l'intégration par les applications puisque l'utilisateur n'interagit qu'avec le processus, et c'est à travers lui que l'utilisateur possède une vision intégrée des applications sous-jacentes (et de leurs données).

Un système de gestion de processus possède 3 dimensions. La première dimension concerne le processus métier qui définit le *quoi*, c'est-à-dire les activités à exécuter et l'ordre dans lequel elles doivent s'exécuter. La deuxième dimension, orthogonale au processus, concerne l'organisation. Cette dimension définit la structure organisationnelle et va permettre d'associer aux activités définies un responsable (*qui*). Enfin, la dernière dimension correspond au niveau technique, et permet de définir le lien entre une activité et *quelles* ressources (applications, sous-processus, personnes ou données) sont nécessaires à cette activité. À travers le processus, on va donc avoir besoin d'intégrer plusieurs applications et plusieurs sources de données.

L'exécution d'une activité comprend plusieurs composants (figure 4.1). Lorsqu'un participant démarre un bon de travail, le WfMS, grâce au modèle de processus, connaît l'activité associée à ce bon de travail. Il construit alors une instance d'un conteneur d'entrée qui va contenir les données nécessaires à l'exécution de l'activité (1). Le WfMS invoque alors l'application ou le sous-processus implantant l'activité (2) et cette application reçoit l'instance du conteneur d'entrée comme paramètre (3). Ensuite, le WfMS réalise les opérations correspondantes en interagissant avec l'utilisateur et les éventuelles bases de données (4) tout en construisant le conteneur de sortie, dans lequel il dépose les données (5). À la fin de l'exécution, le conteneur de sortie contient les résultats de l'application ou du sous-processus, et ce conteneur est renvoyé au WfMS. Lorsque le programme se termine, le WfMS évalue la condition de sortie de l'activité. Si celle-ci n'est pas satisfaite, le bon de travail est remis dans les corbeilles des ressources qui possèdent le bon rôle. Sinon, l'algorithme de navigation choisit la (les) prochaine(s) activité(s) à exécuter en fonction du modèle de processus.

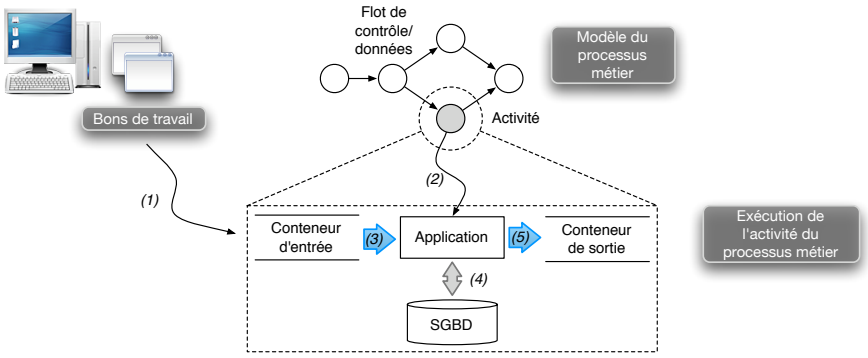


FIGURE 4.1 – Exécution d'une activité.

Il existe de nombreux langages de spécification permettant de définir des processus métier. La *Workflow Management Coalition* (WfMC) a proposé d'unifier ces différents langages à travers un modèle de référence (*Workflow Reference Model*[1]). Le modèle fournit un ensemble standard d'interfaces et de formats d'échange entre les composants d'un système de gestion de processus. La modélisation des processus permet donc d'obtenir une intégration des différentes applications associées aux activités du processus. Bien sûr, les avantages de la modélisation de processus ne concernent pas uniquement cet aspect. Un autre aspect important est que la modélisation des processus permet, grâce à l'utilisation de formalismes représentatifs, d'analyser, de spécifier, de visualiser, de valider et de documenter les processus étudiés. L'objectif de la structuration des processus est de proposer des caractéristiques syntaxiques aidant à la définition de processus corrects. Pour réaliser cet objectif, différentes études, basées en particulier sur des travaux dans le domaine des réseaux de Petri (WF-net bien structuré [vdAtH00, vdAvH02]), ont conduit à une notion simple de « processus bien structuré ». Intuitivement, un processus est bien structuré si :

- il a une seule place initiale et une seule place finale, et toutes les tâches sont sur un chemin allant de la place initiale à la place finale. En d'autres termes, il s'agit d'un graphe connexe avec un point d'entrée et un point de sortie ;
- les opérateurs d'ordonnancement sont bien imbriqués et bien parenthésés. Typiquement, cela correspond aux conditions suivantes :
  - à un « branchement multiple » (*AND-split*) correspond une «jonction multiple » (*AND-join*), à un « choix exclusif » (*XOR-split*) correspond une «jonction simple » (*XOR-join*), à un « choix multiple » (*OR-split*) correspond un (*OR-join*) ou (non exclusif),
  - si les combinaisons mettent en œuvre une «jonction multiple » ou un « discriminateur » comme définis dans la section précédente ;
- les processus entre les parenthèses sont eux-mêmes bien structurés.

Le paradigme actuel pour modéliser les processus métier est orienté uniquement vers le processus, c'est-à-dire les flux de contrôle des activités. Cette façon de modéliser les processus possède un certain nombre d'avantages, dont celui de pouvoir passer à un niveau d'abstraction suffisant pour pouvoir raisonner sur les propriétés des processus étudiés : complétude, structuration, situation de blocage, validation grâce à des formalismes tels que les réseaux de Petri,

1. `www.wfmc.org/standards/docs/tc003v11.pdf`

les graphes orientés, le calcul de processus (π calculus),... Cependant, ces modèles ne sont plus adaptés lorsque l'on s'intéresse à des processus plus complexes, tels que ceux dont nous allons parlé maintenant dans la suite de ce chapitre.

4.2 Contexte

4.2.1 Introduction

J'ai montré dans le chapitre précédent que l'intégration de données ne constitue qu'un des aspects du problème de l'intégration. De part mon changement d'équipe [2], je me suis intéressé aux processus, et plus particulièrement aux processus interentreprises créatifs. Ces processus sont un peu différents des processus métier classiques à cause de leur nature distribuée entre plusieurs entreprises (*interentreprises*) et de leur nature dynamique car ils sont potentiellement de longue durée, et donc propices aux changements (*créatifs*). Dans ce chapitre, je vais donc présenter mes contributions aux problèmes que posent les procesus interentreprises créatifs.

Le contexte dans lequel nous avons mené nos recherches était celui des « *entreprises virtuelles* » [MC94, GRST97]. Une entreprise virtuelle est d'abord une entreprise, dans le sens classique du terme, c'est-à-dire qu'elle possède et réalise des objectifs en implantant des processus et en les exécutant. Cependant, cette entreprise est virtuelle dans le sens où elle est distribuée à travers le temps, l'espace et que les processus peuvent être distribués dans plusieurs organisations. On distingue donc une entreprise virtuelle par le fait qu'elle est issue d'une alliance temporaire autour d'un projet et qu'elle est ainsi amenée à composer certains fragments de processus, à partager certaines ressources et compétences provenant de plusieurs entreprises tout en garantissant que les entreprises participantes vont conserver leur autonomie [Bit03].

Les entreprises virtuelles sont nées du besoin des entreprises classiques de coopérer afin d'abaisser les coûts de développement de leurs produits. Les objectifs des entreprises virtuelles sont d'une part de pouvoir partager certaines informations, de pouvoir mettre en commun certaines des compétences des partenaires, et de pouvoir achever un projet commun dans des délais plus courts que s'il avait fallu le mener à bien seul. Ainsi, certains processus qui étaient auparavant internes aux entreprises sont devenus interentreprises (appelés également multi-entreprises). Cela signifie par exemple qu'un processus peut être scindé en plusieurs parties (un processus interentreprises dépasse les frontières d'une entreprise) et que chaque partenaire implante une partie seulement du processus (sous-traitance). Cela peut également signifier que la même activité peut être implantée par plusieurs partenaires, et qu'il faut alors gérer la synchronisation des données produites (co-développement).

Une entreprise virtuelle nécessite donc des mécanismes de composition relativement flexibles (modification de la ressource chargée de réaliser une activité, modification du processus, modification des participants à une activité donnée, adaptation de processus issus de projets précédents avec modifications mineures ou majeures...), et il est maintenant établi que les architectures centralisées issues du monde des systèmes d'information qui sont utilisées dans les systèmes de gestion de processus classiques ne sont pas adaptées au support des entreprises virtuelles. De plus, la distribution fait qu'il doit exister un certain niveau d'interopérabilité entre les partenaires, que ce soit au niveau des plates-formes, des applications, ou des processus utilisés. L'objectif

2. J'ai rejoint le projet ECOO en 1999.

est alors de permettre aux partenaires de pouvoir s'échanger les données et les informations nécessaires à la réalisation du projet commun, de pouvoir réunir rapidement les compétences de chaque partenaire, de pouvoir coordonner leurs processus respectifs, et de pouvoir améliorer le temps de réalisation des différents objectifs.

Nous avons été amené à travailler dans le domaine de la conception, plus précisément dans le domaine automobile, où les processus de conception fortement itératifs ont été remplacés par l'ingénierie concurrentielle où plusieurs équipes de constructeurs travaillent en parallèle sur le même projet [HH02, WMS02] afin de proposer des véhicules reposant sur la même plate-forme. Ce fut d'abord l'action de développement AEE – Architecture pour l'Electronique Embarquée – qui s'intéressait à la maîtrise de la complexité croissante pour la définition de l'Architecture Système et le développement des logiciels embarqués dans le domaine de l'automobile. Un des objectifs du projet était de définir un processus permettant aux constructeurs et aux équipemetiers de coopérer pour la réalisation d'architectures électroniques des véhicules. Le projet national AEE s'est achevé en 2001, et il s'est poursuivi dans le cadre Européen du programme ITEA de EUREKA. Le projet nommé EAST-EEA a démarré en juillet 2001. Il comprenait les partenaires de AEE sauf SAGEM et EADS, et avait comme nouveaux partenaires des constructeurs et des équipementiers européens du domaine de l'automobile, à savoir VOLVO, AUDI, BMW, FIAT et DAIMLER. Dans ce projet, un des lots visait à offrir un support pour des processus interentreprises de développement des architectures électronique embarquées des véhicules. Ces processus possèdent les caractéristiques suivantes. Ils sont interentreprises, chaque partenaire amenant son expertise et ses connaissances, ainsi que ses besoins et ses contraintes. Ils sont coopératifs, puisqu'une architecture électronique embarquée est conçue par tous les partenaires, constructeurs et équipementiers. Ils sont partiellement définis, car de nombreuses contraintes ou modifications apparaissent au fur et à mesure du développement (à cause de l'intégration de nouvelles technologies en cours de projet, ou à cause de correction sur des fragments de l'architecture par exemple). Les interactions entre les partenaires sont assez complexes et sont différentes suivant les cas : par exemple, certaines activités sont plutôt dirigées par les constructeurs, alors que d'autres sont plutôt dirigées par les équipementiers. De même, certaines activités peuvent être répliquées chez plusieurs équipementiers, mais elles ne sont pas tout à fait identiques, afin de prendre en compte la spécificité de chaque partenaire. Les processus doivent également être garants de l'autonomie de chaque partenaire, et du respect de certaines informations privées. Enfin, ces processus doivent être dynamiques, puisqu'ils sont ajustés et adaptés très fréquemment au cours de leur exécution.

Pour supporter de tels processus, on doit donc :
- intégrer les processus en portant une attention particulière à l'hétérogénéité au niveau logiciel et plate-forme,
- coordonner les interactions complexes qui existent dans ces processus interentreprises créatifs (composition et/ou agrégation de services, transactions avancées, recouvrement,...), et leurs besoins en terme de flexibilité,
- respecter les caractéristiques d'autonomie (public/privé), et de distribution dans le temps et l'espace requises par la notion d'entreprise virtuelle,
- fournir les technologies permettant de modéliser, de composer et de coordonner les fragments de processus des partenaires.

4.2.2 Les processus interentreprises

Les processus interentreprises nécessitent de s'intéresser à la fois à l'intégration de données afin de pouvoir s'échanger/partager des documents, même si les entreprises ne possèdent pas les mêmes outils, et à l'intégration du contrôle, afin de synchroniser tous les participants d'un processus. Pour assurer cette synchronisation, les processus interentreprises doivent *coordonner* les activités des partenaires. Ils gèrent l'échange des résultats entre les partenaires, coordonnent la réalisation des processus des partenaires, et contrôlent le respect des conventions prédéfinies. Étant données les propriétés propres aux processus interentreprises créatifs (distribution, dynamicité, autonomie), la coordination des activités peut prendre plusieurs formes. Une activité peut être attribuée à une seule entreprise, ou la même activité peut être dupliquée chez plusieurs entreprises concurrentes pour, par exemple, comparer ensuite leurs résultats. De même, des partenaires de différentes organisations peuvent effectuer ensemble une activité. Si on connaît le sous-processus de réalisation de cette activité, les partenaires peuvent se voir attribuer chacun une tâche individuelle. Il peut aussi y avoir des activités coopératives qui ne sont pas décomposables, mais sont réalisées par des partenaires des différentes entreprises.

On peut noter au passage que la vision de Wasserman n'était donc pas dénuée d'intérêt puisqu'elle prenait déjà en compte le fait de pouvoir synchroniser des outils avec des échanges de messages ou d'événements. En effet, à partir du moment où plusieurs participants partagent des données, ou travaillent à plusieurs sur un document (ou une ressource plus généralement), ils doivent non seulement pouvoir échanger facilement leurs travaux, et ils doivent également pouvoir se coordonner afin de ne pas refaire un travail déjà fait, ou au contraire défaire un travail existant. Il peut donc être intéressant de combiner dans une solution commune les aspects données et processus. C'est l'idée qui a mené à la proposition de l'approche PointSynchro. Avant de présenter cette approche, nous présentons ci-dessous les problèmes liés aux processus créatifs dans un environnement de travail interentreprises, avant de détailler les problèmes liés à la coordination des partenaires au sein d'une activité coopérative que nous allons illustrer par un exemple.

4.2.3 Problèmes liés à l'environnement interentreprises

Autonomie Si chaque entreprise est consciente du besoin de coopérer pour la réalisation du projet commun, elle souhaite malgré tout conserver son autonomie (infrastructure, méthodes, outils), conformément à ses propres intérêts. La coordination doit se baser sur les processus déjà existants et doit être indépendante de la façon dont sont réalisées et exécutées les activités au sein des entreprises. De même, la coopération doit être effective en dépit de l'hétérogénéité au niveau du matériel, des logiciels et de la modélisation dans les différentes entreprises.

Confidentialité Un autre besoin clairement identifié dans un environnement de travail interentreprises est l'exigence de respect du besoin de confidentialité des entreprises et du savoir-faire (processus métiers internes). Pour cette raison, chacun des partenaires essaie de ne dévoiler que le minimum nécessaire d'information concernant la réalisation d'activités au sein de son entreprise. En effet, la coopération autour d'un projet commun n'exclut pas la compétition et la concurrence entre les participants sur d'autres projets.

Attribution dynamique des activités Dans un environnement interentreprises, plusieurs partenaires peuvent avoir les mêmes compétences et peuvent effectuer la même activité. Au moment de la définition d'une activité, on peut donc avoir le choix. On peut aussi ne pas avoir encore trouvé de partenaire pour l'activité. C'est pour cela que la définition des activités n'est pas liée à un participant. Le partenaire effectuant l'activité est dynamiquement choisi au moment de son exécution, et l'affectation peut se faire à n'importe quel moment avant le début de l'activité ou même être remise en question à lors de l'exécution du processus.

4.2.4 Problèmes liés au domaine des processus créatifs

Un des problèmes majeurs liés aux processus créatifs vient de leur **nature dynamique**. En effet, dans l'environnement fortement dynamique d'une entreprise virtuelle, on peut avoir besoin de changer le processus de travail au fur et à mesure de son déroulement. Ceci peut être provoqué par des changements dans les exigences des clients ou bien par l'apparition de nouvelles possibilités technologiques. Bien que les grandes étapes du processus soient définies au début de manière assez précise, la conception de quelque chose de nouveau rend difficile l'existence d'un processus prédéterminé [GSCB99], et le processus sera sans doute soumis à des modifications au cours de son exécution.

De plus, le bon déroulement des opérations distribuées nécessite le support d'exécutions transactionnelles. La durée assez importante des activités de coopération et leur nature multi-participants ne permettent pas leur encapsulation par des transactions traditionnelles dotées de propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité). Le principe du « tout ou rien » de l'Atomicité [GR93] peut causer de grandes pertes de travail lors d'activités de longue durée. Pour utiliser les progrès « partiels » des partenaires, on doit autoriser les interactions entre activités en cours d'exécution. L'échange de résultats intermédiaires est indispensable pour la réalisation des activités coopératives. Cela implique la nécessité de relâcher les propriétés d'Atomicité et d'Isolation. Cependant, le non-respect de l'Atomicité compromet les mécanismes traditionnels d'annulation. Cela provoque le besoin des nouvelles méthodes de compensation qui annulent logiquement les effets d'une activité ainsi que des méthodes de recouvrement qui permettent la reprise après des pannes et assurent la fiabilité du système. Nous devrons également utiliser des nouveaux critères de sérialisabilité adaptés aux activités coopératives [GPS99].

4.2.5 Problèmes liés aux activités coopératives

Une activité coopérative est une activité menée par un ensemble de partenaires provenant de différentes entreprises qui travaillent simultanément à la réalisation d'un projet commun. Ainsi, il est fréquent qu'une activité coopérative dépende des résultats d'autres activités qui se déroulent en parallèle. Pour permettre à un partenaire de se situer par rapport aux autres, les activités coopératives sont constituées de multiples interactions, et la nature de ces interactions peut être très complexe. Ainsi, elles peuvent être aussi bien synchrones qu'asynchrones, à deux ou à plusieurs, imprévues ou planifiées, instantanées ou de durée importante. Les interactions peuvent s'effectuer soit via des échanges directs, soit via le partage d'objets communs, et l'élaboration du résultat commun passe généralement par l'échange de propositions successives concernant l'objectif souhaité. La synchronisation d'activités parallèles est connue comme potentiellement source d'erreur (par exemple dead-lock et livelocks), et il est difficile d'assurer la cohérence

pour l'ensemble du processus interentreprises [AMVW99]. Une solution à ce problème consiste à utiliser les résultats intermédiaires, chaque partenaire pouvant ainsi synchroniser son travail avec les autres participants de l'activité. L'échange des résultats intermédiaires permet d'optimiser le processus. L'objectif est alors d'éviter une trop grande divergence entre les versions d'un même objet pour faciliter ensuite la convergence vers l'objectif commun. En effet, un trop grand écart entre les versions peut se révéler tellement difficile à réconcilier que le coût des efforts de convergence peut dépasser le gain de la cooperation [MSMO02].

Pour éviter une trop grande divergence entre les versions des partenaires, les périodes de travail individuel sont encadrés par des étapes de planification et d'évaluation [Kva00]. La planification représente soit les objectifs initiaux soit une demande de changements suite à une évaluation des résultats livrés. L'évaluation s'effectue généralement par les partenaires qui prennent connaissance des différentes versions proposées. En cas d'incompatibilité avec leur vision, ils peuvent demander des corrections. La politique d'évaluation peut varier suivant les relations entre les partenaires. Le moment de la coordination n'est pas fixé précisément, de même que le nombre de synchronisations. Cela dépend de l'évolution du processus et de l'expérience des participants. L'objectif est de se coordonner assez souvent pour éviter une trop grande divergence dans les propositions. Le processus de coordination au sein d'une activité coopérative est un processus itératif avec un échange initial des propositions, suivi généralement dans les étapes suivantes par une amélioration du résultat.

4.3 Approches possibles

Les systèmes de workflow classiques sont très performants lorsqu'il s'agit d'exécuter des procédés dont on peut définir précisément les activités. Par contre, il est désormais avérés qu'ils posent plus de problèmes lorsqu'il s'agit d'exécuter et de coordonner des activités coopératives non nécessairement séquentielles. Fin 2003, Ellis déclarait que ≪*current approaches for coordinating cooperative activities do not fit because implementations of workflow tend to be coercive, isolationistic, and inflexible whereas the natural interaction of people frequently incorporates flexibility, opportunistic behaviors, social awareness, and compromise*≫. Lorsque l'on utilise un système de workflow classique pour modéliser et exécuter de telles activités, on se heurte à un certain nombre de problèmes. Si ils représentent un bon moyen de coordination interne pour une entreprise, leur gestion centralisée de toutes les données d'exécution les rendent difficilement adaptables à la coopération interentreprises. Les propositions qui visent à les rendre plus dynamiques sont difficilement applicables pour les processus coopératifs car elles nécessitent l'anticipation de toutes les possibilités de changement. Par ailleurs, les différents outils actuels manquent d'interopérabilité. Enfin, les partenaires doivent attendre la fin d'une activité s'ils veulent pouvoir évaluer ou utiliser le résultat de cette activité. Cela peut poser problème si la durée de l'activité est importante, et si l'on souhaite remettre en cause les résultats de celle-ci. Le modèle de gestion utilisé par les moteurs de workflow – les transactions ACID – est assez mal adapté aux transactions longue durée. Les critères d'atomicité et d'isolation doivent pouvoir être relâchés afin que les résultats intermédiaires puissent être accessibles avant la fin d'une activité.

Les outils de travail de groupe (*groupware*) fournissent cette fonctionnalité, grâce à la gestion de plusieurs versions et au contrôle de la divergence entre ces versions. Ils fournissent un espace commun partagé pour les données (par exemple CVS, Subversion, les fermes – SourceForge). Ces espaces supportent les accès concurrents de la part des partenaires. La coordination des

différents partenaires est *implicite*, dans le sens où elle n'est pas guidée par un processus défini, mais uniquement par le respect par les partenaires de certaines règles pas nécessairement définies au préalable. Les partenaires peuvent mettre à jour simultanément les mêmes données, et dans ce cas, plusieurs versions existent, et elles peuvent être réconciliées grâce à des paradigmes comme le paradigme «Copy – Modify – Merge» qui permet de fusionner les modifications de chaque partenaire. Les problèmes posés par ces outils sont de plusieurs ordres. Le premier problème concerne le fait que la résolution de certains conflits est complètement à la charge des utilisateurs. Cela peut-être gênant puisqu'il se peut que l'on ne réussisse pas à converger vers une version stable [10]. Le deuxième problème dans le contexte qui nous intéresse concerne le concept d'espace partagé. Certaines entreprises acceptent de mettre à disposition leurs données mais à condition qu'elles gardent un contrôle total sur ces données, c'est-à-dire qu'elles ne souhaitent pas que celles-ci soient disponibles dans un espace partagé sur lequel elles n'ont pas un contrôle total. Enfin, le troisième problème que pose les outils de travail de groupe concerne le fait que les interactions entre les partenaires ne sont guidées que par les données. Or dans le contexte qui nous intéresse, les constructeurs comme les équipementiers souhaitent avoir un processus support, même si ce dernier n'est pas totalement défini.

Ainsi, les principaux besoins pour supporter des procédés coopératifs sont :

- la gestion des interactions complexes, telles que la coordination des processus multi-entrprises de longue durée, les interactions engendrées par les événements synchrones et asynchrones, et la gestion de la distribution et le partage de fragments de procédés, de ressources et de participants ;
- la coordination des flots de données et de contrôle ;
- la gestion des transactions longues (accès aux résultats intermédiaires) et le support de l'adaptation des procédés ;
- la gestion de moyens de coordination tels que la conscience de groupe, la communication synchrone,...

4.4 Contribution

Pour répondre aux problèmes liés au support des processus interentreprises créatifs, nous avons proposé l'approche PointSynchro [3] [Bit03] qui est principalement basée sur deux composants. Le premier composant est un méta-modèle permettant de représenter les entités d'une entreprise virtuelle et les liens entre ces entités. Ce méta-modèle est important puisqu'il permet à chaque partenaire de pouvoir se positionner par rapport à une sémantique commune de l'entreprise virtuelle, de son organisation et de ses composants. Cette vision commune permet de diminuer les problèmes liés aux différences de représentation qui existent entre les différents partenaires. Le second composant est le *point de synchronisation* qui permet de gérer la coordination des processus partagés par les partenaires de l'entreprise virtuelle, et qui offre la gestion des partenaires, de leurs processus, et de leurs ressources.

3. Ces travaux ont été menés dans le cadre du projet EAST et de la thèse de Julia Bitcheva.

4.4.1 Un méta-modèle de processus interentreprises

Le méta-modèle d'entreprise virtuelle, présenté en figure 4.1, se base sur l'hypothèse préalable que pour collaborer, les partenaires définissent et rendent publique la description de leurs compétences (que nous appelons service processus). Le rôle du méta-modèle d'entreprise virtuelle est de représenter les rôles, les activités et les produits publiques d'une entreprise en termes d'artefacts disponibles pour le processus coopératif. Pour cela, nous avons proposé en fait trois méta-modèles : un méta-modèle d'organisation, un méta-modèle de processus, et un méta-modèle de contrat. Ces trois méta-modèles peuvent être vus comme des ontologies qui peuvent être utilisées pour définir les concepts communs aux partenaires et les relations entre ces concepts.

Le méta-modèle organisationel

L'objectif du méta-modèle organisationnel est d'abstraire le modèle organisationnel de chaque partenaire et d'être ainsi indépendant des modèles des organisations des partenaires.

En effet, dans un procédé coopératif, des événements exterieurs à l'entreprise virtuelle peuvent survenir : les partenaires peuvent changer durant le cycle de vie de l'entreprise virtuelle, des employés peuvent changer de rôle ou quitter une organisation, ou un partenaire peut changer son organisation interne (suite à un regroupement ou à une séparation d'entreprise par exemple). Dans ce cas, de nouvelles responsabilités sont attribuées aux employés au sein des entreprises correspondantes, mais cela ne devra en rien changer leurs engagements (contractuels) vis-à-vis de l'entreprise virtuelle ni impliquer des changements dans le fonctionnement de cette dernière (organisation, gestion de processus. . .). De plus, la même ressource peut jouer des rôles différents au sein de différentes entreprises virtuelles. On doit donc séparer l'organisation de l'entreprise virtuelle des organisations internes de chaque partenaire. Cette séparation fait que chaque partenaire doit être capable d'établir une correspondance entre ses structures internes et les rôles qu'il effectue au sein des différentes entreprises virtuelles dans lesquelles il est impliqué. Cette correspondance permet également de ne pas restreindre l'évolution du modèle interne des partenaires, indépendamment de l'appartenance à une entreprise virtuelle.

Le méta-modèle permet également la gestion des *droits d'accès* aux données communes. Comme les participants de l'entreprise virtuelle appartiennent à différentes organisations, nous devons gérer la sécurité d'accès aux ressources. On répond à ce besoin par l'attribution de droits d'accès suivant les rôles. Ainsi les participants reçoivent les permissions liées au rôle qui leur est attribué (*Role-based Access Control*) [FK92].

Le méta-modèle de processus

L'objectif du méta-modèle de processus est d'abstraire les activités de chaque partenaire afin d'assurer l'autonomie et la confidentialité des partenaires.

Ainsi, le concept de service processus permet de représenter une activité issue d'un processus métier d'un partenaire. Ainsi, un service processus est défini par un rôle, un flot de données, un flot de contrôle, et un flot transactionnel. On peut associer aux flots de données et de contrôle des conditions qui permettent de préciser quelles sont les pré-conditions à réaliser avant l'exécution du service processus.

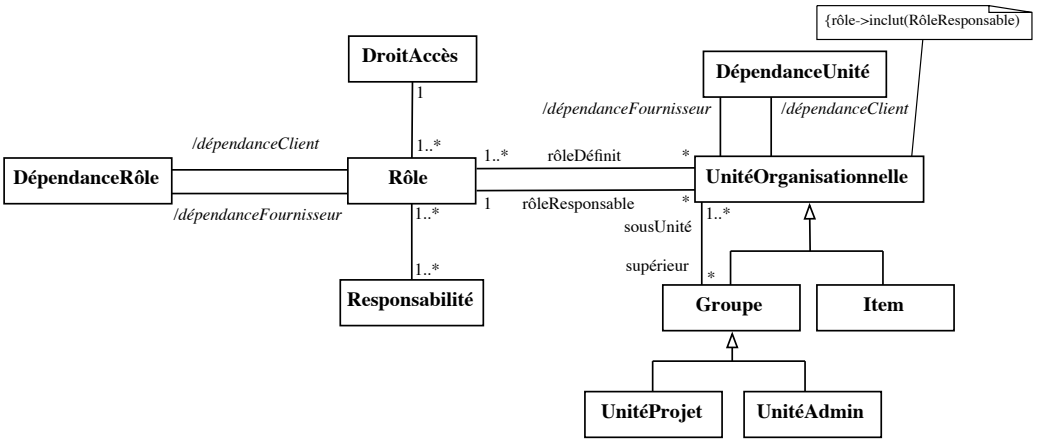


FIGURE 4.2 – Méta-modèle de l'organisation d'entreprise.

La figure 4.3 présente notre méta-modèle de service processus et montre comment un service processus est dérivé d'une activité interne de l'entreprise.

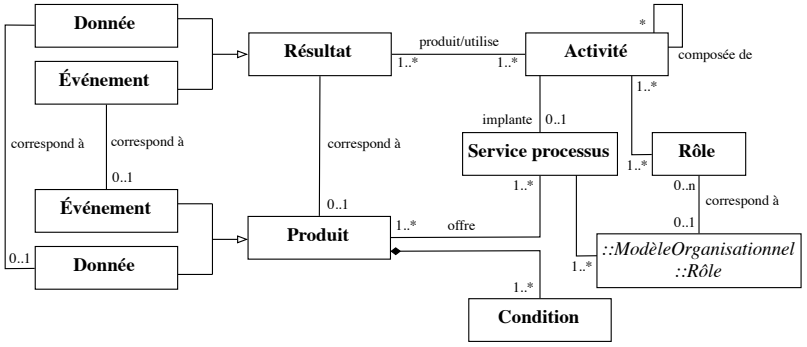


FIGURE 4.3 – Méta-modèle de service processus.

La figure présente le diagramme UML du méta-modèle de processus. Un processus abstrait est un objet qui encapsule toutes les informations nécessaires à son exécution. Ainsi, on définit les règles qui vont permettre de prendre en compte l'activation du processus en tenant compte du flot de contrôle, du flot de données, et du flot transactionnel. Le flot de contrôle définit les règles de transition entre les activités. Chaque service processus possède des activités qui l'ont précédé, et des activités qui vont lui succéder. Lorsqu'un service processus prend fin (une activité chez un partenaire se termine), l'exécution de ses successeurs conformément au flot de contrôle démarre. En plus du flot de contrôle, on distingue également deux autres flots, le flot de données et le flot transactionnel. Le flot de données est classique puisqu'il permet de définir ce dont le service processus a besoin comme données pour s'exécuter, et ce qu'il produit lorsqu'il s'achève. Le flot transactionnel définit quant à lui les règles qui régissent l'exécution transactionnel du service processus, c'est-à-dire son comportement en cas d'échec par exemple. En particulier, on

décrit si le service processus doit être exécuté de façon atomique (isolation) ou s'il peut coopérer avec d'autres applications ou services, s'il est réexécutable plusieurs fois, s'il est compensable ou s'il est pivot (il n'est ni réexécutable, ni compensable – on dit alors qu'il est pivot dans la sémantique des transactions flexibles [MRB+92]).

Pour introduire le lien entre le modèle de processus interentreprises et le modèle de service procédé, nous avons représenté en figure 4.4 l'utilisation des produits provenant de services processus. Le contexte interentreprises se traduit par la consommation de produits de services processus externes à l'entreprise. Par rapport à la figure 4.3 qui illustre comment une organisation publie un service processus, la situation représentée ici illustre la capacité pour un processus d'entreprise d'utiliser les produits issus de services processus externes pour ses propres activités. On distingue donc pour une organisation la capacité de publier un service processus couplé à ses propres activités d'une part, et d'utiliser les produits de services processus provenant d'autres organisations d'autre part (cf. figure 4.4).

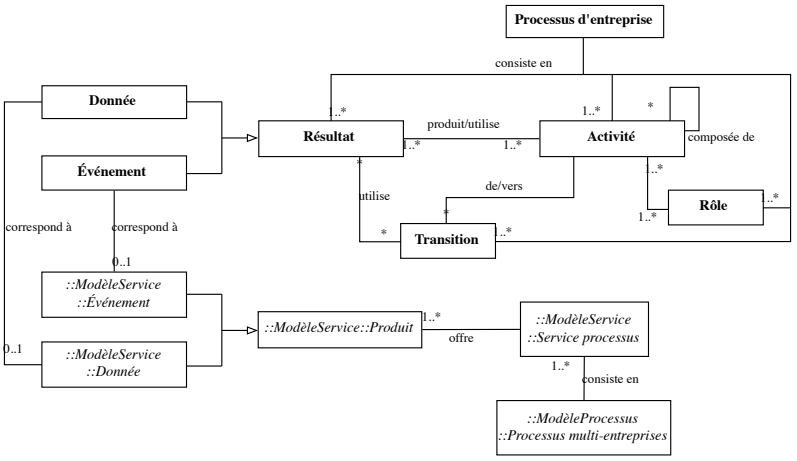


FIGURE 4.4 – Utilisation des produits d'un service procédé au sein d'une entreprise.

Le méta-modèle de contrat

Les contrats sont un élément indispensable dans le domaine de la coopération interentreprises. Ils précisent qui va faire quoi et dans quelles conditions. Le méta-modèle des contrats permet la formalisation des obligations prises par les partenaires qui seront la base pour la spécification des règles de la collaboration. Le contrat contient des références vers les définitions de services processus des entreprises correspondantes et détermine les interactions entre ces services via leurs artefacts. Les clauses définies dans un contrat sont inspirées de la logique déontique et regroupent les obligations, les permissions, et les interdictions.

Un modèle de contrat conforme à ce méta-modèle est présenté plus en détail dans la section 4.4.3.

4.4.2 Modèle de processus interentreprises

Le deuxième composant du modèle PointSynchro permet de supporter la coordination d'un ensemble de services processus qui représentent les processus métiers exportés par les différents partenaires impliqués dans le processus interentreprises. Ce composant, appelé *Point de Synchronisation* (PS) a pour rôle principal coordination des échanges de résultats (données, événements) entre les services processus des différents partenaires. Pour ce faire, l'état et les résultats du service processus d'un partenaire peuvent être envoyés au point de synchronisation correspondant. Ce dernier est alors responsable de la transmission des résultats vers les autres participants. Les résultats d'un service processus peuvent être, selon les cas, des données ou des événements. Ils correspondent aux engagements pris dans un contrat, qui lui-même fait référence aux résultats offerts dans un service processus. Chaque point de synchronisation est sous la responsabilité d'une personne, définie par son rôle au sein de l'entreprise virtuelle. Un point de synchronisation assure la coordination des services processus fournis par les différents partenaires grâce à une liste de contrôle définie entre les partenaires impliqués dans le point de synchronisation, et qui se base sur les clauses des contrats passés entre les entreprises concernées. Les contrôles effectués permettent de vérifier le respect ou pas des engagements contractualisés. Si des déviations sont constatées par rapport aux engagements contractualisés, il est possible de décider d'amender les engagements. Ainsi, une décision peut avoir pour effet la définition d'avenants aux contrats interentreprises : insertion, suppression, modification d'une ou plusieurs clauses de contrat.

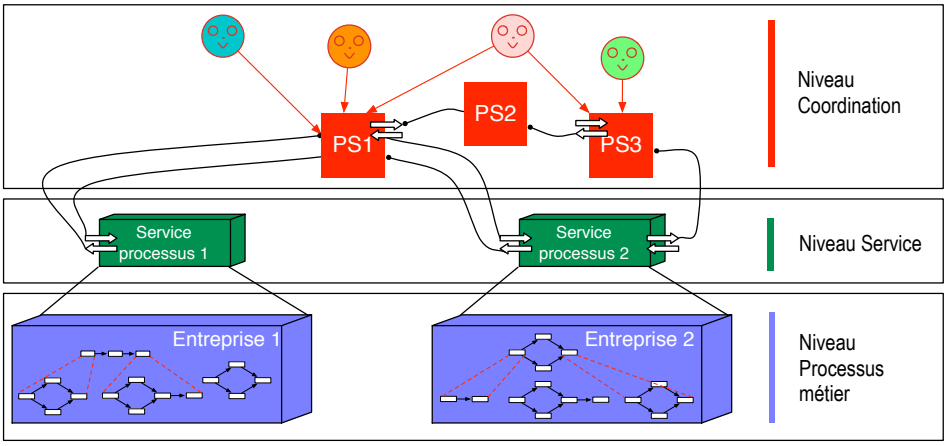


Figure 4.5 – Le modèle PointSynchro.

Un service processus est une vue abstraite d'un processus existant dans une organisation. Il est défini par une localisation, un port et un mapping entre le port et un protocole de transport et un format d'encodage (*binding* en WSDL). Un service processus définit également un mode de coordination qui indique quel est le comportement du procédé par rapport à ses propriétés transactionnelles. Ce mode est soit *isolation*, soit *coopératif*. Si le mode est isolation, un service processus est réexécutable un certain nombre de fois. Si le mode est coopératif, le service processus est soit compensable, soit pivot (au sens du modèle des transactions flexibles [MRSK92]). Cette information nous permet de composer les services processus en veillant à ce que les ex-

ceptions soient gérées conformément aux propriétés transactionnelles du service processus qui pose problème. Un service procédé publie également ses entrées et ses sorties (événements ou données). Il s'agit des produits qui pourront être utilisés par les autres services processus dans le procédé coopératif. Les rôles sont ceux que l'organisation possède en interne mais on peut restreindre la liste à ceux que l'organisation désire voir publier dans le cadre de la coopération. Les rôles possibles sont par exemple : développeur, chef de projet... Enfin, un service processus peut être connecté indépendamment à une application, à un système de gestion de workflow ou à une activité humaine manuelle. La figure 4.5 montre le squelette d'un service processus.

Une activité coopérative

Un point de synchronisation est, puisqu"il nécessite la coopération d'un nombre variable de partenaires, une « activité coopérative » par excellence. Le concept d'activité coopérative n'est pas si courant que cela dans la littérature actuelle. Sans doute est-ce CMI (*Collaborative Management Infrastructure*) qui en a fait l'étude la plus spécifique en introduisant une gestion sophistiquée de rôles, les activités optionnelles et les fenêtres d'opportunité [GSBC00], mais le concept développé reste limité et ne répond pas à la demande des PS.

Nous avons pour notre part choisi de considérer les points de synchronisation comme de véritables mini-projets de coopération, nécessitant de ce fait les fonctionnalités de coopération typiques d'une application coopérative mise en œuvre par une équipe distribuée au sein d'une entreprise virtuelle. Pour cela, nous nous sommes appuyés sur l'expérience du projet ECOO dans ce domaine (les projets ToxicFarm [GMO+04] dans le domaine du *e-engineering*, Coopera [GCP+03] dans le domaine du *e-learning*, Bonita [GCG03] dans le domaine du *e-process*). Les fonctionnalités proposées dans un point de synchronisation intègrent ainsi le *partage d'objets*, la *communication* et la *coordination*.

Le partage d'objets Le partage d'objets est à la base de toute coopération, et c'est donc le service de base à prévoir lorsque l'on parle de coopération dans le contexte des processus multi-entreprises. Il inclut à la fois la gestion de version des objets, mais également la gestion des espaces de travail privés afin de respecter la confidentialité et l'autonomie de chaque partenaire. Il doit être capable de modéliser et de gérer les besoins de chaque partenaire en terme d'accès aux objets, et ces besoins peuvent être relativement complexes à gérer, en tout cas plus complexes que les simples droits de lecture/écriture/modification. En effet, suivant le niveau de coopération qui existe entre les partenaires, il faut être en mesure de définir précisément quels sont les droits des autres partenaires. Ceci influence directement la définition du *point de synchronisation* : par exemple, ce n'est pas parce qu'une organisation A partage un objet avec le participant B et que B partage un autre objet avec le participant C que ce A accepte de partager l'objet avec C. De même, ce n'est pas parce que A partage le même objet avec B et C qu'il souhaite se synchroniser simultanément avec B et C.

La communication Elle est aussi à la base de toute coopération. Les partenaires qui coopèrent ont besoin d'échanger, de se parler, éventuellement de partager des objets, de façon spontanée, sans véritable organisation de la coopération ; c'est ce qu'on appelle la communication. Ainsi, on peut utiliser des outils de communication synchrone (messagerie instantanée [ICQ03], outils de vidéo-conférence [CUs02], de partage d'application [Net02], téléphonie sur IP, éditeurs

temps réels [EG89]) ou des outils de communication asynchrone (courrier électronique, forum, Web site). En fait, il est nécessaire de faire co-exister ces différentes formes de communication, de part la nature des cycles de vie des procédés qui nous intéressent et qui alternent des phases de coopération asynchrones avec des phases de coopérations synchrones. Enfin, un *point de synchronisation* doit également offrir et supporter différents patrons de communication (*request/response*, *sollicit response*, *one-way*, *notification*...). Il est intéressant de constater que Google Wave [4] est une approche qui tente de réunir à la fois les outils de partage d'objets et la communication.

La coordination Comme dans toute application distribuée, la coordination est fondamentale, du fait de la perte d'unité de lieu qui rend les habituelles rencontres formelles (ou informelles) impossibles. Nous pensons que ce problème peut être considéré de deux points de vue complémentaires [GBC+01] :

- la coordination explicite, basée sur l'idée qu'il est possible de définir explicitement un processus, ou plutôt des processus, voire des fragments de processus, et les exécuter pour assurer la coordination des partenaires,
- la coordination implicite, basée sur l'idée que si la bonne information, sur les événements représentatifs qui arrivent dans la coopération, est envoyée au bon moment à la bonne personne, cette information va provoquer de la communication qui aura pour effet une forme d'auto-coordination de l'équipe distribuée constituée par les partenaires du point de synchronisation. C'est ce qu'on appelle plus traditionnellement la « conscience de groupe ».

Bien sûr, ces deux dimensions sont complémentaires, tant il est illusoire dans un processus créatif de tout vouloir formaliser par des workflows, ou de croire que la seule conscience de groupe peut suffire à la coordination d'une équipe distribuée. D'ailleurs, la connaissance de l'état d'avancement des processus respectifs est une bonne source de connaissance pour la conscience de groupe, et la remontée des événements du terrain un bon retour sur l'efficience de la mise en œuvre des processus.

Planifier-Faire-Contrôler-Agir

Nous avons considéré le problème de la coordination des partenaires en intégrant à la fois l'approche de coordination des activités (coordination explicite) et l'approche de conscience de groupe (coordination implicite). La coordination des tâches est basée sur l'hypothèse qu'il est possible de se mettre d'accord sur un processus et de faire respecter ce processus aux partenaires. Pour cela, nous avons proposé un processus inspiré du cycle de Deming (*Plan, Do, Check, Act*) [Dem92], mais nous n'avons retenu que trois activités : *Check*, *Act* et *Plan*.

Un processus planifie (*Planifier*) le moyen d'atteindre son objectif en définissant un plan d'exécution de ses activités (par exemple, il peut s'agir d'un workflow classique au sens de la WfMC [Coa99]) et en établissant des jalons permettant de mesurer l'avancement du processus (typiquement au sens des dates limites de la gestion de projet). Les activités planifiées sont exécutées (*Faire*). Au cours de cette exécution, des contrôles (*Contrôler*) sont effectués pour mesurer le « bon » avancement du projet. Ces contrôles ont pour effet de prévenir des risques de problèmes (non respect des dates limites, difficulté à réaliser une activité, mauvaise qualité, mauvaise organisation...) et de décider des actions à entreprendre pour réduire ces risques

4. `http://wave.google.com`.

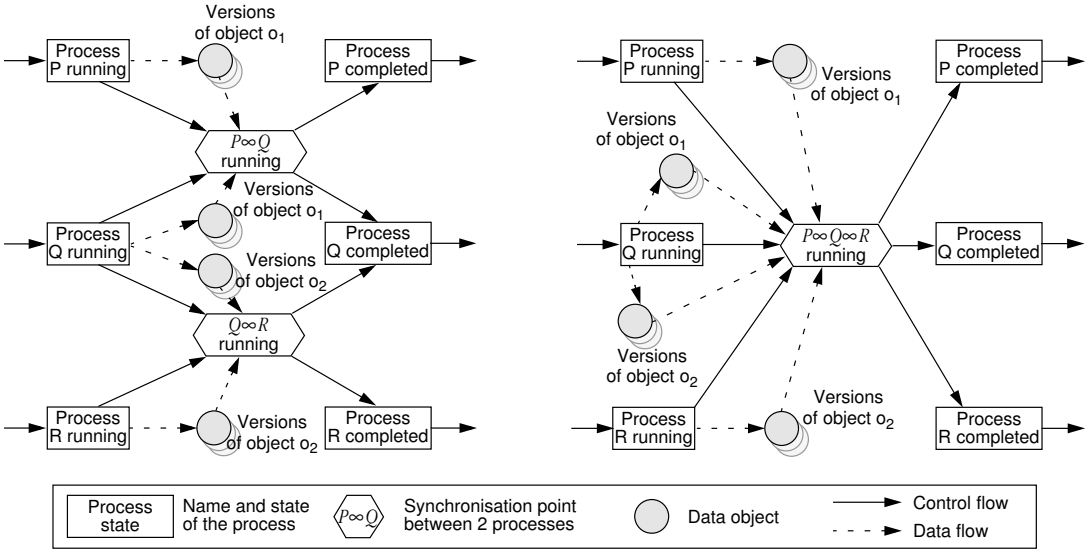


FIGURE 4.6 – Points de synchronisation multiples.

ou pour corriger les problèmes (*Agir*). Une classe importante d'action consiste à pouvoir re-planifier (*Planifier*) une partie des activités en cours. On entre donc ainsi dans un nouveau cycle *Planifier-Faire-Contrôler-Agir*.

Le cycle Contrôler-Décider-Déployer dans le contexte d'une entreprise virtuelle

Du point de vue des processus interentreprises qui nous intéressent, ce cycle mérite quelques commentaires. Le rôle *Contrôler* reste central, et prend même de la valeur dans le cadre qui nous intéresse. Les rôles *Planifier* et *Agir* sont tout aussi fondamentaux, si ce n'est que l'on se contente au niveau du *point de synchronisation* de décider des (re)-planifications et des actions, et de leur déploiement, mais que l'on ne les met pas en œuvre. Enfin, les *points de synchronisation* sont des entités principalement de contrôle et pas de production ; le rôle *Faire* n'est donc pas nécessaire dans le contexte du *point de synchronisation*.

Nous avons ainsi décidé de mettre un œuvre un *point de synchronisation* en suivant un cycle *Contrôle-Décision-Déploiement*. Le contrôle a un sens proche de celui du cycle de départ. *Décision* regroupe les décisions de (re)-planification et d'action. *Déploiement* recouvre tout ce qui prépare la mise en œuvre des décisions de planification et d'action prévues précédemment dans les entreprises partenaires, la mise en œuvre effective leur revenant et étant de leur responsabilité.

4.4.3 Le modèle de contrats

Un contrat est un accord entre deux (ou plusieurs) partenaires qui définit l'ensemble des obligations d'un processus coopératif [CCT03]. La spécification et la définition de e-contracts est actuellement un sujet très important dans le contexte des services web. [AG01] propose une classification des étapes nécessaires à la constitution d'un contrat. Si on se réfère à cette classification, notre travail se situe dans la phase post-contractuelle appelée « respect du contrat » (*contract*

fulfillement) et qui se compose de quatre étapes : exécution du contrat, surveillance du contrat, renégociation et évaluation de l'achèvement du contrat. En particulier, notre modèle de contrat s'intéresse aux objectifs suivants :

- premièrement, le contrat défini nous permet de déployer le graphe des points de synchronisation, au-delà des frontières des organisations qui participent au processus coopératif,
- deuxièmement, le contrat garantit que le processus coopératif entre deux organisations s'exécutera correctement par rapport aux règles du contrat grâce au point de synchronisation.

La première étape de la définition d'un contrat est une phase d'intention. Pour cela, deux organisations définissent un contrat. Les deux organisations vont mettre à disposition l'une de l'autre un certain nombre d'artefacts qu'elles auront préalablement déclarés comme produits d'un service processus. Un service processus fournit un certain nombre d'informations sur une organisation, les rôles et les produits disponibles qui seront utilisés par le processus coopératif. Dans un deuxième temps, les organisations vont définir une activité coopérative comme un graphe de services processus et de points de synchronisation. Elles vont utiliser le contrat comme le moyen de définir un ensemble de clauses qui s'appliqueront aux artefacts qui seront partagés et/ou échangés dans le cadre de l'activité coopérative. Le contrat sera représenté par un point de synchronisation.

La deuxième étape concerne l'application et le contrôle des clauses du contrat par les points de synchronisation. Un point de synchronisation utilise et fournit des produits issus des services processus. Il vérifie que ces produits sont conformes aux besoins du processus coopératif et donc aux clauses du contrat. La figure 4.7 illustre le cycle de vie d'un contrat. Premièrement, un contrat décrit les services processus disponibles (1) pour l'activité coopérative. Il permet la construction du graphe formé par les différents services processus et points de synchronisation (2). Ensuite, le contrat permet aux points de synchronisation de surveiller et de contrôler le déroulement du processus coopératif en s'intéressant aux produits délivrés par les services processus. Une conclusion des contrôles effectués au sein d'un point de synchronisation peut être de replanifier certaines activités et de modifier un contrat en conséquence (3).

Nous allons maintenant détailler comment grâce au modèle de contrat nous déployons les points de synchronisation, et comment ces derniers permettant d'assurer que la coordination du processus coopératif est conforme aux engagements des partenaires.

Le modèle de contrat que nous proposons [PWBG03] repose sur les concepts de « *relations* » et de « *fonctions de satisfaction* » (fonctions *Sat*). Une relation, notée $x\ rel\ y$, est une contrainte entre deux entités (deux processus, ou deux produits, ou un produit et une organisation,...). Une relation est validée par une (ou plusieurs) fonctions appelées les fonctions *Sat*. Ainsi, une relation peut porter sur un document ou une date afin de vérifier par exemple que le document est fourni dans les délais prévus à l'origine de la coopération. Le résultat de la fonction *Sat* associée est un booléen qui précise si une relation d'un contrat entre deux entités est ou n'est pas vérifiée.

D'un point de vue abstrait (cf. figure 4.8), les informations nécessaires à la création d'un contrat se trouve dans quatre composants principaux :

- la partie *Qui* définit les organisations impliquées dans le contrat, et donc dans le point de synchronisation. Une organisation est impliquée dans un contrat grâce aux services processus qu'elle publie et en particulier aux produits de ces services processus. Une organisation est instanciée comme une entité légale, et peut être représentée aussi bien par une

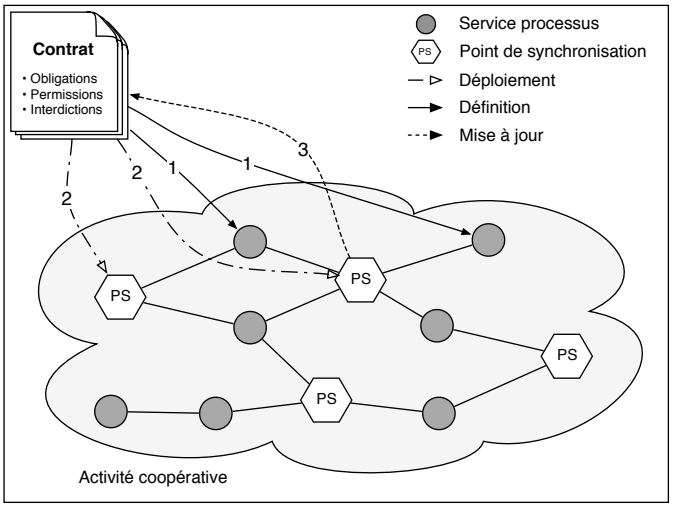


FIGURE 4.7 – Rapport entre contrat et point de synchronisation.

personne ou une organisation proprement dite. Bien sûr, dans le cas où il s'agit d'une organisation, celle-ci devra être représentée par une personne. Les participants d'un contrat et donc d'un point de synchronisation sont donc liés à une ou plusieurs organisations. De même, une organisation agit en fonction d'un rôle donné qui sera défini ultérieurement dans le contrat. Ainsi, une organisation permet de connaître quelle est l'entité légale qui est impliquée et abstrait à la fois les activités et les rôles (modifiables ultérieurement lors de l'exécution du point de synchronisation) qu'aura à assumer cette entité.

– la partie *Quoi* regroupe les informations concernant le sujet du contrat, c'est-à-dire les clauses définies dans ce dernier. Cette partie définit les relations qui devront être satisfaites. Il s'agit donc du cœur du contrat et du point de synchronisation. Les relations décrivent soit des obligations, des permissions ou des interdictions de chacun des partenaires [MM01]. Une relation peut par exemple exprimer une expression de QoS qui a été présentée précédemment (gestion, coût,...). Enfin, la satisfaction des relations devra être effectuée lors de l'exécution du contrat, c'est-à-dire lors de l'exécution des activités de contrôle du point de synchronisation. Ainsi, on va utiliser ces clauses pour définir, déployer et exécuter un point de synchronisation et ses activités.
– la partie *Comment* décrit les étapes nécessaires à l'exécution d'un contrat, c'est-à-dire les activités qui doivent être menées à bien. Cette partie définit précisément quelles sont les relations. Si dans la partie précédente, il s'agissait de définir les clauses du contrat sous forme textuelle, dans cette partie, on définit les relations entre les objets des services processus des organisations partenaires. Il peut s'agir des résultats à délivrer (données ou événements), des dates limites, des clauses qui s'appliquent lorsque l'un des partenaires ne remplit pas ces obligations, des droits d'accès de chacun des partenaires aux objets en fonction des rôles de chacun,... L'ensemble des relations sera utilisé pour générer les activités du point de synchronisation, et en particulier les relations causales, les règles ECA, les transferts de données, les délais, les dates limites, et les conditions de terminaison d'une activité coopérative, c'est-à-dire d'un point de synchronisation.

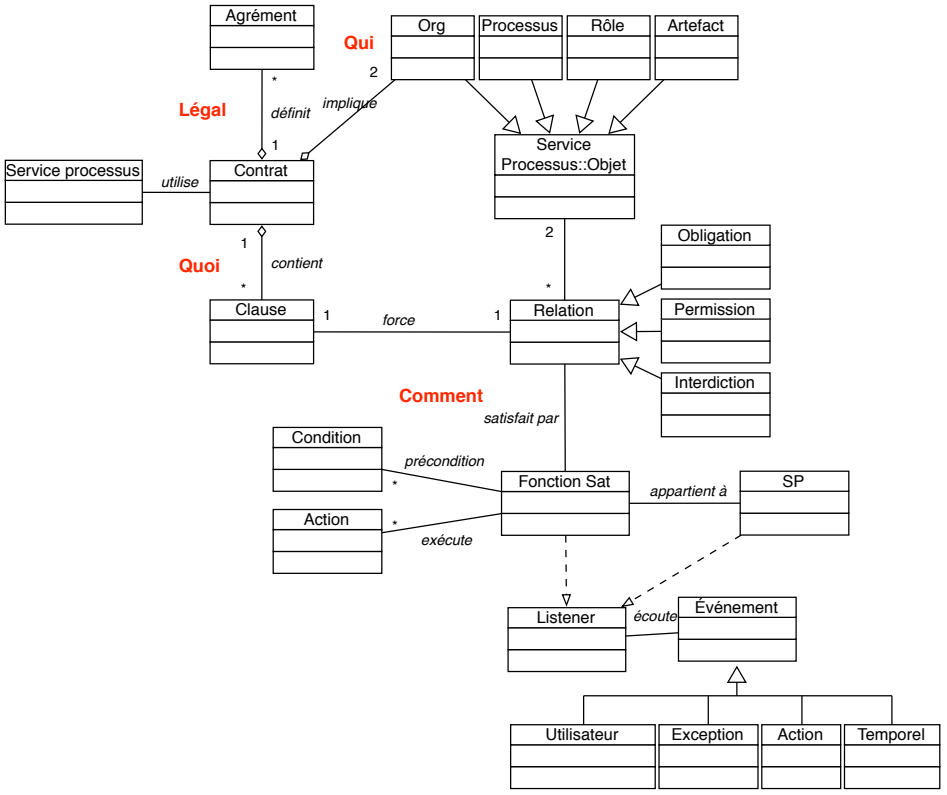


FIGURE 4.8 – Le modèle de contrat.

– enfin, les clauses *Légal* constituent la quatrième partie d'un contrat. Il s'agit de clauses textuelles qui prennent en compte les termes généraux et les conditions générales d'un contrat. Il peut par exemple s'agir de références vers d'autres contrats en cours, de documents de lois, d'attendus, ou de contrats précédents.

Grâce aux services processus, un *point de synchronisation* utilise une vue abstraite des procédés internes des organisations afin de garantir la confidentialité. Il utilise également la notion de relation afin de définir les interactions entre les objets faisant partie d'un processus coopératif, et il gère ces relations en donnant la possibilité de vérifier qu'une relation est (ou n'est pas) satisfaite, de générer une action en fonction du résultat précédent, ou de replanifier le processus coopératif si cela est nécessaire. Pour obtenir un *point de synchronisation* à partir d'un contrat, on transforme les clauses (i.e. les relations et leurs fonctions *Sat*) d'un contrat en règles ECA qui seront utilisées par le *point de synchronisation*. Ainsi, le contrat devient un support pour la définition et l'activation des points de synchronisation (nombre, participants...) et réciproquement, les règles ECA des points de synchronisation garantissent le respect des relations et des fonctions *Sat* des contrats, c'est-à-dire les problèmes liés aux violations de clauses des contrats (obligations, permissions et interdictions).

Événements

Les événements permettent de spécifier quels sont les modifications qui peuvent survenir dans un point de synchronisation. Deux catégories d'événements sont gérées. La première concerne les données. Dès qu'une nouvelle version d'un objet est déposée dans l'espace de travail du point de synchronisation, les partenaires qui sont concernés par cet objet sont notifiés. De même, si un document n'a pas été délivré à une date donnée, c'est un événement. Un autre exemple concerne le fait qu'un document ait été délivré correctement. La deuxième catégorie d'événements concerne l'état des service processus. Lorsqu'un MEP (Message Exchange Pattern) est exécuté, plusieurs événements sont produits. Le point de synchronisation enregistre ces événements et exécute en réponse un certain nombre d'actions. Dans un point de synchronisation, les relations et les fonctions *Sat* sont transformées en règles ECA. Les événements sont capturés par le point de synchronisation grâce à l'existence de *listeners* auxquels le point de synchronisation s'abonne (cf. figure 4.8).

Condition

Les conditions des règles ECA permettent d'affiner la sélection des actions à effectuer en fonction du contexte. Par exemple, on pourra vérifier si un partenaire désire accéder à un objet de l'espace de travail qu'il possède bien les droits suffisants pour pouvoir faire un *check-out* sur cet objet.

Action

Les résultats du contrôle déterminent la suite du processus de coopération. Si les produits des activités sont conformes aux critères prédéfinis, le point de synchronisation continue à suivre le déroulement du processus (fonction contrôler). Dans le cas contraire, nous entrons dans la phase d'élaboration des changements nécessaires pour l'obtention des objectifs, fonction que nous avons appelée « décider ».

Il n'y a pas dans un point de synchronisation de décision automatique. Les différents types d'actions sont :

- accepter la différence entre le plan et la réalité (par exemple accepter une version d'un document),
- ajouter une activité (par exemple pour étudier une nouvelle alternative),
- abandonner une activité (par exemple abandon d'une alternative),
- supprimer une activité (par exemple une activité d'un intérêt relatif, pour rester dans les délais ou l'enveloppe des coûts),
- décaler une activité en avant (par exemple parce qu'il s'avère que cette activité peut aider à lever des ambiguïtés),
- décaler une activité en arrière (par exemple parce qu'il s'avère que cette activité n'a pas un impact fort sur le processus),
- paralléliser des activités prévues en séquence (par exemple pour gagner du temps),
- mettre en en séquence des activités préalablement en parallèle (par exemple du fait de l'absence d'une ressource),
- itérer sur une activité (parce que l'objectif n'est pas atteint mais semble atteignable sans changer le processus),

– ajouter une synchronisation (par exemple pour renforcer la relation entre deux entreprises),
– supprimer une synchronisation entre deux activités (par exemple pour alléger un processus).

On peut également modifier les règles de contrôle, à savoir ajouter des règles, modifier des règles, ou supprimer des règles.

4.4.4 Modèle de confidentialité

La relation entre contrat et point de synchronisation définit un véritable modèle de confidentialité. Certaines clauses de contrat définissent des règles de visibilité qui restreignent la vue et les droits qu'une entreprise possède sur un objet (via les règles ECA). Ensuite ces clauses sont utilisées pour définir la liste des points de synchronisation, leurs participants et indirectement le processus interentreprises. Réciproquement, la mise en œuvre de ce processus assure le respect des règles de confidentialité qui peuvent éventuellement évoluer comme une décision prise au sein d'un point de synchronisation.

4.4.5 Be2gether

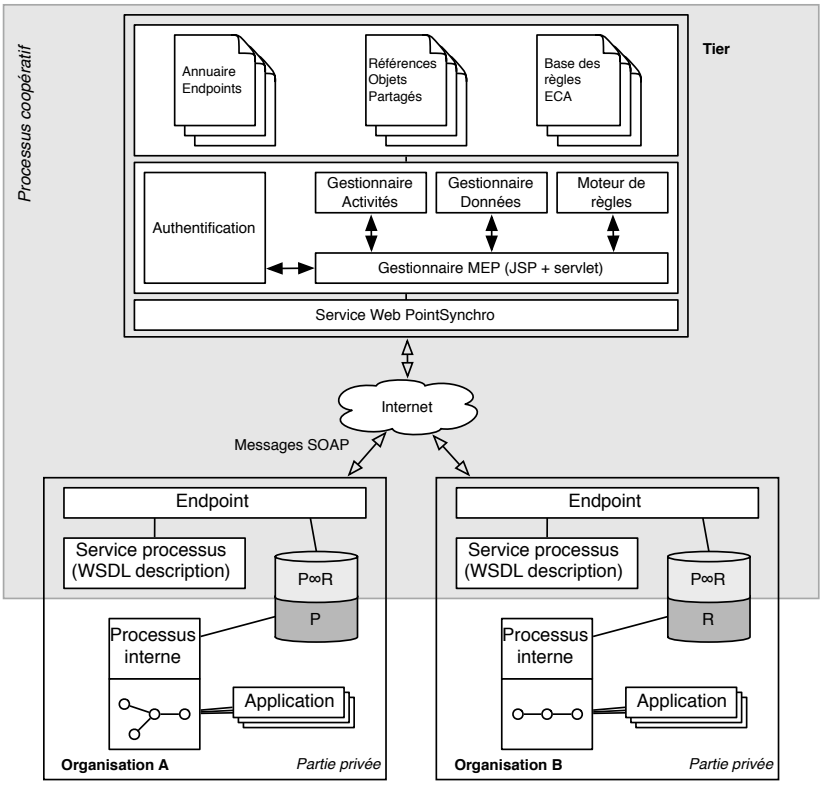


FIGURE 4.9 – Entreprise virtuelle avec PointSynchro.

Be2gether est l'implantation de POINTSYNCHRO. Be2gether est basé sur une architecture orientée service qui utilise les standards des services Web. Chaque partenaire héberge un four-

nisseur de services, et les échanges entre chaque partenaire et le point de synchronisation se font grâce à des messages SOAP. La description des services processus de chaque partenaire sont définies en WSDL, avec certaines propriétés rajoutées au standard (aspects transactionnels par exemple).

Un point de synchronisation est géré par un tiers. Ce tiers héberge les *endpoints* des services processus des partenaires, les contrats (sous la forme des règles ECA), un espace commun pour les données. Un contrat est en fait un document XML qui sera transformé en un ensemble de règles ECA. Un point de synchronisation est un service Web qui gère également un ensemble de MEP (*Message Exchange Pattern*). Nous avons défini plusieurs MEPs : *deliver* qui est l'équivalent d'un *check-in*, *download* qui est l'équivalent d'un *check-out*, *reconcile* qui est l'équivalent d'un *diff+merge*, *Request/Response* pour toutes les informations concernant l'état d'avancement des services processus, et *Out* pour les notifications venant des services processus.

Pour chaque événement (données ou contrôle) qui arrive au niveau du point de synchronisation, un moteur de règles (*Jess Rule Engine*) vérifie si une (ou plusieurs) règle(s) peuvent être déclenchées. Par exemple, lorsqu'un objet est déposé dans l'espace de travail du point de synchronisation, on vérifie que la date de dépôt est cohérente avec la règle qui gère cet objet. De même, lorsqu'un partenaire essaie de lire un objet, on vérifie que les droits de lecture sont cohérents. Lorsqu'un service processus envoie son état courant, on vérifie si une règle peut être déclenchée (retard par exemple). Enfin, si un service processus n'a pas envoyé un document conformément au contrat, on demande une notification permettant de savoir quel est son état actuel.

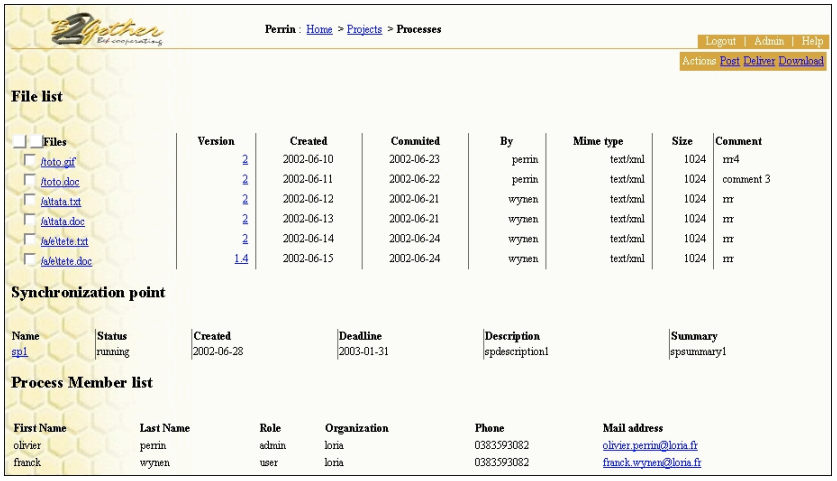


FIGURE 4.10 – Interface.

Un objectif clairement énoncé était la nécessité pour les logiciels développés de s'intégrer, d'une part au Web et à sa technologie, d'autre part aux outils traditionnels utilisés par les partenaires d'une coopération.

Cet objectif est atteint par la conjonction des propriétés du modèle de procédé interentreprises, du modèle de coopération, et par une implantation basée sur des protocoles de programmation standardisés sur le Web.

Le modèle de procédé interentreprises ne nécessite pas de prise de contrôle sur les gestion-

naires de processus locaux et les interactions se font par l'intermédiaire des objets partagés. L'environnement de coopération repose lui aussi beaucoup sur les relations de partage d'objets, mais ne fait pas d'hypothèse sur le type et la nature des objets partagés.

Ensuite, tous les services, que ce soit ceux généraux de coopération, ou ceux plus spécifiques de coordination, ont été développé avec une technologie Web XML [5], UDDI [6], SOAP [7] et WSDL [8] qui assure leur intégration et leur accès depuis n'importe quelle plate-forme Web.

Enfin, PointSynchro peut être vu comme un service d'agrégation de services Web. Dans la terminologie de [LK03], il s'agit plutôt d'un service d'agrément qui permet la définition de protocoles permettant la coordination de services Web (pour nous des services processus) dans le cadre particulier d'une application et pour une durée de vie limitée. La mise en œuvre du procédé Contrôler/Décider/Déployer peut être considérée comme une agrégation de type orchestration.

4.5 Synthèse

L'opportunité qui m'a été offerte de m'intéresser aux processus m'a permis de tenir compte, en plus des données, des activités qui manipulent ces données. Dans le contexte des processus interentreprises, cette dualité entre activités et données est très importante, et amène à raisonner non seulement en termes d'abstraction au niveau des activités, mais également au niveau des données. Les formalismes utilisés pour la représentation des processus classiques sont alors insuffisants pour modéliser correctement ce nouveau types de processus. De même, les formalismes qui permettent de raisonner au niveau données sont insuffisants pour raisonner sur les activités. L'objectif des travaux que j'ai menés était donc de trouver un équilibre entre la dimension activités et la dimension données. Le fait d'avoir eu comme support des exemples concrets de processus interentreprises a été une grande chance, car les problèmes sont plus faciles à cerner. En particulier, il est nécessaire d'avoir, comme dans le cas des données, une compréhension commune des artefacts manipulés (grâce à des méta-modèles, ou des ontologies). Les méta-modèles de PointSynchro répondent à ce besoin. De même, l'aspect coordination, l'intimité (*privacy*) et l'autonomie sont des besoins critiques pour les partenaires. Le point de synchronisation vu comme une composition de services processus est une proposition pour prendre en compte ces aspects. La flexibilité, l'absence de modèle très structuré, et le support d'activités non nécessairement prévues sont des caractéristiques qui n'existent pas dans les processus classiques et qui doivent être proposées dans le cadre des processus interentreprises. Le cycle *Contrôler-Décider-Déployer* répond à ces besoins. Enfin, le modèle de contrat permet de vérifier que le processus (en prenant en compte à la fois les activités et les données) se déroule conformément aux limites fixées par les différentes partenaires.

Dans le futur, il est clair que le paradigme actuel pour modéliser les processus métier orienté uniquement vers le processus, c'est-à-dire les flux de contrôle des activités, a atteint une limite. Les données sont soit complètement abstraites (via les activités), soit ajoutées après coup grâce aux annotations. Il existe donc un challenge qui permette de placer les données au même niveau que les activités. C'est ce que l'on appelle les modèles de processus orientés données (Siena (IBM), WebML(Milan), WAVE (UCSD), Kepler(UCDavis), Hilda(Cornell)). Du coup, toutes les

5. `http://www.w3.org/XML/`
6. `http://uddi.xml.org/`
7. `http://www.w3.org/TR/soap/`
8. `http://www.w3.org/TR/wsdl`

propositions permettant de vérifier un processus doivent être repensées en termes de vérification des flux de données et non plus uniquement des flux de contrôle. L'idée consiste alors à avoir un formalisme de suffisamment haut niveau ayant un pouvoir d'abstraction suffisant pour passer d'une spécification à son exécution [Hul08, DHPV09].

Articles

A model to support collaborative work in virtual enterprises
Olivier Perrin, Claude Godart.
Data Knowledge Engineering Journal. 50(1) : 63-86 (2004).

The ToxicFarm Integrated Cooperation Framework for Virtual Teams
Claude Godart, Pascal Molli, Gérald Oster, Olivier Perrin, Hala Skaf-Molli, Pradeep Ray, Fethi Rabhi.
Distributed and Parallel Databases, Volume 15(1), 2004.

A Model to Support Collaborative Work in Virtual Enterprises
Olivier Perrin, Franck Wynen, Julia Bitcheva, Claude Godart.
Proceedings of the International Conference, BPM 2003, LNCS 2678.

Coordination of Cooperative Processes with the Synchronization Point Concept
Julia Bitcheva, Olivier Perrin, Claude Godart. Proceedings of the International Conference on Web Services, ICWS'03. 2003

A Contract Model to Deploy and Control Cooperative Processes
Olivier Perrin, Claude Godart.
Proceedings of the 4th International Workshop, TES 2003, LNCS 2819

Cooperative Workflows to Coordinate Asynchronous Cooperative Applications in a Simple Way
Claude Godart, François Charoy, Olivier Perrin, Hala Skaf-Molli.
Proceedings of the Seventh International Conference on Parallel and Distributed Systems (ICPADS'00). IEEE Computer Society, 2000.

Coo : A Workflow Operator to Improve Cooperation Modeling in Virtual Processes
Claude Godart, Olivier Perrin, Hala Skaf.
Proceedings of the Ninth International Workshop on Research Issues on Data Engineering (RIDE 99) : Information Technology for Virtual Enterprises. IEEE Computer Society, 1999.

Chapitre 5

L'intégration par les services

5.1 Introduction

Dans le chapitre précédent, nous avons rapidement décrit comment les systèmes de gestion de processus permettent de gérer et contrôler un ensemble d'activités, et le flot de contrôle qui gère l'enchaînement de ces dernières. Nous avons également présenté la problématique des processus utilisés dans le contexte des entreprises virtuelles. Les systèmes de gestion de processus sont encore majoritairement utilisés dans le contexte d'une seule et même entreprise. Il y a plusieurs raisons à cela. Premièrement, les activités sont codées dans une logique métier propre à l'entreprise, avec tout le savoir-faire de cette entreprise. Ensuite, les formats utilisés par les systèmes sont propriétaires, et donc il est difficile de faire communiquer entre eux différents systèmes malgré les efforts de normalisation de la WfMC. Enfin, les processus gérés sont très souvent destinés à l'entreprise elle-même, sans lien avec d'autres entreprises ou avec l'extérieur, et donc sans aucune attention particulière pour palier les problèmes liés à l'échange et au partage.

Grâce à l'expérience acquise avec Be2gether, nous avons commencé à travailler sur les architectures orientées services. En effet, Be2gether était un prototype basé sur une architecture à base de services. Les architectures orientées service (*Service-oriented architecture* – SOA) sont basées sur le fait que les fonctions d'une application sont vues comme des services qui peuvent être offerts sur Internet (ou sur un intranet). Ces services sont, de manière intrinsèque, distribués, hétérogènes, dynamiques, et surtout faiblement couplés. Dans ce contexte, les limites des systèmes actuels ne sont plus celles de l'entreprise, et d'un point de vue architectural, la coopération entre plusieurs organisations est plus facilement envisageable et réalisable. C'est par exemple le cas pour un processus qui franchirait les frontières d'une entreprise pour être distribué dans plusieurs organisations. Le concept d'architecture orientée service ne doit pas être vu comme une simple technologie mais plutôt comme une philosophie de conception : exposer comme des services les fonctionnalités métiers d'une application, et utiliser ces services dès lors que la fonctionnalité (ou l'information) est requise. En raison du couplage faible des services, l'impact le plus original de ces architectures est de permettre de prendre en compte les difficultés liées au changement. En effet, le besoin de flexibilité (on parle également d'agilité) est une demande importante des nouveaux processus métiers, qu'ils soient ou non interorganisationnels, et c'est l'une des forces de ces architectures que de pouvoir répondre à ce besoin puisque les services sont faiblement couplés, et que l'on peut potentiellement accéder à plusieurs services

offrant les mêmes fonctionnalités, ce qui permet à un processus d'être très flexible. L'instance la plus connue des architectures orientées service est basée sur les services *web*.

Les architectures orientées service ne sont pas complètement nouvelles, puisque CORBA (*Common Object Request Broker Architecture*) ou DCOM (*Distributed Component Object Model*) fournissaient déjà des fonctionnalités proches. Les architectures orientées services possèdent cependant un certain nombre d'avantages par rapport aux approches traditionnelles d'architectures distribuées. En particulier, elles offrent des services indépendants des plates-formes, indépendants de la localisation, indépendants des systèmes et des réseaux. Mais l'avantage principal concerne le couplage réellement faible entre les services, ce qui n'est pas tout à fait le cas en CORBA ou DCOM en particulier.

Les services Web diffèrent donc des composants de processus métiers traditionnels car ils sont généralement fournis par des organisations différentes, indépendamment de tout contexte d'exécution. Puisque chaque organisation possède ses propres règles de travail, les services Web doivent être traités comme étant strictement autonomes. C'est d'ailleurs ce qui à notre avis les différencie des services composants dont la maîtrise est possible par le client du composant.

Cette autonomie et la potentielle hétérogénéité qui en découle rendent difficile la connaissance du comportement global d'un service composé. C'est ce problème de composition auquel nous nous sommes intéressés et plus particulièrement le problème qui consiste à garantir des compositions fiables de services Web, c'est-à-dire des compositions dont toutes les exécutions sont correctes du point de vue «métier». Une exécution sera définie comme correcte si elle est capable de réagir au(x) éventuel(s) échec(s) de certain(s) service(s) composant(s) conformément aux besoins et aux prévisions des concepteurs du service composé.

Les technologies actuelles des services Web sont incapables de résoudre ce problème de manière efficace. Pour dépasser ces limites, nous avons proposé un modèle de services (composés) transactionnels qui combine la puissance d'expression des modèles de workflow et la fiabilité des modèles transactionnels. Pour modéliser les services composés en intégrant la dimension complexe qu'est la gestion des échecs à l'exécution, nous avons introduit le concept de patron transactionnel, paradigme qui étend les patrons de workflow de [vdAtHKB03] avec des dépendances transactionnelles.

5.2 Travaux relatifs

La problématique de composition fiable de services Web touche à deux domaines de recherche : les workflows d'une part, et en particulier l'avantage que fournit le pouvoir d'expression des modèles des workflow, et les modèles transactionnels d'autre part, et en particulier la fiabilité de ces modèles. Dans ce qui suit, nous présentons rapidement ces deux approches, les technologies des services Web autour des aspects transactionnels et certains travaux connexes.

5.2.1 Modèle Transactionnels Avancés

Les MTA [ELLR90, Elm92, GMS87, GMGK$^+$91] étendent le modèle de transaction ACID pour répondre aux besoins de certaines applications que ce modéle ne peut pas supporter. Ces modèles partagent certaines caractéristiques : une structure plus complexe qu'une simple séquence d'opérations, un degré de concurrence et de parallélisme intra transaction plus élevé,

le relâchement des propriétés d'atomicité et d'isolation, et l'adoption de nouvelles sémantiques transactionnelles comme la compensation et l'alternative.

Malgré le degré de flexibilité qu'ils apportent, ces modèles restent limités pour supporter des applications orientées processus [WS97] . En effet, leurs structures de contrôle restent primitives par rapport à celles des services Web composés qui nécessitent des constructeurs de branchement et de synchronisation plus évolués. En plus, ils imposent des contraintes que les concepteurs doivent respecter lors de la spécification de leurs applications. De plus, en adoptant un modèle donné, les concepteurs doivent adhérer au critère de correction implicitement défini par l'ensemble des règles prédéfinies.

5.2.2 Workflows transactionnels

Les systèmes de workflow [JB96] facilitent la description, la modélisation, l'analyse et l'exécution des processus métiers. Ces systèmes ont toujours été sujets à de nombreuses critiques à cause de l'absence de mécanismes de correction et de fiabilité [AAAM97]. Le terme de workflow transactionnel [SR93] a été introduit pour reconnaître la pertinence des transactions dans le contexte des workflows pour assurer un certain degré de correction et de fiabilité. Les workflows transactionnels concernent l'exécution coordonnée de plusieurs tâches et suggèrent l'utilisation sélective de propriétés transactionnelles pour des tâches individuelles ou pour des workflows tout entiers.

Le modèle des contrats [WR92] permet d'intégrer des sémantiques transactionnelles (forcer l'atomicité de certaines étapes, définir des mécanismes de recouvrement en avant, etc.) à la logique d'exécution d'une application. Malheureusement, l'absence de critère de correction et des mécanismes de vérification ne permettent pas d'assurer la fiabilité comme pour les MTA.

Le système METEOR [KS95] permet de spécifier le flot de contrôle, les mécanismes de recouvrement et l'ensemble des états de terminaison acceptés (ETA) comme un critère de correction. Cependant, son incapacité de vérifier ce critère de correction à priori limite son application aux applications centrée bases de données.

5.2.3 Technologies des services Web

Concernant les technologies actuelles des services Web, on peut distinguer là aussi deux types de contributions : celles orientées plutôt workflow, comme WSBPEL [OAS07] et WS-CDL [KBRL04], et celles orientées plutôt modèles transactionnels comme WS-AtomicTransaction [NRLW06], WS-BusinessActivity [FGHL06] et WS-TXM [AFI$^+$03].

WSBPEL est un langage de définition de processus métiers permettant d'orchestrer des services Web. WS-CDL est un langage de description de chorégraphie. Comme les modèles de workflow, ces deux langages répondent aux besoins des processus métiers en terme de structure de contrôle, mais ne supportent pas de mécanismes efficaces pour assurer la fiabilité en particulier selon les besoins spécifiques des concepteurs. Notre contribution peut compléter ces deux technologies et être utilisée au dessus d'elles en implantant nos patrons transactionnels en WSPBEL ou WSCDL. Mais ceci nécessite d'étendre ces deux langages pour prendre en compte les dépendances d'annulation et d'alternatives.

WS-Transaction et WS-TXM définissent un ensemble de protocoles de coordination transactionnels basés respectivement sur WS-Coordination et WS-CF. WS-Transaction et WS-TXM distinguent plusieurs types de protocole. Les protocoles définis sont des variantes des modèles

transactionnels (avancés) existants. CLF/Mekano [AAP+99] est une infrastructure permettant de coordonner un ensemble de composants distribués notamment en tant qu'une transaction avancée.

Ces approches héritent de certaines limites des MTA, surtout au niveau de la rigidité des structures de contrôle. Notre approche permet d'étendre ces protocoles pour supporter des structures complexes tout en préservant la fiabilité. En effet, une composition de patrons transactionnels peut être considérée comme un protocole de coordination transactionnel.

5.2.4 Autres travaux connexes

RosettaNet [1] et ebXML [2] sont deux propositions basées sur XML pour mener des interactions B2B. Elles définissent des concepts communs (au niveau transport, données et processus) à partager et à respecter par les partenaires. Par exemple, une entreprise qui veut publier un service doit respecter les modèles des processus prédéfinis (≪business process≫ pour ebXML et PIP pour RosettaNet). L'inconvénient majeur de ces deux approches est que les partenaires sont contraints de respecter et se conformer aux spécifications communes prédéfinies.

[HA00] définit la notion de sphère comme un moyen de spécifier, parmi d'autres, l'atomicité d'un processus. En utilisant des propriétés transactionnelles similaires aux nôtres, il définit un ensemble de règles à respecter pour construire ce qu'on appelle une sphère bien formée. Une sphère bien formée est une extension d'une transaction flexible [ZNBB94] pour permettre des exécutions en parallèle. Grâce à une meilleure spécification du parallélisme et de la synchronisation via l'utilisation des patrons de workflow, notre approche définit des règles moins contraignantes que celle définies par [HA00] ce qui la rend plus de flexible pour définir des compositions fiables.

[LZ04] présente un modèle qui intègre la notion de compensation au protocole ≪Tentative Hold≫. [SD05] présente une méthodologie de modélisation des services composés qui distingue entre différents perspectives notamment structurelle et transactionnelle. [Pap03] présente une assez large spécification des transactions commerciales et une revue des activités de standardisation et de recherche en relation. Il identifie les principaux composants d'une plate-forme pour mener des transactions commerciales. Selon cette classification, notre approche permet de définir des modèles transactionnels pour mener des applications métiers de longue durée.

5.3 Contribution

5.3.1 Modèle de services Web Transactionnels

Pour présenter notre contribution [3], nous considérons une application Web d'organisation de voyage dont la logique d'exécution est illustrée par la figure 5.1. Le client spécifie ses besoins en terme de destination et de choix d'hôtel via la tâche ≪Spécification des Besoins du Client≫ (SBC). L'application lance ensuite deux tâches en parallèle ≪Réservation du Vol≫ (RV) et ≪Réservation d'Hôtel≫ (RH) pour réserver un vol et un hôtel selon les choix du client. Une

1. `http://www.rosettanet.org`
2. `http://www.ebxml.org`
3. Ces travaux ont été menés dans le cadre de la thèse de Sami Bhiri.

fois les réservations faites, la tâche «Paiement en Ligne» (PL) permet au client de payer ses réservations. Enfin, et selon l'évaluation de trois conditions exclusives, les documents de voyage (billet d'avion et réservation d'hôtel) sont envoyés au client via l'une des tâches «Envoi de Documents par Fedex» (EDF), «Envoi de Documents par DHL» (EDD) ou «Envoi de Document par TNT» (EDT).

On peut au passage facilement constater que la problématique de la composition de services Web possède un lien fort avec celle de la gestion de processus métiers. Mais la composition de services Web est rendue plus complexe à cause de l'autonomie et de l'hétérogénéité de ces derniers, ainsi que de la nature dynamique des compositions.

Le problème auquel nous nous sommes intéressés est le suivant : comment garantir la fiabilité de la composition de services représentée dans la figure 5.1 ? Par composition fiable, nous entendons toute composition dont toutes les exécutions sont correctes, dans le sens où elles répondent aux attentes des concepteurs, y compris, et surtout en cas d'échec d'un des services.

Pour gérer les échecs des services composants, les concepteurs spécifient en plus du flot de contrôle des mécanismes de recouvrement permettant au service composé de recouvrer un état cohérent en cas d'échec d'un service composant. Cela repose en général sur la spécification de propriétés spécifiques des services et de réactions en cas d'échec d'un service. Par exemple, afin de garantir la fiabilité du service composé de la figure 5.1, les concepteurs peuvent spécifier que : (i) les services RV, PL et EDT sont sûrs de se terminer avec succès, (ii) en cas d'échec du service RH, il faut annuler ou compenser le service RV (selon son état courant) et (iii) en cas d'échec du service EDF, il faut activer le service EDD comme une alternative. Si ces règles répondent aux besoins d'une application de réservation particulière, une autre application peut faire d'autres choix en combinant des services qui ont des propriétés différentes et en définissant d'autres règles. Le problème auquel nous nous intéressons consiste donc à assurer que les mécanismes de recouvrement spécifiés garantissent des exécution correctes, notamment selon les besoins spécifiques des concepteurs.

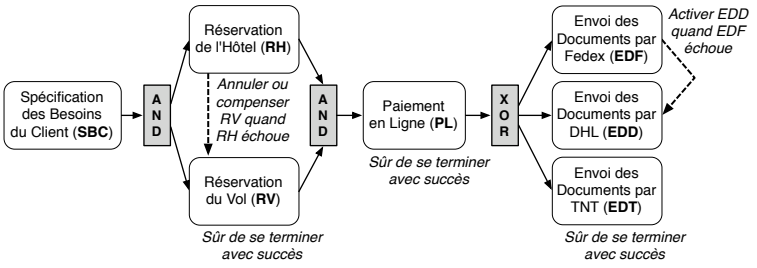


FIGURE 5.1 – Service Web composé.

Service Web transactionnel

Un service Web est un programme modulaire auto descriptif qui peut être publié, découvert et invoqué via le Web. Un service Web transactionnel est un service Web dont le comportement prend en charge des propriétés transactionnelles.

Un service Web transactionnel est un service Web agrémenté de propriétés transactionnelles. Les propriétés transactionnelles que nous considérons sont *rejouable*, *compensable* et *pi-*

vot [MRKS92]. Un service est dit *rejouable* s'il se termine toujours avec succès après un nombre fini d'activation. Il est dit *compensable* s'il offre des mécanismes de compensation pour annuler sémantiquement son travail. Enfin, il est dit *pivot* si une fois terminé avec succès, ses effets ne peuvent pas être compensés. L'ensemble des combinaisons de propriétés possibles pour un service est {$\emptyset$ *; rejouable ; compensable ; pivot ; (rejouable,compensable) ;(rejouable,pivot)*}. Le choix de ces propriétés s'explique par le fait qu'elles permettent de caractériser deux comportements pertinents liés au problème de la fiabilité, à savoir l'éventualité d'échec d'un service et sa capacité à compenser son travail.

À chaque service transactionnel est également associé un diagramme à transition d'états qui modélise son comportement interne (d'un point de vue transactionnel). Ce diagramme décrit les états possibles par lesquels le service peut passer durant son cycle de vie transactionnel et les transitions possibles entre ses états.

Nous avons adapté et étendu le diagramme à transition d'état défini par le WfMC [WFM95] afin de caractériser plus précisément la terminaison d'un service (notamment en cas d'échec) et sa capacité de compenser son travail. Ainsi, nous ne considérons pas l'état *suspendu* (que nous intégrons avec l'état *activé*), nous ajoutons un état *compensé* pour spécifier que le service a été compensé et nous ajoutons une transition *rejouer()* pour caractériser que le service est rejouable. Ainsi, l'ensemble des états que nous considérons est {**initial, abandonné, activé, annulé, échoué, terminé, compensé**}. L'ensemble des transitions que nous considérons est {**abandonner(), activer(), annuler(), échouer(), terminer(), compenser(), rejouer()**}.

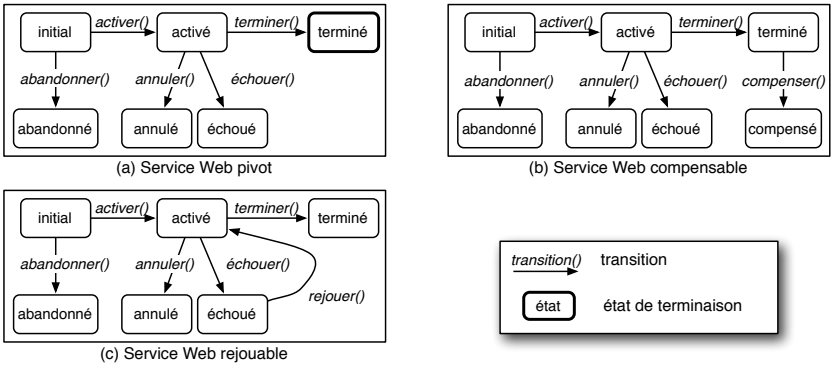


FIGURE 5.2 – États et transitions d'un service transactionnel.

Au départ, quand un service est instancié, il est dans l'état **initial**. La transition **abandonner()** permet d'abandonner l'exécution du service (avant d'être activé) et le fait passer ainsi de l'état **initial** à l'état **abandonné**. La transition **activer()** permet d'activer un service (il passe ainsi de l'état **initial** à l'état **activé**). La transition **annuler()** permet d'annuler un service (en cours d'exécution) et le fait ainsi passer de l'état **activé** à l'état **annulé**. La transition **échouer()** permet de marquer l'échec d'un service (il passe de l'état **activé** à l'état **échoué**). La transition **terminer()** permet de marquer la terminaison d'un service avec succès (il passe de l'état **activé** à l'état **terminé**). La transition **compenser()** permet de compenser sémantiquement le travail effectué par le service (il passe de l'état **terminé** à l'état **compensé**). Enfin, la transition **rejouer()** permet d'activer un service après son échec (il passe de l'état **échoué** à l'état **activé**). L'ensemble des états et des transitions dépendent des propriétés transactionnelles du

service (voir figure 5.2). Par exemple, un service ne possède un état **compensé** et une transition **compenser()** que s'il est compensable.

Nous faisons la distinction entre les transitions externes et les transitions internes d'un service Web transactionnel. Une transition externe est activée par une entité externe, permettant à un service d'interagir avec l'extérieur. Les transitions externes que nous considérons dans notre approche sont : abandonner(), activer(), annuler() et compenser(). Une transition interne est activée par le service lui-même (l'agent responsable de l'exécuter). Les transitions internes que nous considérons dans notre approche sont échouer(), terminer() et rejouer().

Mise en œuvre Grâce à la description sémantique des services Web, il est possible de spécifier leurs propriétés transactionnelles. WSMO [RLK06] définit un ensemble de propriétés non fonctionnelles d'un service sémantique. Cette initiative définit la propriété non fonctionnelle `transactional`. WSMO ne spécifie pas un modèle pour les propriétés non fonctionnelles. En utilisant WSML [Bru05], il est possible de définir les propriétés transactionnelles d'un service comme la valeur de la propriété `transactional`. Par comparaison à WSMO, OWL-S [Mar04] définit un nombre limité de propriétés non fonctionnelles. Cependant, elle offre la possibilité de définir de nouvelles propriétés, y compris une pour spécifier les propriétés transactionnelles, en utilisant la propriété *ServiceParameter* du concept *ServiceProfile*. Ainsi, il est possible d'ajouter une propriété permettant de définir les propriétés transactionnelles d'un service.

Conformément aux spécifications de coordination des services Web [NRFJ07, AFI+03], nous supposons que chaque service transactionnel possède un coordinateur transactionnel exposant ses transitions externes en tant qu'opérations permettant ainsi d'abandonner, d'activer, d'annuler ou de compenser le service correspondant.

5.3.2 Service composé transactionnel

Un service Web composé coordonne un ensemble de services pour réaliser un objectif commun. Un service Web composé transactionnel (SCT) est un service Web composé dont les services composants sont des services transactionnels. Un tel service utilise les propriétés transactionnelles de ses services composants pour spécifier des mécanismes de recouvrement.

Pour gérer la coordination de ses services composants, et en particulier les interactions qui existent entre ces services, un service composé définit des dépendances entre les services composants. Ces dépendances permettent de spécifier comment le service réagit aux changements d'états des autres services et comment il peut agir sur leurs comportements (i.e. quand est-ce qu'un service sera abandonné, activé, annulé ou compensé). Par exemple, le service composé défini dans la figure 5.1 spécifie que le service ≪Paiement en Ligne≫ (PL) sera activé après la terminaison des services ≪Réservation d'Hôtel≫ (RH) et ≪Réservation de Vol≫ (RV). Cela signifie qu'une dépendance d'activation existe entre les services RH et RV et le service PL.

Nous avons défini une forme générale de dépendance qui permet d'exprimer tous les types de relation (activation, alternative,...) qui peuvent exister entre plusieurs services. Les dépendances considérées dans notre approche sont les suivantes : activation, alternative, abandon, compensation et annulation :

– une dépendance *d'activation* exprime une relation de succession entre deux services, et définit un ordre total entre leurs activations,

- une dépendance *d'alternative* permet de définir une alternative d'exécution (mécanismes de recouvrement en avant),
- une dépendance *d'abandon* permet de propager les échecs (causant l'abandon du SCT) d'un service vers son ou ses successeur(s) en le(s) abandonnant,
- une dépendance de *compensation* permet de définir un mécanisme de recouvrement en arrière par compensation,
- enfin, une dépendance *d'annulation* permet de signaler les échecs d'exécution d'un service à d'autre(s) service(s) s'exécutant en parallèle en provoquant si nécessaire leur annulation.

Flot de contrôle et flot transactionnel

Les dépendances de *contrôle* (activation, abandon) et les dépendances *transactionnelles* (compensation, annulation et alternative) définissent respectivement le *flot de contrôle* et le *flot transactionnel* d'un service composé transactionnel, et sont liées par le fait qu'un flot transactionnel est défini en fonction du flot de contrôle auquel il est associé.

Le *flot de contrôle* d'un service composé transactionnel définit un ordre partiel entre les activation de ses services composants. Intuitivement, il est défini par l'ensemble de ses dépendances d'activation.

Le *flot transactionnel* d'un service composé transactionnel définit les mécanismes de recouvrement en cas d'échec. Intuitivement, il est défini par les propriétés transactionnelles de ses services composants (rejouable, pivot, compensable) et l'ensemble de ses dépendances transactionnelles.

Bien sûr, le flot transactionnel est étroitement lié au flot de contrôle. En effet, les mécanismes de recouvrement (définis par le flot transactionnel) dépendent de la logique d'exécution (définie par le flot de contrôle). Par exemple, dans le service composé de la figure 5.1, il est possible de définir une dépendance de compensation de RH vers RV parce que l'on peut induire de l'opérateur *AND-join* du flot de contrôle que l'échec de RH nécessite la compensation du travail partiellement effectuée, c'est-à-dire la réservation d'avion RV.

Plus généralement, un flot de contrôle définit implicitement tous les mécanismes de recouvrement possibles. Nous appelons *flot transactionnel potentiel* d'un flot de contrôle *fc* le flot transactionnel qui inclut toutes les dépendances transactionnelles qui peuvent être définies en fonction de *fc*. Plus formellement, chaque service composant s possède en fonction du flot de contrôle du service composé transactionnel :

- une condition potentielle de compensation qui spécifie quand s peut éventuellement être compensé.
- une condition potentielle d'alternative qui spécifie quand s peut éventuellement être activé en tant qu'alternative.
- une condition potentielle d'annulation qui spécifie quand s peut éventuellement être annulé.

5.3.3 Patrons transactionnels

Nous avons proposé le concept de patron transactionnel, un paradigme qui étend les patrons de workflow avec les dépendances transactionnelles vues précédemment afin de proposer des compositions fiables de services Web. De façon générale, un patron est une description

abstraite d'une forme récurrente dans un contexte spécifique [GHJV95]. Ainsi, un patron de workflow [vdAtHKB03] est défini comme une description abstraite d'une classe d'interactions. Par exemple, le patron *Synchronisation* (voir figure 5.3.b) décrit une classe d'interactions où un service est activé après la terminaison d'un ensemble de services.

Les patrons de workflow de base [vdAtHKB03] ne considèrent que l'aspect flot de contrôle. Par rapport à notre modèle de SCT, ils peuvent être considérés comme des patrons de flot de contrôle. Nous définissons un patron de composition comme une fonction qui retourne un *flot de contrôle* à partir d'un ensemble de services.

Un patron de workflow définit un flot de contrôle d'un ensemble de service. Comme tout flot de contrôle, le patron de workflow possède un flot transactionnel potentiel. Ainsi, nous définissons un *flot transactionnel potentiel* à partir du flot de contrôle de tout patron de workflow. Cela permet alors de définir pour chaque service composant les conditions potentielles de compensation, d'alternatives et d'annulation. Illustrons ceci sur trois exemples de patrons de workflow : le patron *Parallel Split*, le patron *Synchronisation*, et le patron *Exclusive Choice*.

Un **patron transactionnel** dérivé d'un patron de composition est un flot de contrôle enrichi par un flot transactionnel sélectionné dans son flot transactionnel potentiel.

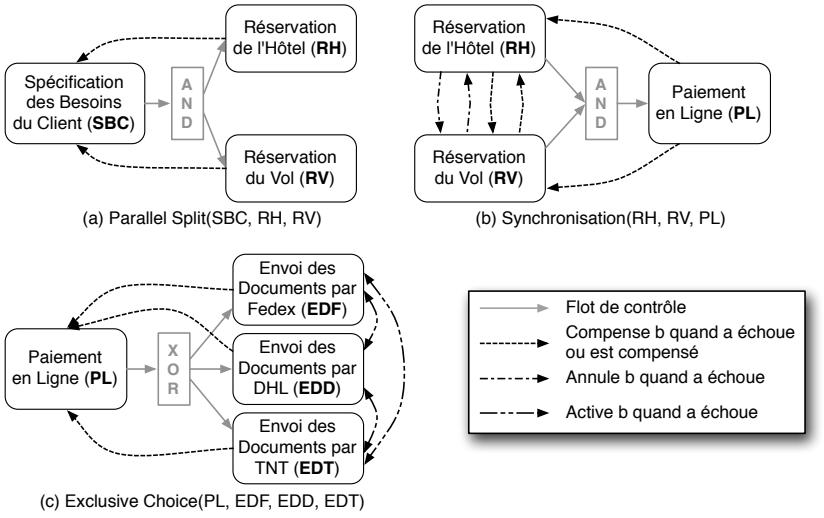


FIGURE 5.3 – Flot transactionnel potentiel des patrons Split, Choice et Synchronisation.

Si on considère le flot de contrôle *AND-Split* défini à la figure 5.3.a en grisé, SBC peut éventuellement être compensé en cas d'échec de RH ou de RV, ce qui représente le *flot transactionnel potentiel* du flot de contrôle *AND-Split*(SBC,RH,RV). De même, si on considère le flot de contrôle défini à la figure 5.3.b, RV peut être éventuellement compensé (respectivement annulé) suite à l'échec de RH, ou à l'échec de PL ou à la compensation de PL (respectivement l'échec de RH). Ceci signifie que le *flot transactionnel potentiel* du flot de contrôle *Synchronisation*(RH,RV,PL) permet de choisir entre la compensation ou l'annulation en fonction des besoins et/ou des propriétés transactionnelles des services disponibles. Enfin, le flot de contrôle défini à la figure 5.3.c définit que EDD peut éventuellement être activé comme alternative à EDF ou EDT. La figure 5.3.c illustre le *flot transactionnel potentiel* du flot de contrôle *Exclusive Choice*(PL,EDF,EDD,EDT).

Ainsi un patron transactionnel intègre deux dimensions complémentaires, l'une permettant d'exprimer la logique des processus métiers, l'autre permettant de définir les mécanismes de recouvrement. Montrons maintenant comment nous utilisons les patrons transactionnels pour définir des compositions fiables.

5.3.4 Compositions fiables de services Web

Nous utilisons les patrons transactionnels comme des briques de base pour spécifier des services Web composés. Un SCT peut être défini comme un ensemble de patrons transactionnels connectés (c'est-à-dire ayant des services composants en commun) d'une manière éventuellement imbriquée. La figure 5.4 montre le service composé d'organisation de voyage en ligne spécifié comme la composition de trois patrons transactionnels : *Parallel Split*, *Synchronisation* et *Exclusive Choice*.

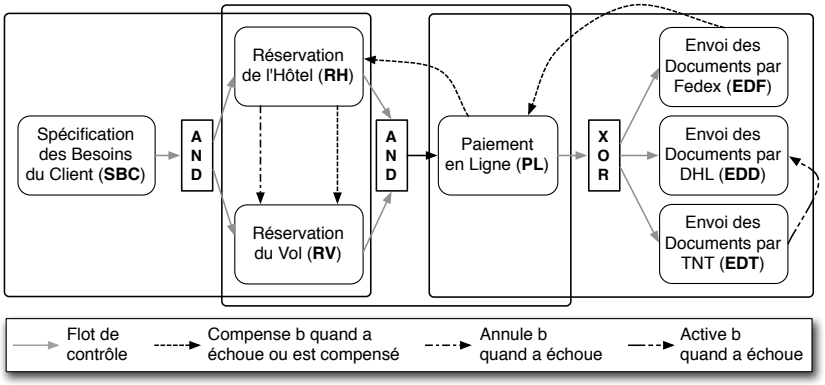


FIGURE 5.4 – Service composé défini comme une union de patrons transactionnels.

Composer un ensemble de *patrons transactionnels* peut cependant conduire à des incohérences à la fois au niveau du contrôle et/ou au niveau transactionnel. Par exemple, une incohérence du contrôle est présente quand les patrons transactionnels sont disjoints ou quand une instance du patron *Exclusive Choice* est suivie par une instance du patron *Synchronisation*. De même, une incohérence transactionnelle est présente quand par exemple un service composant peut échouer causant l'abandon du SCT alors que les effets de l'exécution partielle n'est pas compensée. Prenons par exemple le SCT défini à la figure 5.4, nous remarquons d'une part que le service PL peut échouer (puisqu'il est spécifié que RH va être compensé quand PL échoue), et d'autre part que RV n'est pas compensé quand PL échoue. D'une façon générale, la cohérence transactionnelle assure la fiabilité du service composé. Dans ce qui suit, nous montrons comment nous assurons la cohérence du contrôle et la cohérence transactionnelle d'un SCT défini en connectant des patrons transactionnels.

Pour assurer la cohérence du contrôle, nous avons proposé un certains nombres de règles qui nous permettent d'assurer que la composition des patrons d'un SCT est cohérente. Ainsi, un flot de contrôle cohérent :

– doit commencer par une instance d'un patron *Sequence* ou d'un patron de parallélisme, *Parallel Split*, *Exclusive Choice* ou *Multi Choice*,

– doit s'assurer qu'un patron *Sequence* peut être suivi par un patron *Sequence* ou par un patron de parallélisme,
– doit s'assurer qu'un patron *Parallel Split* peut être suivi par n'importe quel patron de synchronisation,
– doit s'assurer que le patron *Multi Choice* doit être suivi par une un des patrons *Synchronisation* ou *Simple Merge*
– doit s'assurer que le patron *Exclusive Choice* doit être suivi par un patron *Simple Merge*.

De plus, un service composant dans un SCT peut être lui même un service composé dont le flot de contrôle est cohérent (respecte les règles), ce qui permet ainsi d'utiliser les patrons transactionnels de façon imbriquée.

On peut remarquer que les mécanismes de recouvrement de notre modèle découlent de trois règles transactionnelles plus générales :
R1 : lors de l'échec d'un service, il faut toujours essayer une alternative si elle existe,
R2 : lors de l'échec d'un service causant l'abandon du service composé, il faut compenser le travail partiel déjà effectué,
R3 : lors de l'échec d'un service causant l'abandon du service composé, il faut annuler toute exécution en cours (cette règle permet de prendre en compte le parallélisme entre les sous transactions pour les services composés).

Nous appelons service intuitivement valide tout service composé qui respecte ces règles de cohérence transactionnelles.

Si on analyse le service composé spécifié dans la figure 5.4, on remarque qu'il n'est pas intuitivement valide puisqu'il ne respecte ni R1 pour le service EDF (puisque lors de son échec, aucune des ses alternatives, EDT ou EDD, n'est activée) ni R2 pour le service PL (puisque lors de son échec, la réservation du vol est maintenue). Au contraire, le service illustré par la figure 5.5 est intuitivement valide puisqu'il respecte les trois règles pour chacun de ses services composants, grâce à l'ajout d'une dépendance entre EDF et EDD, et d'une dépendance entre PL et RV.

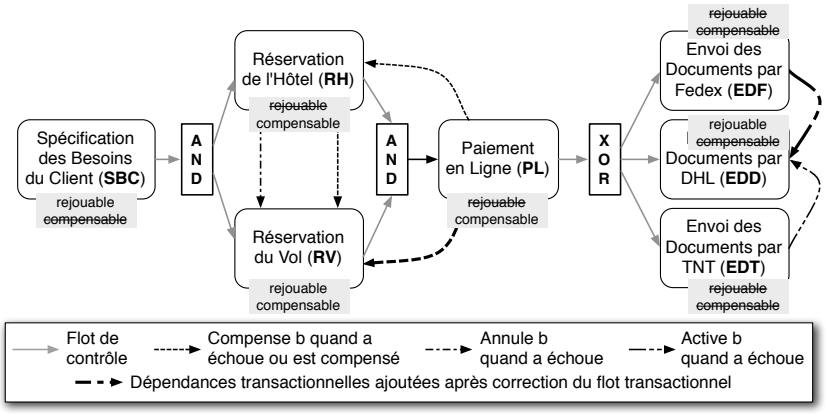


FIGURE 5.5 – Service composé valide.

5.3.5 Comment assurer la cohérence transactionnelle

Nous allons terminer cette présentation en montrant comment on assure la cohérence transactionnelle d'un service composé en utilisant les services transactionnels et les patrons transactionnels.

La première étape pour assurer la cohérence transactionnelle consiste à déterminer les propriétés transactionnelles des services composants. Si ces propriétés transactionnelles ne sont pas connues, nous appliquons les règles suivantes (dans l'ordre indiqué) pour les calculer :

1. chaque service est par défaut *rejouable* et *pivot*.
2. chaque service cible d'une dépendance de compensation est *compensable*.
3. chaque service source d'une dépendance de compensation (activé suite à son échec), d'annulation ou d'alternative n'est pas *rejouable*.

Dans le SCT défini à la figure 5.4, la dépendance de compensation de RH vers RV implique que RH n'est pas rejouable (puisqu'il peut échouer) et que RV est compensable. De même, nous pouvons déduire que le service EDD est rejouable puisqu'il n'est source d'aucune dépendance de compensation, d'alternative ou d'annulation, contrairement au service EDF qui n'est pas rejouable. La figure 5.5 illustre les propriétés transactionnelles déduites à partir du SCT initialement spécifié par les concepteurs (illustré par la figure 5.4).

Les modèles de transactions ne permettent pas de prendre en compte les besoins spécifiques des concepteurs en terme de fiabilité. En adoptant un modèle transactionnel donné les concepteurs héritent des mécanismes de recouvrement définis par le modèle.

Une façon d'assurer la fiabilité selon les besoins spécifiques des concepteurs est de leur permettre de définir l'Ensemble des États de Terminaison Acceptés (ETA) de la transaction. ETA a été introduit initialement par [ELLR90], comme un critère de correction des exécutions, et a été repris par [KS95, BPG05]. En fait, même les modèles de transactions considèrent implicitement un ETA. Cependant, cet ETA est prédéfini par l'ensemble des mécanismes de recouvrement du modèle et non en fonction des besoins des concepteurs.

Dans [BPG05], nous avons montré comment de générer les mécanismes de recouvrement en fonction des besoins spécifiques des concepteurs exprimés sous forme d'ETA. Si l'aspect flexibilité est intéressant, l'approche présente un problème qui est dû à la difficulté de spécifier l'ETA, notamment quand le nombre de services composants est assez élevé.

Afin de prendre en compte les besoins spécifiques des concepteurs tout en évitant de spécifier l'ETA, nous utilisons les règles de cohérence transactionnelles définies ci-dessus comme un repère de correction et pas comme un critère de correction automatique. En partant du modèle de composition initialement défini, nous procédons d'une façon interactive avec les concepteurs en générant un ensemble de propositions pour assurer un flot transactionnel intuitivement valide. Les concepteurs peut accepter ou refuser ces propositions selon leurs besoins. Nous utilisons pour cela un ensemble de règles définies comme suit :

- la première règle vise à assurer la règle de cohérence transactionnelle R1. Elle assure qu'au moins une condition d'alternative d'un service s éventuellement vraie est une condition d'alternative de s.
- la deuxième règle vise à assurer la règle de cohérence transactionnelle R2. Elle assure que toute condition de compensation d'un service s éventuellement vraie est une condition de compensation de s (évidement s devra être dans ce cas compensable).

– la troisième règle vise à assurer la règle de cohérence transactionnelle R3. Elle assure que toute condition d'annulation d'un service s éventuellement vraie est une condition d'annulation de s.

Voyons comment nous pouvons appliquer ces règles pour améliorer le flot transactionnel du service composé initialement défini illustré par la figure 5.4 :

– pour EDD, $EDF.échoué$ est éventuellement vraie (car EDF n'est pas rejouable) et n'est pas une condition d'alternative de EDD. En appliquant la première règle, nous générons la suggestion S_1. L'application de la première règle permet de générer les suggestions S_2 et S_3.

  S_1 : ajouter une dépendance d'alternative de EDF vers EDD,

  S_2 : ajouter une dépendance d'alternative de EDT vers EDF.

  S_3 : ajouter une dépendance d'alternative de EDF vers EDT.

– pour RV, la condition de compensation potentielle $PL.échoué$ est éventuellement vraie (car PL n'est pas rejouable) et n'est pas une condition de compensation de RV. En appliquant la deuxième règle, nous générons la suggestion S_4. De même, l'application de la deuxième règle à SBC permet de générer la suggestion S_5.

  S_4 : ajouter une dépendance de compensation de PL vers RV,

  S_5 : ajouter une dépendance de compensation de RH vers SBC.

Les concepteurs peuvent rejeter la suggestion S_2 puisque le service EDF est non rejouable, la suggestion S_3 puisque EDD est déjà une alternative de EDF et la suggestion S_5 puisque le service SBC est sans effet. La figure 5.5 illustre le service d'organisation de voyage en ligne après correction.

5.3.6 Implantation

La faisabilité de notre approche est démontrée par un environnement de conception et de validation de services Web composés fiables. Cet environnement est intégré à Bonita, un système de workflow développé par l'équipe ECOO. Cet environnement offre une interface graphique et implante les différentes techniques que nous avons développées précédemment. Comme illustré par la figure 5.6, le concepteur dispose :

– d'un ensemble de services Web avec leurs propriétés transactionnelles,
– d'un ensemble de patrons transactionnels,
– d'une page pour visualiser toutes les informations sur un service particulier : ses opérations ainsi que ses propriétés transactionnelles,
– d'une page pour spécifier un service composé. Le concepteur sélectionne les opérations des services Web et définit la logique de leurs invocations en utilisant les opérateurs de contrôle dont il dispose. Il peut aussi enrichir le flot de contrôle spécifié par un ensemble de mécanismes de recouvrement.

L'architecture de notre prototype est définie de la manière suivante. Comme de nombreux moteurs d'exécution permettant l'exécution de code WSBPEL, nous avons choisi d'adapter un moteur de workflow existant, Bonita [VC]. Bonita est un système de workflow développé au-dessus de JonAS qui est une implantation de la spécification J2EE faite par ObjectWeb. Nous avons choisi Bonita pour deux raisons principales. La première, c'est que Bonita est un moteur

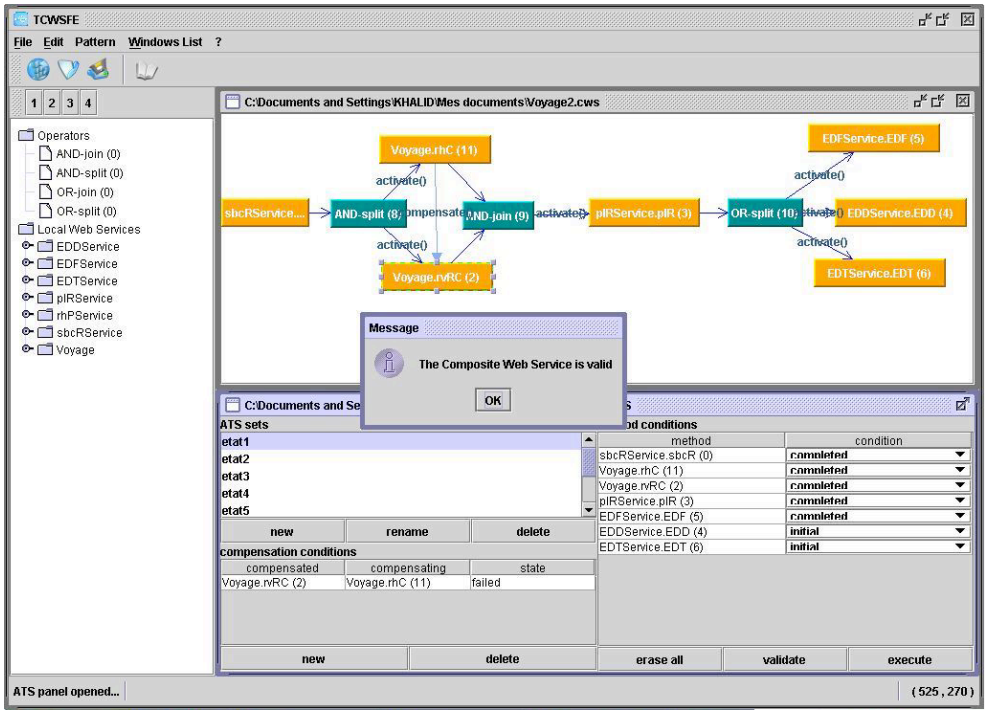


FIGURE 5.6 – Interface graphique fournie par notre environnement de conception.

de workflow qui permet d'exécuter des services Web (il supporte les appels SOAP), quelle que soit leur localisation (dans la figure 5.6, les services sont situés localement, mais cela n'est pas obligatoire). La deuxième raison concerne la conception modulaire de Bonita qui nous a permis de facilement implanter nos propositions concernant les aspects transactionnels.

Afin d'implanter notre propre approche transactionnelle, nous avons dû adapter et étendre Bonita pour qu'il puisse prendre en compte les propriétés et les dépendances transactionnelles supportées par les patrons transactionnels (rejouable, pivot et compensable). Dans un premier temps, nous avons extrait du moteur d'exécution la gestion transactionnelle des activités. Nous avons identifié les aspects communs aux différents modèles transactionnels, ainsi que les aspects qui sont fortement dépendant d'un modèle particulier. Puis nous avons implanté un mécanismes de modules qui permet à Bonita d'assurer la gestion des aspects communs, alors que les aspects particuliers d'un modèle transactionnel donné sont pris en charge par le module dédié. Le moteur de Bonita a été donc modifié afin d'être complètement indépendant d'un modèle transactionnel particulier. Ainsi, lorsque le moteur rencontre une propriété transactionnelle telle que compensable (il y a une demande de compensation), il délègue au module concerné la gestion de cette propriété.

5.4 Synthèse

Les travaux autour des compositions fiables de services Web correspondent à une étape importante de nos travaux. Nous avons introduit un modèle de services Web transactionnel

permettant (i) d'enrichir la description de services Web pour mieux exprimer leurs propriétés transactionnelles et (ii) de modéliser un service Web composé (SC) dont les services composants manifestent de telles propriétés.

Si l'approche peut paraître simple, la prise en compte de besoins non-fonctionnels (l'aspect transactionnel) dans la composition de services était assez novatrice à l'époque des premières versions du modèle. Et nous verrons dans le chapitre Perspectives que de nombreux problèmes demeurent quant aux aspects non-fonctionnels dans la composition de services. L'intérêt de l'approche réside dans le fait qu'elle permet de spécifier des compositions de services ayant des structures de contrôle aussi complexes que les modèles des workflows, tout en assurant leur fiabilité notamment en tenant compte des besoins spécifiques des concepteurs.

Articles

Patrons Transactionnels pour Assurer des Compositions Fiables de Services Web.
Sami Bhiri, Olivier Perrin, Claude Godart.
In Technique et Science Informatiques, 28(3) : 301-330, 2009.

Extending workflow patterns with transactional dependencies to define reliable composite Web services.
Sami Bhiri, Olivier Perrin, Claude Godart.
Proceedings of AICT/ICIW 2006.

Transactional patterns for reliable web services compositions.
Sami Bhiri, Claude Godart, Olivier Perrin.
Proceedings of ICWE 2006.

A Contract Layered Architecture for Regulating Cross-Organisational Business Processes.
Mohsen Rouached, Olivier Perrin, Claude Godart.
Proceedings of Business Process Management 2005.

Overview of Transactional Patterns : Combining Workflow Flexibility and Transactional Reliability for Composite Web Services.
Sami Bhiri, Khaled Gaaloul, Olivier Perrin, Claude Godart.
Proceedings of Business Process Management 2005.

Reliable Web Services Composition using a Transactional Approach.
Sami Bhiri, Claude Godart, Olivier Perrin.
Proceedings of EEE 2005.

A Contract-Based Approach for Monitoring Collaborative Web Services Using Commitments in the Event Calculus.
Mohsen Rouached, Olivier Perrin, Claude Godart.
Proceedings of WISE 2005.

Ensuring required failure atomicity of composite Web services.
Sami Bhiri, Olivier Perrin, Claude Godart
Proceedings of WWW 2005.

A Transaction-Oriented Framework for Composing Transactional Web Services.
Sami Bhiri, Claude Godart, Olivier Perrin.
Proceedings of IEEE SCC 2004.

5.5 Travaux en cours

J'ai soulevé à la fin du chapitre 4 sur les processus la difficulté à prendre en compte au même niveau le flux de contrôle et le flux de données. Si on se place dans le contexte des services, et de la composition de services, le problème est à peu près le même. Notre contribution, par l'intermédiaire de la prise en compte des aspects transactionnels des services, constitue un premier élément de réponse. Jusqu'à très récemment, les données n'étaient pas du tout considérées dans les papiers à propos de la composition de services Web, et seul le modèle Colombo [BCG+05b, BCG+05a] propose leur prise en compte. Là encore, raisonner sur une composition en prenant en compte à la fois les opérations des services et leurs données nécessite d'avoir un niveau d'abstraction suffisant de manière à pouvoir raisonner sur la composition. Le problème de la composition devient également encore plus intéressant si l'on se place dans le contexte dans lequel les services sont conversationnels et avec état (*stateful*).

Nous avons commencé à travailler sur un autre aspect important dans le contexte de la composition de services Web. Il s'agit de prendre en compte le temps. Le problème est le suivant : étant donné un ensemble de services Web, un service abstrait (le service but) décrivant le résultat que l'on souhaite obtenir, comment composer cet ensemble de services de telle sorte que le service but soit satisfait ? Pour ce faire, on essaie de coordonner les services Web capables d'échanger des messages. Cependant, si la coordination ne satisfait pas le service but, nous essayons alors de générer un nouveau service, appelé *Médiateur*. Le rôle de ce dernier est d'essayer de générer les flux de messages manquants à la composition, i.e., lorsque un service indispensable à la composition attend un message pour s'exécuter et qu'il ne le reçoit pas, alors le médiateur va tenter de générer ce message. Ces travaux servent maintenant de base à l'intégration de contraintes temporelles au niveau des services Web considérés.

Plusieurs travaux ont été proposés dans le cadre de la modélisation des contraintes temporelles dans les services Web [BCPT05, AD94, Pon06, PBCT07]. Dans le cadre de l'analyse de la compatibilité et de la substitution dynamique des services Web, B. Benatallah et al. [BCF04] ont proposé d'étendre les protocoles de conversation (business protocols) par les contraintes temporelles d'activation. Les protocoles de conversations permettent de modéliser le comportement externe des services Web par les séquences de messages supportées. Les protocoles de conversations sont modélisés par les machines à états finis, où les états correspondent aux états du service et les transitions correspondent aux échanges de messages (messages reçus ou émis). Les contraintes temporelles d'activations permettent de déclencher une transition automatiquement et ce après l'écoulement d'un certain délai. Ce travail a été étendu pour tenir compte des formes temporelles plus riches et plus expressives. Les contraintes temporelles définies dans [Pon06]

permettent d'exprimer des contraintes qui dépendent des contraintes temporelles précédentes. Par exemple, pour la présentation d'un article dans une conférence, il faut qu'un délai d'au moins 15 jours soit passé après l'enregistrement. Cependant, ce modèle ne permet pas d'exprimer certaines formes temporelles telle que l'expression de contraintes temporelles basées sur le moment de déclenchement des opérations. Ceci est dû au fait que aucune trace n'est conservée sur le moment de déclenchement des opérations.

Le travail présenté dans [KPP06] vise à fournir une approche de modélisation des propriétés temporelles dans la composition des services Web, définie par un ensemble de processus BPEL. L'ensemble de ces processus sont modélisés par les systèmes à transitions (*State Transition System*) STS. Ces derniers ont été étendus par les propriétés temporelles. Pour analyser ces propriétés temporelles, le formalisme QDDC (*Quantified Discrete-time Duration Calculus*) a été utilisé. Nous nous sommes quant à nous basés sur le modèle présenté dans [KPP06] pour étendre le formalisme WSTTS (*Web Services Timed Transition Systems*), et ce pour considérer d'autres formes de contraintes temporelles plus expressives.

Nous menons également des travaux dont l'objectif est de permettre de composer un ensemble de services, de vérifier la composition par rapport à un ensemble de propriétés, et de surveiller l'exécution de la composition en utilisant le même niveau d'abstraction. Pour cela, nous nous basons sur l'utilisation du calcul d'événements.

Articles

An Integrated Declarative Approach to Web Services Composition and Monitoring.
Ehtesham Zahoor, Olivier Perrin, and Claude Godart.
Proceedings of the Tenth International Conference on Web Systems Engineering, WISE 2009. To be published.

Mashup Model and Verification using Mashup Processing Network.
Ehtesham Zahoor, Olivier Perrin, Claude Godart.
Proceedings of the 4th International Conference on Collaborative Computing, CollaborateCom 2008, ACM.

Dynamic Web Services Provisioning with Constraints.
Eric Monfroy, Olivier Perrin, Christophe Ringeissen.
Proceedings of CoopIS 2008.

A Mediator Based Approach For Services Composition.
Nawal Guermouche, Olivier Perrin, Christophe Ringeissen.
Proceedings of SERA 2008.

Timed Specification For Web Services Compatibility Analysis.
Nawal Guermouche, Olivier Perrin, Christophe Ringeissen.
Proceedings of WWV 2007.

Chapitre 6

Perspectives

Mes perspectives de recherche concernent plusieurs aspects autour de la composition de services Web. Les services Web sémantiques peuvent être considérés comme une solution aux problèmes de l'intéropérabilité. En effet, sous cette appellation, on trouve à la fois l'aspect données et la représentation de ces données, les aspects fonctionnels et opérationnels, et les aspects non-fonctionnels comme le temps, la fiabilité, la qualité de service, la sécurité. Un de mes objectifs est donc de m'intéresser à la problématique de la composition de services, en considérant simultanément les aspects suivants : la représentation formelle d'une composition et la capacité à raisonner sur les propriétés attendues, la description sémantique des services et des données qu'ils manipulent, et la prise en compte d'aspects non fonctionnels ou non opérationnels. Plus particulièrement, nous envisageons trois approches possibles : les aspects temporels, la sécurité dans les compositions de services, et les données.

6.1 Les aspects temporels d'une composition

Un premier objectif a trait à l'intégration du temps. Actuellement, ce critère est peu pris en compte dans les propositions de modèles de composition, or il existe de nombreux exemples dans lesquels le temps intervient : un service est accessible pour une durée donnée (référence absolue), ce qui peut avoir un impact sur la composition. Un autre exemple concerne l'accessibilité d'un service pendant une durée donnée à partir de la fin d'un autre service (référence relative), ce qui là encore a un impact sur la composition. Introduire le temps dans la composition n'est pas une chose triviale, et pour cela, nous pensons que nous avons besoin de nous intéresser à une formalisation qui permettra de raisonner sur la composition de manière automatique. Nous avons déjà initié ce genre de travaux dans [12,13,18]. Dans [Tou05, PBCT07], on trouve des propositions pour la notion d'analyse de compatibilité et de remplaçabilité pour des protocoles avec des aspects temporels.

La composition de plusieurs services Web devient un protocole contraint par des propriétés temporelles. Dans nos précédents travaux, nous avons abordé les aspects transactionnels des services que nous souhaitions composer. Les résultats sont encourageants, mais nous nous sommes posés la question de savoir si par exemple un service pouvait voir certaines de ses propriétés changer avec le temps. Par exemple, un service est rejouable pendant 3 jours, puis il devient pivot. On peut avoir ce genre de situation dans le cas d'achats sur Internet, avec un service qui

est annulable pendant une période donnée (7 jours), puis qui devient non modifiable ensuite. Dans ce cas, on voit bien que le temps joue un rôle non négligeable sur la composition obtenue, et qu'il peut être nécessaire d'intégrer ce concept de temps dans la composition elle-même. De la même façon, un service peut éventuellement être remplacé par un autre service mais dans un intervalle de temps donné. Par exemple, on peut supposer qu'un service sera accessible dans une plage de temps définie, ce qui là encore a une implication sur la composition.

Les conséquences de ces contraintes liées au temps sont doubles. D'une part, il faut pouvoir être capable d'exprimer les contraintes temporelles des services que l'on souhaite composer, et il faut être également en mesure de raisonner sur la composition en termes de respect des contraintes temporelles. Pour cela, il faut pouvoir travailler à un niveau d'abstraction qui permette de raisonner sur la composition. Nous avons donc commencé des travaux qui permettent d'exprimer certaines contraintes sur les services et qui permettent de spécifier dans une logique temporelle ces contraintes. Le problème est alors de pouvoir se ramener à une logique du premier ordre afin de pouvoir raisonner. Nous avons ainsi travaillé sur des contraintes sur les services en termes de contractualisation. Nous avons ensuite transformer ces contraintes en engagements, puis nous avons exprimé ces engagements dans le calcul d'événements. Tout ce processus est actuellement manuel, et ce que nous souhaitons faire maintenant est de pouvoir raisonner de manière automatique. Pour cela, nous avons commencé à travailler avec le projet CASSIS afin de pouvoir coder une logique temporelle (calcul d'événement discret, calcul de situation,...) dans une logique du premier ordre et de vérifier les propriétés attendues au moyen d'un système de *Model Checking* tel que UPPAAL . Ce système est capable de gérer efficacement des systèmes modélisés comme des automates temporisés, avec en plus un support pour des types de données simples tels que les entiers ou les tableaux.

Si la partie raisonnement sur la composition *a priori* est primordiale, la partie surveillance et vérification *a posteriori* des compositions obtenues est également importante. Nous comptons donc travailler sur la surveillance de l'exécution du service précédemment composé. Cet aspect sera abordé grâce aux travaux menés précédemment dans le projet ECOO sur la découverte de procédés. L'idée est de pouvoir vérifier l'exécution réelle d'une composition à partir des traces d'exécution, en prenant en compte les aspects temporels pris en compte dans la première partie. L'exécution telle qu'elle s'est réellement produite sera ensuite comparée à celle qui était prévue dans la composition de départ. Des différences peuvent en effet survenir (service non disponible à un instant donné, ré-exécution multiple d'un service,...). Ces travaux devrait également permettre de savoir s'il n'y a pas eu de problèmes concernant la sécurité (par exemple DoS sur un service...).

6.2 La sécurité et les services Web

Un deuxième objectif concerne toujours la composition de services, mais cette fois en présence de contraintes supplémentaires qui ne sont pas des contraintes fonctionnelles, à savoir des contraintes concernant la sécurité. Nous avons commencé à travailler sur ces problèmes dans les projets COPS et COWS. L'idée est de s'intéresser à la synthèse de services Web en prenant en compte les aspects authentification, autorisation,... Il s'agit de contraintes non-fonctionnelles qui ont un impact sur le résultat de la composition, et sur son exécution. Comme en ce qui concerne les aspects temporels, le protocole issu de la composition devra respecter toutes les contraintes fixées pour le service composé et toutes les contraintes des services composants. Une

fois encore, le problème est de pouvoir trouver un modèle (un niveau d'abstraction) qui permette d'exprimer ces contraintes de manière précise, et qui permette ensuite de raisonner sur la composition obtenue, afin de calculer la validité de la composition par rapport aux contraintes initiales, et de vérifier à l'exécution que la composition ne viole pas ces contraintes.

Il existe actuellement encore peu de travaux en ce qui concerne la sécurité des interactions entre différents services lorsque l'on désire composer ces derniers. En particulier, les trois questions suivantes sont actuellement sans réponse :

- comment définir un modèle de contrôle d'accès qui soit compatible avec les besoins et le fonctionnement des architectures orientées service (*Service Oriented Architecture*),
- comment savoir si une politique de sécurité est valide et décidable étant donné un modèle de composition de services,
- comment vérifier que l'exécution d'une politique de sécurité est correcte.

De plus, si l'on se place dans le contexte des services Web, plusieurs modèles de sécurité différents (issus des systèmes que les services sont censés abstraire) coexistent, et une composition de services prend en compte ces différents modèles. Le problème posé est donc celui qui consiste à être sûr que malgré ces différences, la composition des services ne sera pas exposée à des problèmes liés à la sécurité.

Actuellement, ce n'est pas possible. D'une part, il existe beaucoup de standards dans la pile WS, et certains ont des intersections non vides, d'autres sont concurrents. D'autre part, il n'y pas de modèle formel sous-jacent permettant de calculer si les propositions d'un service client en terme de sécurité (au sens large) sont compatibles avec celles d'un service fournisseur. Une deuxième catégorie de problème concerne l'audit, c'est-à-dire la vérification a posteriori (cf. 6.1) que les obligations des uns et des autres se sont déroulées comme prévu. Cela doit permettre de pointer les endroits où il y a eu des problèmes. Pour tenter de répondre à ces deux problèmes, on envisage de définir un modèle qui permette d'unifier la notion de sécurité en termes d'authentification, d'autorisation, d'obligations, d'intégrité, d'intimité,... Si l'on arrive à définir un modèle (formel) qui prenne en compte tous ces aspects, on aura fait un grand pas, puisque l'on aura la possibilité de raisonner sur des concepts bien définis. Ensuite, il s'agirait de décliner chaque WS-spécification dans le modèle. Ensuite, on pourrait proposer des procédures permettant d'analyser et de gérer la sécurité en se servant de la proposition du point précédent. On pourrait ainsi calculer la compatibilité entre deux services d'un point de vue sécurité, calculer la remplaçabilité (similarité et différences entre deux services), et analyser la cohérence d'un ensemble de spécifications liées à la sécurité pour un service donné. Bien sur, on devra prendre en compte l'aspect multi-niveaux (intimité, contrôle d'accès, confiance,...). Les questions auxquelles il faudra répondre sont les suivantes :

- quel type de formalisme : assertions/contraintes, règles,... ? sur quels objets ? pour quel aspect de la sécurité ?
- comment coder les politiques de sécurité et le protocole (interactions multiples) ?
- comment surveiller ? cohérence avec le formalisme choisi précédemment ?
- correspondance entre une politique de sécurité et un ensemble de services ?

L'objectif des travaux que je compte mener est donc de proposer un modèle de composition qui permette d'obtenir un protocole permettant l'interaction entre les différents services appartenant à la composition et paramétré par les politiques de sécurité de chaque service et par la politique de sécurité attendue.

6.3 L'aspect données

Jusqu'à présent, peu de travaux s'intéresse aux données lorsque l'on parle des services Web. Pourtant, ces dernières sont présentes en permanence, mais elles ne sont pas mises en avant. Avec nos travaux sur les aspects transactionnels des services, nous avons commencé à nous intéresser à cet aspect, mais il reste encore beaucoup de travail. Pourquoi doit-on s'intéresser aux données dans le contexte des architectures orientées services ? Les données sont un point crucial de ces architectures. En effet, lorsque l'on parle de services, on parle d'échanges de messages qui contiennent des données, c'est-à-dire des représentations des objets des systèmes dont les services exposent une partie des fonctionnalités. Cependant, jusqu'à présent, très peu de travaux se sont intéressés à la gestion de ces données. Pourtant, quelques problèmes existent.

Le premier problème concerne le fait que, très souvent, les données ont des dépendances les unes avec les autres. C'est un problème que le couplage faible et l'autonomie des services font oublier. Par exemple, si on crée au sein de la même organisation deux services CommandeClient et GestionClient, ces deux services ne sont pas censés avoir des échanges de messages, et pourtant, on peut imaginer qu'ils collectent souvent leurs données de la même base de données. De même, ces deux services peuvent avoir besoin de données gérées par l'autre service. Cela a un impact, car si on imagine la situation dans laquelle on souhaite récupérer toutes les commandes faites par un client (pour lui accorder une remise par exemple), comment doit-on procéder, comment est-on sûr que les différentes commandes ont été enregistrées avec une référence valide vers ce client pour le service GestionClient, comment gère-t-on les contrôles qui peuvent être demandés concernant la solvabilité d'un client, comment gère-t-on les accès concurrents aux mêmes données si un service échoue,... Ces problèmes sont encore plus difficiles si on suppose deux services qui ne sont pas issus de la même entreprise. On constate donc sur cet exemple pourtant simple qu'il faut s'intéresser aux données lorsque l'on souhaite comprendre les interactions entre plusieurs services.

Un problème corrélé au problème précédent concerne la sémantique des données manipulées par les services, c'est-à-dire non plus seulement l'information brute (son type), mais également la connaissance sur ces informations. Comme indiqué dans le chapitre sur l'intégration de données, l'interprétation d'une donnée ne peut se contenter d'une annotation dite sémantique ou d'une ontologie. La connaissance dépend également des relations de la donnée avec d'autres données, du contexte d'utilisation de la donnée en intégrant des propriétés temporelles par exemple, ou de la possibilité de pouvoir transformer une donnée [PLC+08, Wie92, MGB+08].

Un deuxième problème concerne le fait que les données, une fois extraites du système (ERP, CRM, base de données, WfMS,...), peuvent éventuellement être invalidées, périmées,... Supposons à nouveau un service qui permette de passer une commande. On peut préparer son bon de commande, puis le fournir en attachement à l'appel SOAP du service. Le bon de commande est alors analysé par le système abstrait par le service. Le bon de commande est ensuite renvoyé au client afin que ce dernier l'accepte. Or, il se peut que entre l'envoi du bon de commande à confirmer par le client et la réponse de celui-ci, certaines données soient invalidées. Par exemple, l'article n'est plus disponible, son prix a été modifié, la date de livraison a été mise à jour, la remise potentielle n'est plus valide,... En fait, tout ce concerne la logique métier des données manipulées par les services ne sont pas répercutées. Cet exemple montre le problème qui se pose une fois que les données extraites ne sont plus sous contrôle de l'application qui les manipule en temps normal. Cela pose donc le problème de pouvoir contractualiser les services et surtout les

données qui sont associées à ces services. Ce problème peut être relié au problème cité en 6.1 sur l'inclusion du temps dans les compositions (et les instances des compositions – les protocoles) de services.

Enfin, une autre catégorie de problèmes concerne l'intimité des données (*privacy*). Là encore, une fois les données extraites de la base de données, du système ERP, ou du WfMS, puis utilisées en dehors des frontières de l'organisation qui les gère, il faut garantir qu'elles ne seront pas utilisées d'une manière non prévue, et que les règles de propriété sur lesquelles se sont entendus les services seront respectées. Ce point peut d'ailleurs être relié aux aspects sécurité du paragraphe 6.2.

On constate donc que les problèmes relatifs aux données dans les architectures orientées service sont difficiles. Pour les aborder, il faut dépasser le stade actuel de la recherche sur les services afin de réfléchir aux liens entre les objets métiers, les données, les systèmes gérant ces données et les services (et leurs interactions). Ainsi, les problèmes concernant l'identité des données, la fédération de données, la réplication, les transactions, la sémantique des données, l'intimité des données, la gestion de l'état des données dans le temps, les services gérant de grande masse de données, la sécurité des données,... sont autant de problèmes qui sont relativement peu abordés par les spécifications actuelles, et sont des problèmes qui me motivent énormément.

Bibliographie

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and Limitations of Current Workflow Management Systems. *IEEE Expert - Special Issue on Cooperative Information Systems*, 12(5), 1997.

[AAP+99] J. Andreoli, D. Arregui, F. Pacull, M. Riviere, J. VionDury, and J. Willamowski. Clf/mekano : a framework for building virtual-enterprise applications, 1999.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[AFI+03] Arjuna, Fujitsu, IONA, Oracle, and Sun. Web services composite application framework (ws-caf). http ://www.arjuna.com/standards/ws-caf/, 2003.

[AG01] Samuil Angelov and Paul W. P. J. Grefen. B2b econtract handling - a survey of projects, papers and standards. In *CTIT Technical Report 01-21*. University of Twente, 2001.

[AMVW99] Will M. P. van der Aalst, D. Moldt, R. Valk, and F. Wienberg. Enacting interorganizational workflow using nets in nets. In J. Becker, M. Zur Muhlen, and M. Rosemann, editors, *Workflow Management'99*, pages 117–136. Universitat Munster, Task Force Workflow, 1999.

[BCF04] B. Benatallah, F. Casati, and F.Toumani. Analysis and management of web service protocols. In *Int : Procs of ER'04, Shanghai, China. (2004)*, 2004.

[BCG+05a] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, Maurizio Lenzerini, and Massimo Mecella. Modeling data & processes for service specifications in colombo. In Michele Missikoff and Antonio De Nicola, editors, *EMOI-INTEROP*, volume 160 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[BCG+05b] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 613–624. ACM, 2005.

[BCPT05] Boualem Benatallah, Fabio Casati, Julien Ponge, and Farouk Toumani. On temporal abstractions of web service protocols. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05). Short Paper Proceedings*, 2005.

[BHP00] Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4) :55–63, 2000.

[Bit03] J. Bitcheva. *PointSynchro : un service pour la coordination de procédés interentreprises coopératifs*. PhD thesis, Université Henri Poincaré – Nancy 1 à Nancy, 14 novembre 2003.

[BN84] Andrew Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1) :39–59, 1984.

[BPG05] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In Allan Ellis and Tatsuya Hagino, editors, *WWW*, pages 138–147. ACM, 2005.

[Bru05] Jos De Bruijn. D16.1v0.21 the web service modeling language wsml. *available from http ://www.wsmo.org/TR/d16/d16.1/v0.21/*, 2005.

[CCT03] D. K. W. Chiu, S. C. Cheung, and S. Till. A three layer architecture for e-contract enforcement in an e-service environment. In *36th Hawaii International Conference on System Sciences (HICSS-36)*, 2003.

[Coa99] WfMC (Workflow Management Coalition). Terminology & glossary. In *WfMC-TC-1011 Version 3.0*. 1999.

[CUs02] CUseeMe. *Video conferencing and video chat with CUseeMe software*. www.cuseeme.com, 2002.

[Dem92] William Edwards Deming. Out of the crisis : Quality, productivity and competitive position. Cambridge University Press, Cambridge, MA, USA, 1992.

[DHPV09] Alin Deutsch, Richard Hull, Fabio Patrizi, and Victor Vianu. Automatic verification of data-centric business processes. In Ronald Fagin, editor, *ICDT*, volume 361 of *ACM International Conference Proceeding Series*, pages 252–267. ACM, 2009.

[DJ00] Klaus R. Dittrich and Dirk Jonscher. All together now : Towards integrating the world's information systems. In Carlos Delgado, Esperanza Marcos, and José Manuel Marqués Corral, editors, *JISBD*, page 7. Universidad de Valladolid, Departamento de Informática, 2000.

[EG89] Clarence A. Ellis and Simon J. Gibbs. Concurrency control in groupware systems. In *SIGMOD Conference*, volume 18, pages 399–407, 1989.

[ELLR90] A. Elmagarmid, Y. Leu, W. Litwin, and Marek Rusinkiewicz. A multidatabase transaction model for interbase. In *Proceedings of the VLDB conference*, pages 507–518, 1990.

[Elm92] Ahmed K. Elmagarmid. *Transaction Models for Advanced Database Applications*. Morgan-Kaufmann, 1992.

[FGHL06] Tom Freund, Alastair Green, John Harby, and Mark Little. Web services business activity (ws-businessactivity v1.1). March 2006.

[FHM05] Michael J. Franklin, Alon Y. Halevy, and David Maier. From databases to dataspaces : a new abstraction for information management. *SIGMOD Record*, 34(4) :27–33, 2005.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, 1992.

[GBC+01] Glaude Godart, Christophe Bouthier, Philippe Canalda, Francois Charoy, Pascal Molli, Olivier Perrin, Helene Saliou, Jean-Claude Bignon, Gilles Halin, and Olivier Malcurat. Asynchronous coordination of virtual teams in creative applications (co-design or co-engineering) requirements and design criteria. In M. Orlowska and M. Yoshikawa, editors, *Workshop on Information Technology for Virtual Enterprises (ITVE'01)*, volume 23, pages 135–142, Gold Coast, Queensland, Australia, 2001. IEEE Computer Society Press.

[GCG03] Daniela Grigori, François Charoy, and Claude Godart. Coo-flow : a process model to support cooperative processes. In *Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03)*, San Francisco, 2003.

[GCP+03] Claude Godart, François Charoy, Marc Patten, Nicolas Grégori, Jean-Charles Hautecouverture, and Isabelle Faugeras. Coopéra : apprendre à coopérer et apprendre en coopérant. In *Conférence Internationale sur l'Enseignement Ouvert et en Ligne (ICOOL'03)*, Ile Maurice, 2003.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[GMGK+91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Modeling long-running activities as nested sagas. *IEEE Data Eng. Bull.*, 14(1) :14–18, 1991.

[GMO+04] Glaude Godart, Pascal Molli, Gérald Oster, Olivier Perrin, Hala Skaf-Molli, Pradeep Ray, and Fethi Rabhi. The toxicfarm integrated cooperation framework for virtual teams. *Distributed and Parallel Databases*, 15(1) :67–88, 2004.

[GMS87] H. Garcia-Molina and K. Salem. Sagas. In Umeshwar Dayal and Irving L. Traiger, editors, *Proceedings of the SIGMD Conference, San Francisco, May 27-29, 1987*, pages 249–259. ACM, 1987.

[GPS99] Claude Godart, Olivier Perrin, and Hala Skaf. Coo : A workflow operator to improve cooperation modeling in virtual processes. In *RIDE*, pages 126–131, 1999.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing : Concepts and Techniques*. Morgan Kaufmann, 1993.

[Gri97] R. Grimes. *Professional DCOM Programming*. Wrox Press, 1997.

[GRST97] V.F. Gornev, V.B. Rarassov, R. Soenen, and K. Tahon. Virtual enterprise : Reasons, sources and tools. In *MCPL'97*, Campinas, Brésil, septembre 1997.

[GSBC00] Dimitrios Georgakopoulos, Hans Schuster, Donald Baker, and Andrzej Cichocki. Managing escalation of collaboration processes in crisis mitigation situations. In *16th International Conference on Data Engineering (ICDE 2000)*, pages 45–56, San Diego, California, USA, 2000.

[GSCB99] Dimitrios Georgakopoulos, Hans Shuster, Andrzej Cichocki, and Donald Baker. Managing process and service fusion in virtual enterprises. *Information systems*, 24(6) :429–456, 1999.

[HA00] Claus Hagen and Gustavo Alonso. Exception handling in workflow management systems. *IEEE Trans. Softw. Eng.*, 26(10) :943–958, 2000.

[Hal01] Alon Y. Halevy. Answering queries using views : A survey. *VLDB J.*, 10(4) :270–294, 2001.

[Hei95] Sandra Heiler. Sematic interoperability. *ACM Comput. Surv.*, 27(2) :271–273, 1995.

[HH02] Naoki Hayashi and George Herman. A coordination-theory approach to exploring process alternatives for designing differentiated products. In *Working Paper #218*. MIT Center for Coordination Science., 2002.

[HS97] Michael N. Huhns and Munindar P. Singh. Agents on the web : Ontologies for agents. *IEEE Internet Computing*, 1(6) :81–83, 1997.

[Hul08] Richard Hull. Artifact-centric business process models : Brief survey of research results and challenges. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (2)*, volume 5332 of *Lecture Notes in Computer Science*, pages 1152–1163. Springer, 2008.

[ICQ03] ICQ. http ://web.icq.com/, 2003.

[JB96] S. Jablonski and C. Bussler. *Workflow Management : Modeling Concepts, Architecture, and Implementation.* International Thomson Computer Press, 1996.

[KBRL04] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web services choreography description language version 1.0. http ://www.w3.org/TR/ws-cdl-10, October 2004.

[Kon93] Dimitri Konstantas. Object oriented interoperability. In Oscar Nierstrasz, editor, *ECOOP*, volume 707 of *Lecture Notes in Computer Science*, pages 80–102. Springer, 1993.

[KPP06] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Timed modelling and analysis in web service compositions. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES*, pages 840–846. IEEE Computer Society, 2006.

[KS95] Narayanan Krishnakumar and Amit P. Sheth. Managing heterogeneous multisystem tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2) :155–186, 1995.

[Kva00] Thomas Kvan. Collaborative design : what is it ? *Automation in construction*, 9(4) :409–415, 2000.

[Len96] R. Lendenmann. *Understanding OSF DCE 1.1 for AIX and OS/2.* Prentice Hall., 1996.

[Len02] Maurizio Lenzerini. Data integration : A theoretical perspective. In Lucian Popa, editor, *PODS*, pages 233–246. ACM, 2002.

[LK03] Frank Leymann and Rania Khalaf. On web services aggregation. In *4th VLDB Workshop on Technologies for E-Services (TES'03)*, Berlin, 2003.

[LZ04] Benchaphon Limthanmaphon and Yanchun Zhang. Web service composition transaction management. In Klaus-Dieter Schewe and Hugh E. Williams, editors, *ADC*, volume 27 of *CRPIT*, pages 171–179. Australian Computer Society, 2004.

[Man95] Frank Manola. Interoperability ussues in large-scale distributed object systems. *ACM Comput. Surv.*, 27(2) :268–270, 1995.

[Mar04] David Martin. Owl-s : Semantic markup for web services. *available from http ://www.w3.org/Submission/OWL-S/*, 2004.

[MC94] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1) :87–119, 1994.

[MDV+03] Robert McCann, AnHai Doan, Vanitha Varadarajan, Alexander Kramnik, and ChengXiang Zhai. Building data integration systems : A mass collaboration approach. In Vassilis Christophides and Juliana Freire, editors, *WebDB*, pages 25–30, 2003.

[MGB+08] Michael Mrissa, Chirine Ghedira, Djamal Benslimane, Zakaria Maamar, Florian Rosenberg, and Schahram Dustdar. A context-based mediation approach to compose semantic web services. *ACM Transactions on Internet Technology*, 8(1), February 2008.

[MHB06] R. Monson-Haefel and B. Burke. *Enterprise JavaBeans 3.0, Fifth Edition, Developing Enterprise Java Components*. O'Reilly, 2006.

[MM01] O. Marjanovic and Z. Milosevic. Towards formal modeling of e-contracts. In *5th IEEE International Enterprise Distributed Object Computing Conference*, 2001.

[MRB+92] Sharad Mehrotra, Rajeev Rastogi, Yuri Breitbart, Henry F. Korth, and Avi Silberschatz. The concurrency control problem in multidatabases : Characteristics and solutions. In Michael Stonebraker, editor, *ACM SIGMOD International Conference on Management of Data*, pages 288–297, San Diego, California, USA, 1992.

[MRKS92] Sharad Mehrotra, Rajeev Rastogi, Henry F. Korth, and Abraham Silberschatz. A transaction model for multidatabase systems. In *ICDCS*, pages 56–63, 1992.

[MRSK92] Sharad Mehrotra, Rajeev Rastogi, Avi Silberschatz, and Henry F. Korth. A transaction model for multidatabase systems. In *International Conference on Distributed Computing Systems (ICDCS'92)*, 1992.

[MSMO02] Pascal Molli, Hala Skaf-Molli, and Gérald Oster. Divergence awareness for virtuel team through the web. In *Sixth World Conference On Integrated Design and Process Technology (IDPT'02)*, Pasadena, CA, USA, 2002.

[Net02] NetMeeting. http ://www.microsoft.com/netmeeting, 2002.

[NRFJ07] Eric Newcomer, Ian Robinson, Max Feingold, and Ram Jeyaraman. Web services coordination (ws-coordination) version1.1. April 2007.

[NRLW06] Eric Newcomer, Ian Robinson, Mark Little, and Andrew Wilkinson. Web services atomic transaction (ws-atomictransaction 1.1) version 1.1. December 2006.

[OAS07] OASIS. Web services business process execution language (wsbpel). 2007.

[PA91] James M. Purtilo and Joanne M. Atlee. Module reuse by interface adaptation. *Softw., Pract. Exper.*, 21(6) :539–556, 1991.

[Pap03] Michael P. Papazoglou. Web services and business transactions. *World Wide Web*, 6(1) :49–91, 2003.

[PBCT07] Julien Ponge, Boualem Benatallah, Fabio Casati, and Farouk Toumani. Fine-grained compatibility and replaceability analysis of timed web service protocols.

In Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, editors, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 599–614. Springer, 2007.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10 :133–173, 2008.

[Pon06] J. Ponge. A new model for web services timed business protocols. In *Informatique des organisations et systèmes d'information et de décision (SIWS-Inforsid)*, 2006.

[Pur94] James M. Purtilo. The polylith software bus. *ACM Trans. Program. Lang. Syst.*, 16(1) :151–174, 1994.

[PWBG03] Olivier Perrin, Franck Wynen, Julia Bitcheva, and Glaude Godart. A model to support collaborative work in virtual enterprises. In *International Conference On Business Process Management (BPM'03)*, Eindhoven, The Netherlands, 2003. LNCS 2678.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4) :334–350, 2001.

[RLK06] Dumitru Roman, Holger Lausen, and Uwe(Ed.) Keller. D2v1.3. web service modeling ontology (wsmo). *available from http ://www.wsmo.org/TR/d2/v1.3/*, 2006.

[SD05] Benjamin A. Schmit and Schahram Dustdar. Systematic design of web service transactions. In Christoph Bussler and Ming-Chien Shan, editors, *TES*, volume 3811 of *LNCS*, pages 23–33. Springer, 2005.

[SK93] A. Sheth and V. Kashyap. *So Far (Schematically) yet So Near (Semantically).* Interoperable Database Systems, IFIP. D.K. Hsiao, E.J. Neuhold and R. Sacks-Davis (Editors). Elsevier Science B.V. (North-Holland), 1993.

[SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surveys*, 22(3) :183–236, 1990.

[SR93] A. P. Sheth and M. Rusinkiewicz. On transactional workflows. *Data Engineering Bulletin*, 16(2) :37–40, 1993.

[TN92] Ian Thomas and Brian A. Nejmeh. Definitions of tool integration for environments. *IEEE Software*, 9(2) :29–35, 1992.

[Tou05] F. Toumani. *Flexible techniques for querying, analyzing and manipulating web service abstractions* . PhD thesis, Université Blaise Pascal – Clermont-Ferrand II à Clermont-Ferrand, 8 Décembre 2005.

[VC] M. Valdez and F. Charoy. Bonita : Workflow cooperative system. Technical report, http ://bonita.objectweb.org.

[vdAtH00] W. M. P. van der Aalst and Arthur H. M. ter Hofstede. Verification of workflow task structures : A petri-net-baset approach. *Inf. Syst.*, 25(1) :43–69, 2000.

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1) :5–51, 2003.

[vdAvH02] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management : Models, Methods, and Systems*. MIT Press, Cambridge, 2002.

[Was89] Anthony I. Wasserman. Tool integration in software engineering environments. In Fred Long, editor, *SEE*, volume 467 of *Lecture Notes in Computer Science*, pages 137–149. Springer, 1989.

[Weg96] Peter Wegner. Interoperability. *ACM Comput. Surv.*, 28(1) :285–287, 1996.

[WFM95] WFMC. *Workflow Reference Model*. WFMC (Workflow Management Coalition), www.wfmc.org, January 1995.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3) :38–49, 1992.

[WMS02] Etienne Wenger, Richard McDermott, and William M. Snyder. *Cultivating Communities of Practice*. Harvard Business School Press, Boston, 2002.

[WR92] H. Watcher and A. Reuter. The contract model. pages 219–263, 1992.

[WRW] A. Wollrath, R. Riggs, and J. Waldo. A distributed object model for the java system. *Computing Systems*, 9 :265–290.

[WS97] D. Worah and A. P. Sheth. Transactions in transactional workflows. In *Advanced Transaction Models and Architectures*, pages 3–34. 1997.

[WWC92] Gio Wiederhold, Peter Wegner, and Stefano Ceri. Toward mega programming. *Communications of the ACM*, 35(11) :89–99, November 1992.

[WWRT91] Jack C. Wileden, Alexander L. Wolf, William R. Rosenblatt, and Peri L. Tarr. Specification-level interoperability. *Commun. ACM*, 34(5) :72–87, 1991.

[YS94] Daniel M. Yellin and Robert E. Strom. Interfaces, protocols, and the semi-automatic construction of software adaptors. In *OOPSLA*, pages 176–190, 1994.

[ZNBB94] Aidong Zhang, Marian Nodine, Bharat Bhargava, and Omran Bukhres. Ensuring relaxed atomicity for flexible transactions in multidatabase systems. pages 67–78, 1994.

Chapitre 7

Towards a Model for Persistent Data Integration

Auteurs : Olivier Perrin, Nacer Boudjlida

Référence : Fifth Conference on Advanced Information Systems Engineering, CAiSE'93, LNCS 685, 1993.

Abstract Tool integration in software development environments is a major problem, on one hand, for users of these environments, and on the other hand, for tool builders and suppliers. In this paper, we focus on persistent data integration, which is one of the two main points of tool integration. The purpose of this work is to provide a formal data model that includes most of the semantics of the data manipulated to answer problems raised by persistent data integration. To achieve this goal, we introduce a classical data model, and provide a set of operators which are given for the object transformation. We also provide an example describing the transformation process.

Towards a Model for Persistent Data Integration

Olivier Perrin and Nacer Boudjlida

Centre de Recherche en Informatique de Nancy (CRIN-CNRS)
University of Nancy I
Campus Scientifique — B.P. 239
54506 Vandoeuvre-lès-Nancy — FRANCE
email: operrin@loria.fr, nacer@loria.fr

Abstract. Tool integration in software development environments is a major problem, on one hand, for users of these environments, and on the other hand, for tool builders and suppliers. In this paper, we focus on persistent data integration, which is one of the two main points of tool integration. The purpose of this work is to provide a formal data model that includes most of the semantics of the data manipulated to answer problems raised by persistent data integration. To achieve this goal, we introduce a classical data model, and provide a set of operators which are given for the object transformation. We also provide an example describing the transformation process.

Keywords: data integration, integrated software-engineering environments, federated database management systems, meta-data, data heterogenity, PCTE, CDIF.

Introduction

Tool and data integration is a major problem for both users and suppliers of CASE tools. The information held by a tool is seldom compatible with the information that is required by another tool. Moreover, users want to control tools of an environment and need to be able to move information between these tools. The main reason is that users want to get efficient integrated systems.

The integration problem can be approached from two standpoints: the first is the control dimension, which means giving the ability for several tools to communicate. The second is the data dimension, which means enabling a tool to access data from another one. Data integration is relevant only when the tools deal with common data. In [17], there exists a summary of common questions to be answered in order to achieve data integration: "How work must be done to make the data used by one tool useful for the other ?", "How much data managed by a tool is duplicated in or can be derived from data managed by the other ?", "How well do two tools cooperate to maintain the semantic constraints on the data they process ?", "How much work must be done to make the data

generated by one tool usable by the other ?", "How well does a tool communicate changes it makes to the values of nonpersistent, common data so that other tools it is cooperating with may synchronize their values for the data ?". Data integration includes integration of persistent and nonpersistent data. We believe that nonpersistent data integration is relevant to control integration instead of data integration, so in this paper, we concentrate on persistent data integration. We focus on some interest in integrated software-engineering environments (ISEE)([5, 3]) and information systems environments, during both design and operating phases. Our work is also relevant to federated database management systems (FDBMS)([15, 12, 11]) and we examine both *a priori* and *a posteriori* views. Many problems about data exchange in SEE, or cooperating databases in FDBMS concern heterogeneity, i.e. differences in data semantics. Detecting semantic heterogeneity is a crucial problem. The main problems consists in having enough semantic and information to interpret data in a consistent way, and identifying and then solving semantic heterogeneity, such as differences in the definitions (meanings) of two data elements, or differences in the formats of the data elements (values, precision,...).

In [15], there is a list of unsolved problems in FDBMS. An important point deals with the identification and representation of all semantics useful in various FDMBS. This also exists in ISEE, where a various number of platforms and various tools use different formats. Another point deals with the automation, as completely as possible, of a process that transforms data to make a tool use data of another one. This gives rise to questions concerning semantic integrity constraints, serializability, concurrency control and management.

Persistent data integration is characterized by five points:

(*i*) the data being managed,
(*ii*) the representation and the naming of the data elements,
(*iii*) the semantic interpretation,
(*iv*) the syntactic and semantic constraints,
(*v*) the implementation.

The data represents data that each environment's tool can access. It is the "work area" we deal with. The implementation represents file or object structures, concurrency control mechanisms and global environment integrity. Representation and semantic interpretation could be split into four different parts, which can be summarized as follows:

- identical pieces of information are represented by different symbols, which makes up the **synomyms** problem,
- different pieces of information are represented by the same symbol, which makes up the **homonyms** problem,
- different aggregation or scale levels,
- loss of information.

Then, syntactic and semantic constraints are grouped into constraints that are used to manage and process data.

The purpose of this work is to provide a formal data model that includes most of the semantics of the data used by one tool to enable another tool to "understand" these data. The formal model uses the notion of meta-level, commonly used in recent databases or repositories descriptions, and proposes to group attributes which describe the same data. Different levels of compatibility between two definitions of a data description are defined. The model has commonalities with object oriented data models. Its description is split into two parts: the first section is devoted to the introduction of definitions and propositions about compatibility of two data objects. The second section describes the operators of the model, that are given for the object transformation. A third section will show an example to illustrate the mechanisms. Then, we conclude by providing some problems not yet solved by the model.

1 The Data Model

In this section, we describe the model we propose to use to deal with persistent data integration. The model encompasses classical concepts to describe "flat" structures as well as composite structures. It also encompasses an object set concept that enables grouping objects with "similar" descriptions into a set. Finally, a role concept is introduced to include the fact that an element of two different objects has the same meaning. These concepts enable us to define three kinds of object compatibility. Furthermore, a notion of hierarchy of object definitions is introduced. It enables ordering of object definitions on the "goodness of their descriptions" and deducing object definition compatibilities. A feature of object models is their richer semantics. This is important to get enough semantics to provide as much automation as possible. We believe that such a model allows for two tools to have information about *syntax* and *semantic* of shared data. It is the basics of our method to provide a tool with a mechanism to understand and exchange data of another tool of the same class (i.e., tools which share some functionalities and data).

1.1 Objects

We assume the following sets:

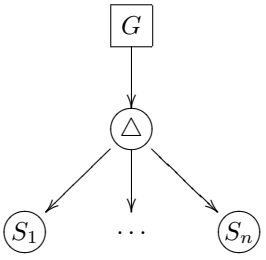
- let $\mathcal{S}$ be an infinite set of symbols. This set is partitioned into disjoint subsets $\mathcal{S}_r$. These subsets contain symbols with identical role and meaning. Let $\mathcal{S}_r \subset \mathcal{S}$ be such a subset where all the symbols belong to the role r. A data element $\ell_r \in \mathcal{S}_r$ is a data element of r role.
  For example, the following elements belong to $\mathcal{S}$: *age, position, comment.*
- let $\mathcal{T}$ be the atomic type set (for example, integer, string, boolean, real). We use usual standard behaviors for these atomic types. We consider that the equality between two types is a syntactic equality.
- let $\mathcal{V}$ be the set of values belonging to $\mathcal{T}$, denoted $\{\mathcal{V}_\mathtt{t} \mid \mathtt{t} \in \mathcal{T}\}$. We assign a specific value to each type $\mathtt{t} \in \mathcal{T}$, the null value, denoted $\mathtt{null}_\mathtt{t}$

which describes the null value of type $\mathtt{t} \in \mathcal{T}$ (in our approach, null values are non-informative values) . Each type has a corresponding domain, namely $\mathcal{D}_{integer}$, $\mathcal{D}_{string}$, $\mathcal{D}_{boolean}$, and $\mathcal{D}_{real}$.

The definitions we give below are very similar to the definitions which are admitted in an object-oriented data model or a the relational data model with nested relations ([10, 1]). Two concepts are commonly used for object definition. The first one is a "flat" structure where all the attributes are defined by an atomic type. The second one introduces composite structures where an attribute can be defined by an object definition. To introduce complex object definition, we classicaly use three type constructors given in [16] and [2]. These are commonly accepted as the tuple constructor, the union constructor, and the set constructor.

Tuple constructor is closely related to the record structure in PASCAL, and to the aggregation described in [16]. This constructor allows to construct more complicated types, and object definitions. The syntactic representation for the tuple constructor is given by a pair of square brackets ([...]). Following is a short example of such a complex object definition. A complex object definition could be [*Name*: string; *Address*:[*Number*: integer; *Street*: string; *City*: string]]. An instance of this object definition would be [*Name* = `"Martin"`; *Address* =[*number* = 9; *Street* = `"Harbor Drive"`; *City* = `"San Diego"`].

The union of type constructors is closely related to PASCAL's record variant structure, and to generalization of [16]. It is used so that objects of differents types can be view as generically the same. The syntactic representation for the union of a type constructor is given by a pair of angle brackets ($\langle \ldots \rangle$). We now give an example: an object definition could be [*Address*: $\langle$*Street Address*: [*Number*: integer; *Street*: string]; *PO-BOX*: string$\rangle$], and an instance would be [*Address* = $\langle$*PO-BOX* = `"400"`$\rangle$]. It is important to notice that we impose two constraints to this constructor. The first one is called the disjoint property and means that the intersection of all subtypes of a given master type is empty. The second one, called the total property, means that the union of all subtypes of a given master type recovers this type. Formally speaking, we define the disjoint and total concepts as follows. Let's consider the following diagram:



where $\triangle$ represents the union of types. The disjoint property means that $S_i \cap S_j = \emptyset$ for i $\neq$ j. The total property means that $\bigcup_{i=1}^{n} S_i = G$.

The third constructor is a set constructor, called the collection constructor. It allows to regroup a set of objects belonging to a given type. The syntactic representation of this constructor is given by a pair of curled brackets ($\{\ldots\}$). In

the following example, we describe the semantic corresponding to this constructor. Suppose we have a text block formed by a set of lines. We write [*Text_Block*: {*Line*: string}]. An instance of this object definition would be [*Text_Block* = {"This is a", "simple example", "of the set constructor"}].

In [9], we could find an extended version of the model, described as the Format model, which subsumes the relational model and some parts of the hierarchical model. It also provides a convenient way to represent types and objects by trees and subtrees.

Definition 1 *An object definition F, either atomic object definition or complex object definition, denoted* $F = [\ell_{r_1} : \mathtt{t}_1; \ldots; \ell_{r_n} : \mathtt{t}_n]$ *(*$n \geq 1$*), is recursively constructed as follows:*

- $\ell_{r_i} \in \mathcal{S}_{r_i}$*,* $(1 \leq i \leq n)$*,*
- $\mathtt{t}_i \in \mathcal{T}$ *(*$\mathtt{t}_i$ *is an atomic type) or*
  $\mathtt{t}_i$ *is a complex type (constructed by recursive application of the aggregation, generalization or collection constructors),* $(1 \leq i \leq n)$*.*

In the rest of the text, we will call a simple object an object defined by an atomic definition and a composite object an object defined by a complex definition. Example 1 shows two simple objects.

Definition 2 *An object, defined by F and denoted* o_F*, is given by* $[\ell_{r_1} = v_1; \ldots; \ell_{r_n} = v_n]$ *(*$n \geq 1$*) where:*

- $\ell_{r_i} \in \mathcal{S}_{r_i}$*,* $(1 \leq i \leq n)$*,*
- $v_i \in \mathcal{V}_{\mathtt{t}_i}$ *or* $v_i = \mathtt{null}_{\mathtt{t}_i}$*,* $(1 \leq i \leq n)$*.*

1.2 Object set

In this section, we introduce the object set concept. Such a set contains two components which are a unique object definition, and an object set where objects are consistent with the set definition. Each object of an object set correspond to the given definition.

Definition 3 *An object set, denoted* D_F*, is defined as:*

$D_F = \{F, o_1, \ldots, o_n\}$ *where every* o_i *is an object defined by F (*$i \leq n$*, F unique).*

The object set represents data to be managed. It is an ensemble of instances refering to the definition of the object set. This definition takes advantage of the separation between the definition of the objects and the entities themselves. This approach is widely used in object oriented databases and recent database models, where a meta-level is introduced to give more information about data. It represents a dictionary of object descriptions. For our purpose, it provides two distinct levels with distinct goals: the first level gives semantic of data, while the second concerns object's instances.

Example 1 Let $F = [text$: string;$xposition$: integer; $yposition$: integer] be an object definition. o_1 and o_2 are two objects defined by F:

o_1 = [$text$ = "sample"; $xposition$= 150; $yposition$ = 150],
o_2 = [$text$ = "longer one"; $xposition$ = 0; $yposition$ = 0].

These two objects are considered as simple objects. A more complex object definition could be given by:

$$F'=[Position: [x:\text{integer}; y: \text{integer}]; Text\_block:\{Line: \text{string}\}]$$

An instance of F' (considered as a composite object), denoted o_3 is defined as:

o_3=[$Position$=[x=150;y=150];$Text\_block$={$Line$="A much more";$Line$="complex example"}]

This way to describe objects is very clear and self explanatory. However, it becomes complicated and too cumbersome when an object contains a lot of elements. So we suggest to shorten the object writing as follows:

o_3=[[150;150];{"A much more";"complex example"}]

Nevertheless, we notice the importance of the description and the meaning of each symbol of an object o. It is important to know that the first value 150 refers to the x position, and value [150,150] refers to *Position*. So we must keep in mind that semantic is related to a symbol defined in an object. There are two issues: first, to keep the symbol's semantic in the object itself, second, to provide the F definition outside the object o, and for all objects refering to the definition. This is used for the object set concept.

1.3 Domain and degree

We will define a way to get all the symbols existing in a given object definition. Next, we will describe the way to access object's values.

Definition 4 *We assume an object definition F. We write* $\mathbf{Dom}(F)$ *the set of all the symbols of the definition F, and we recursively define* $\mathbf{Dom}(F)$ *as:*

- *if* $F = [\ell_{r_1}.\mathtt{t}_1]$, *then* $\mathbf{Dom}(F) = \{\ell_{r_1}\}$,
- *if* $F = [\ell_{r_1}.\mathtt{t}_1;\ldots;\ell_{r_n}.\mathtt{t}_n]$,
  or $F = \{\ell_{r_1}.\mathtt{t}_1;\ldots;\ell_{r_n}.\mathtt{t}_n\}$,
  or $F = \langle\ell_{r_1}.\mathtt{t}_1;\ldots;\ell_{r_n}.\mathtt{t}_n\rangle$, *then* $\mathbf{Dom}(F) = \{\ell_{r_i} \in F,\ 1 \leq i \leq n\}$

We notice that $\mathbf{Dom}(F)$ *and* $\mathbf{Dom}(o_F)$ *contain the same elements.*

We now assume an instance of F given by the object o_F. Then we write $o_F(\ell_r)$ the corresponding value of ℓ_r.

Example 2 If we consider the object o_1 of the first example, we have:
- $\mathrm{Dom}(o_1) = \{text,\ xposition,\ yposition\}$,
- $o_1(text) =$ `"sample"`,
- $o_1(xposition) =$ `150` and $o_1(yposition) =$ `150`.

For the o_3 object, we write:

- $\mathrm{Dom}(o_3) = \{Position,\ x,\ y,\ Text\_block,\ Line\}$,
- $o_3(Position) =$ `[150;150]`,
- $o_3(Position(x)) =$ `150`,
- $o_3(Text\_block) =$ {`"A much more","complex example"`}.

Notice that it is not possible to access a particular element of a set.

Definition 5 *Given o_F an object defined by F, we call* `degree` *of o_F, denoted* `deg`*(o_F) or #Dom(F), the cardinal of o_F, i.e. the cardinal of domain* **Dom(*F*)**.

Example 3
- `deg`$(o_1) = 3$.
- `deg`$(F') = 5$.

1.4 Roles

We have seen the importance of the semantical information of each part of an object. We describe in this section another concept: the role. Intuitively, it specifies that an element has the same meaning in two different definitions. For instance, consider two object definitions $F_1 = [color$: string; x: integer; y: integer; $text$: string$]$ and $F_2 = [description$: string; $xpos$: integer; $ypos$: integer$]$. The two definitions share some parts devoted to the same "role": *text* and *description*, x and *xpos*, and y and *ypos*. The role concept means that same semantic interpretation is related to a data element in two different definitions. Furthermore, the definition of F_2 is included in the definition of F_1 (what could be written $\mathrm{Dom}(F_2) \subseteq \mathrm{Dom}(F_1)$). Consequently, a transformation can make F_2 compatible with F_1 (and conversely if we authorize some loss of information). There exists a way to establishe this kind of correspondence among data element's roles.

The data model we propose relies on the role concept. The symbol set $\mathcal{S}$ is divided into subsets as shown in the definition part. For instance, we regroup in the $\mathcal{S}_r$ subset all the symbols of the role r in the set $\mathcal{S}$. We distinguish between:

- **the role concept**: it consists of a set of data elements that share some properties (basically at the semantic level, i.e., identical meaning),
- **a data type**: it consists of a set of data elements with operations associated with,
- **a value**: value of a data element of role r and of type `t`.

There are at least two distinct type definitions:

- types as sets [4],
- types as algebras [7, 8].

The first definition has a historical origin, while the second appeared around 1975. Definition of role is represented by the notion of "types as sets", which serves to classify data elements whereas our notion of type describes the representation of a data element and operators among them. The notion of role seems to approach the notion of abstract data types where some semantic is related to the types. Roles are classes of data sets, while types are complex and include operators.

The next definition proposes to represent the concept in the object model, using the object's definition. We first introduce the binary predicate symbol $\overset{sem}{=}$ which can be interpreted as follows: $\ell_r \overset{sem}{=} \jmath_{r'}$ holds iff r is semantically identical to r', which stands for semantic equality between two elements of the set $\mathcal{S}$.

Definition 6 *Let F_1 and F_2 be two object definitions. We suppose given $\ell_r \in Dom(F_1)$, and $\jmath_{r'} \in Dom(F_2)$. We say that ℓ_r et $\jmath_{r'}$ are compatible, denoted $\ell_r \equiv \jmath_{r'}$, if and only if:*

$$r \overset{sem}{=} r'$$

Compatibility is not restricted to a one-to-one correspondence between two elements (or symbols) of two definitions. We can extend the binary relation $\equiv$ by providing a new context. For instance, such a representation can exist:

$$\ell_r \equiv \bigcup_x (\jmath_{r_x}), \text{ iff } r \overset{sem}{=} r' \text{ with } r' = \bigcup(r_i, \ldots, r_j) \ (i \leq x \leq j)$$

With such a representation, we can introduce the composition of symbols. There is no constraint on types, the only conditions we retain are the description and the role. Exemaple 4 illustrates definition 6.

Definition 7 *Consider $F = [\ell_{r_1} : \mathtt{t}_1; \ldots; \ell_{r_n} : \mathtt{t}_n]$ $(n \geq 1)$ and an associated object $o_F = [\ell_{r_1} = v_1; \ldots; \ell_{r_n} = v_n]$, we define ℓ_{r_m} $(m > n)$ as follows:*

- $\ell_{r_m} \equiv \mathtt{null}_{\mathcal{N}_{r_m}}$, $(m > n)$,
- $F(\ell_{r_m}) = \mathtt{undefined}_{\mathsf{Type}}$,
- $o(\ell_{r_m}) = \mathtt{null}_{F(\ell_{r_m})}$.

Various $\mathtt{null}_{\mathcal{N}_{r_m}}$, $\mathtt{undefined}_{\mathsf{Type}}$ or $\mathtt{null}_{F(\ell_m)}$ represents an undefined role, an undefined type or an undefined value (e.g., `null` value). There are undefined values which could be used to inform or complete a definition. Definition 9 allows to extend a given definition to match another one, and to provide a way to get a one-to-one correspondence between each symbol of the two definitions. In such a situation, we allow for the completion and the extension of a definition F of an object o_F, only with null values. For example, $F_1 = [$*color*: string; *x*: integer; *y*: integer; *text*: string$]$ and $F_2 = [$*description*: string; *xpos*: integer; *ypos*: integer$]$ match if we rewrite F_2 into:

Example 4 Let us analyze the objects o_1 and o_2 used in example 1. The following properties hold:
- $text \equiv description$,
- $x \equiv xpos$,
- $y \equiv ypos$.

Consider another example, where $F_1 = [Name$:string;$FirstName$:string] and $F_2 = [FullName$:string], then the following property holds:

$$FullName \equiv (Name,\ FirstName)$$

Then, if we apply the definition 8 to $F_1 = [BirthDate$:date] and $F_2 = [BirthDate$:string], we get:

$$BirthDate_{F_1} \equiv BirthDate_{F_2}$$

$$F_2 = [color\text{: } \mathtt{null}_{\mathsf{string}};\ description\text{: string};\ xpos\text{: integer};\ ypos\text{: integer}]$$

Concerning compatibility, the next four properties are used to describe multiple levels of compatibility between two object definitions. We introduce three levels: general, partial and total compatibility (propositions 1 to 3 below). Proposition 4 defines all the authorized states when two definitions are compatible. This property helps us later obtain all the operators given by the model.

In order to define general compatibility, we do not make any assumption on the types of the data elements of the two definitions F_1 and F_2.

Proposition 1 (General compatibility of definitions)
Given two object definitions F_1 and F_2, F_1 and F_2 are generally compatible, denoted $F_1 \sim F_2$, iff:

- $Dom(F_1) \cap Dom(F_2) \neq \emptyset$,
- $\forall \ell_r \in Dom(F_1) \cap Dom(F_2)$, $\exists \jmath_r \in Dom(F_2)$ *(respectively* $\bigcup_x (\jmath_{r_x})$*) such that* $\ell_r \equiv \jmath_r$ *(respectively* $\ell_r \equiv \bigcup_x (\jmath_{r_x})$*).*

□

In the next proposition, we suppose that every type of common elements are syntacticaly equal, and we introduce the partial compatibility of two given object definitions. Therefore, we proceed to define equality between two types. We write that $\mathtt{t} = \mathtt{t'}$ iff $\mathtt{t}$ is syntactically equal to $\mathtt{t'}$. We use the general compatibility proposition to enforce the compatibility of various elements of the definitions.

Proposition 2 (Partial compatibility of definitions)
Given two object definitions F_1 and F_2, F_1 and F_2 are partially compatible, denoted $F_1 \simeq F_2$, iff:

- $F_1 \sim F_2$,

– $F_1(\ell_r) = F_2(\jmath_r)$ *(or* $F_1(\ell_r) = \bigcup_x F_2(\jmath_{r_x})$*), what can be written* $\mathtt{t}_{\ell_r} = \mathtt{t}_{\jmath_r}$ *(or* $\mathtt{t}_{\ell_r} = \mathtt{t}_{\jmath_{r_x}}$ *foreach* x *such that* $\ell_r \equiv \bigcup_x (\jmath_{r_x})$*).*

□

Then, we characterize the total compatibility property, which is the best achievable property about compatibility of two object definitions. There exists a one-to-one correspondence between each element of the definitions.

Proposition 3 (Total compatibility of definitions)
If we consider two object definitions F_1 *and* F_2*,* F_1 *and* F_2 *are totally compatible, denoted* $F_1 \approx F_2$*, iff:*

– $Dom(F_1) = Dom(F_2)$,
– $\forall \ell_r \in Dom(F_1)$ *and* $\jmath_{r'} \in Dom(F_2)$*, we have* $\ell_r \equiv \jmath_{r'}$ *and* $F_1(\ell_r) = F_2(\jmath_{r'})$*, i.e.,* $\mathtt{t}_{\ell_r} = \mathtt{t}_{\jmath_{r'}}$.

□

Our goal is to provide mechanisms to get a complete compatibility between two object definitions F_1 and F_2, in order to transform objects in regard to their definitions and their compatibility. Our method to build such a transformation proceeds step by step, to finally obtain a complete compatibility between two definitions that represent the same data. In order to make an inventory of all existing cases, we introduce proposition 4. We also provide an example which handle all the existing cases described in the proposition.

Proposition 4
Given two object definitions respectively defined as $F_1 = [\ell_{r_1} : \mathtt{t}_1; \ldots; \ell_{r_n} : \mathtt{t}_n]$ *(*$n \geq 1$*) and* $F_2 = [\jmath_{r_1} : \mathtt{t}'_1; \ldots; \jmath_{r_m} : \mathtt{t}'_m]$ *(*$m \geq 1$*). If* $\ell_{r_i} \in Dom(F_1)$ *and* $\jmath_{r_j} \in Dom(F_2)$*, and* $\ell_{r_i} \equiv \jmath_{r_j}$*, we can say that one of these statements is satisfied:*

1) $F_1(\ell_{r_i}) = F_2(\jmath_{r_j})$ *(types are syntactically equals),*
2) $F_1(\ell_{r_i}) \neq F_2(\jmath_{r_j})$ *(types are not equals),*
3) $F_1(\ell_{r_i}) = \mathtt{undefined}_{\mathsf{Type}}$ *(the element does not exist),*
4) $F_2(\jmath_{r_i}) = \mathtt{undefined}_{\mathsf{Type}}$ *(the element does not exist),*
5) $F_1(\ell_{r_i}) = \bigcup_x F_2(\jmath_{r_x})$ *(composition of several elements),*
6) $F_2(\jmath_{r_i}) = \bigcup_x F_1(\ell_{r_x})$ *(composition of several elements).*

□

Example 5
1) F_1 = [*date*: string] and F_2 = [*date*: string],
2) F_1 = [*length*: string] and F_2 = [*length*: integer],
3) F_1 = [*length*: string; *checked*: boolean] and F_2 = [*length*: string], (cases 3 and 4)
4) F_1 = [*day*: string; *month*: string; *year*: string] and F_2 = [*date*: string], (cases 5 and 6)

1.5 Hierarchy

Intuitively, the idea that elements of a definition are ordered in terms of their "goodness of descriptions" is already quoted in [13]. But what does it mean? We think that two definitions could be ordered, and therefore, we can deduce when a definition can be compatible with another one.

Let $\mathcal{E}$ be a set, and $\preceq$ an order on $\mathcal{E}$ (we call $(\mathcal{E},\preceq)$ a preordered set). Two elements $x,y \in \mathcal{E}$ are consistent if there exists $z \in \mathcal{E}$ such that $x \preceq z$ and $y \preceq z$. We use such a structure $(\mathcal{E}, \preceq)$ to represent the notion of inclusion of a definition in another one. Consider two definitions F_1 and F_2. We say that $F_1 \preceq F_2$ iff F_1 is included in F_2.

Proposition 5 (Ordering on flat definition)
The information ordering $\preceq$ on flat definition types is the simplest relation satisfying:

$$[\ell_{r_1} : \mathsf{t}_1; \ldots; \ell_{r_n} : \mathsf{t}_n] \preceq [\ell_{r_1} : \mathsf{t}_1; \ldots; \ell_{r_n} : \mathsf{t}_n; \ldots; \ell_{r_m} : \mathsf{t}_m].$$

□

If we consider F_1=[*text*: string] and F_2=[*text*: string; *xpos*: integer; *ypos*: integer], we say that the structure represented by F_2 "contains" the structure represented by F_1. This intuitive notion is now formalized by a partial order.

Proposition 6
Given F_1=$[\ell_{r_1} : \mathsf{t}_1; \ldots; \ell_{r_n} : \mathsf{t}_n]$ and F_2=$[\ell_{r_1} : \mathsf{t}_1; \ldots; \ell_{r_n} : \mathsf{t}_n; \ldots; \ell_{r_m} : \mathsf{t}_m]$. F_1 and F_2 satisfy:

$$\begin{cases} F_1 \preceq F_2 \\ \\ F_1 \simeq F_2 \end{cases}$$

□

Then, we can deduce from definition 9 and proposition 6 that, considering $F_1 = [\ell_{r_1} : \mathsf{t}_1; \ldots; \ell_{r_n} : \mathsf{t}_n]$ and $F_2 = [\jmath_{r_1} : \mathsf{t}'_1; \ldots; \jmath_{r_n} : \mathsf{t}'_n]$, we have:

$$\begin{array}{c} F_1 \preceq F_2 \text{ iff } \mathsf{t}_n = \mathtt{undefined}_{\mathsf{Type}} \text{ or} \\ \ell_{r_n} \equiv \jmath_{r_n} \text{ or} \\ \ell_{r_n} \equiv \bigcup_x (\jmath_{r_x}) \end{array}$$

We now have an order for every object definition, and can interpret this ordering of "goodness of description". For example, F_1 = [*text*: string ; *xpos*: $\mathtt{null}_{\mathsf{integer}}$; *ypos*: $\mathtt{null}_{\mathsf{integer}}$] and F_2 = [*text*: string; *xpos*: integer; *ypos*: integer] can be ordered as follows:

$$F_1 \preceq F_2$$

Moreover, we have:

$$F'_1 = [\textit{text}: \mathsf{string}] \preceq F_2$$

2 Operators

In the previous section, we described the concepts of the data model. However, we believe a data model is not sufficient to allow for data exchange and data sharing. It only enables object description. To make a tool use the objects of another tool, it is sometimes necessary to transform the object of the latter, i.e. to make the objects fully compatible. In this framework, the data model is provided with a set of atomic operators that are used to make totally compatible two generally or partially compatible object definitions. Thus, the transformation cycle consists in:

- analyzing the level of compatibility of the two object definitions,
- making totally compatible the two object definitions,
- transforming the object instances.

We introduce in this section rewritten expressions of definitions. It allows for the restructuration both of the definitions and the instances. Each rewritten expression is an "atomic" rewritten expression. It means that a complete transformation can be represented by a set of rewritten expressions, and that each step of a transformation is represented by such an expression. This view allows us to provide clear and simple rewritten rules, with no huge complexity. It also makes the study of properties attached to expressions easier, for instance composition support.

We first present rewritten rules that define accepted transformations upon definitions. These rules are divided into two parts: structural rewritten rules and content rewritten rules. Then, we introduce operators derived from these rules.

2.1 Structural Rewritten Rules

We discuss here rewritten expressions acting on structures. We have to take into account that it should be possible to transform instances. So we establish a difference between "structural" rewritten expressions and "content" rewritten expressions.

We first present what we call Structural Rewrite Rules (SRR). At this level, we only show how such a rule acts. We do not provide any information on how it is possible to effectively transform structures and instances. Some of these SRR are already quoted in [2].

Rule 1 Given the following definition:
$[\ell_1\colon \mathtt{t}_1;\ \ldots;\ \ell_{i-1}\colon \mathtt{t}_{i-1};\ \ell_i\colon[\ell_k\colon \mathtt{t}_k;\ \ldots;\ \ell_{k+m}\colon \mathtt{t}_{k+m}];\ \ell_{i+1}\colon \mathtt{t}_{i+1};\ \ldots;\ \ell_n\colon \mathtt{t}_n]$
It could be rewritten as:
$[\ell_1\colon \mathtt{t}_1;\ \ldots;\ \ell_{i-1}\colon \mathtt{t}_{i-1};\ \ell_k\colon \mathtt{t}_k;\ \ldots;\ \ell_{k+m}\colon \mathtt{t}_{k+m};\ \ell_{i+1}\colon \mathtt{t}_{i+1};\ \ldots;\ \ell_n\colon \mathtt{t}_n]$

Rule 2 Given the following definition:
$[\langle\ell_1\colon \mathtt{t}_1;\ \ldots;\ \ell_{i-1}\colon \mathtt{t}_{i-1};\ \ell_i\colon\langle\ell_k\colon \mathtt{t}_k;\ \ldots;\ \ell_{k+m}\colon \mathtt{t}_{k+m}\rangle;\ \ell_{i+1}\colon \mathtt{t}_{i+1};\ \ldots;\ \ell_n\colon \mathtt{t}_n\rangle]$
It could be rewritten as:
$[\langle\ell_1\colon \mathtt{t}_1;\ \ldots;\ \ell_{i-1}\colon \mathtt{t}_{i-1};\ \ell_k\colon \mathtt{t}_k;\ \ldots;\ell_{k+m}\colon \mathtt{t}_{k+m};\ \ell_{i+1}\colon \mathtt{t}_{i+1};\ \ldots;\ \ell_n\colon \mathtt{t}_n\rangle]$

Rule 3 Consider the following definition:
$[\ell_1\colon \mathsf{t}_1; \ldots; \ell_i\colon\langle\ell_k\colon \mathsf{t}_k; \ldots; \ell_{k+m}\colon \mathsf{t}_{k+m}\rangle; \ldots; \ell_n\colon \mathsf{t}_n]$
It could be rewritten as:
$[\langle[\ell_1\colon \mathsf{t}_1; \ldots; \ell_k\colon \mathsf{t}_k; \ldots; \ell_n\colon \mathsf{t}_n]; \ldots; [\ell_1\colon \mathsf{t}_1; \ldots; \ell_{k+n}\colon \mathsf{t}_{k+n}; \ldots; \ell_n\colon \mathsf{t}_n]\rangle]$

Rule 4 Consider the definition:
$[\{\langle\ell_1\colon \mathsf{t}_1; \ldots; \ell_n\colon \mathsf{t}_n\ \rangle\}]$
It could be rewritten as:
$[[\{\ell_1\colon \mathsf{t}_1\ \}; \ldots; \{\ell_n\colon \mathsf{t}_n\}]]$

Rule 5 Consider the definition:
$[\ell_1\colon \mathsf{t}_1; \ldots; \ell_k\colon \mathsf{t}_k; \ldots; \ell_n\colon \mathsf{t}_n]$
It could be rewritten as:
$[\ell_1\colon \mathsf{t}_1; \ldots; \jmath_k\colon \mathsf{t}_k; \ldots; \ell_n\colon \mathsf{t}_n]$

Properties As underlined above, these rules are atomic. The following example shows how we can combine some of these rules. Suppose the two following definitions: $F = [\mathit{BirthDate}\colon \mathsf{date}]$ and $F' = [\mathit{Birth}\colon[\mathit{day}\colon \mathsf{integer};\ \mathit{month}\colon \mathsf{integer};\ \mathit{year}\colon \mathsf{integer}]]$. We apply the following rules:

$\phi = \text{rewrite}(\mathit{date}\colon \mathsf{date} \rightarrow \mathit{date}\colon \mathsf{integer})$
$\psi = \text{rewrite}(\mathit{date}\colon \mathsf{integer} \rightarrow \mathit{date}\colon[\mathit{day}\colon \mathsf{integer};\ \mathit{month}\colon \mathsf{integer};\ \mathit{year}\colon \mathsf{integer}])$
$\xi = \text{rewrite}(\mathit{BirthDate} \rightarrow \mathit{Birth})$

and deduce:
$F' = \xi(\psi(\phi(F)))$ and $F = \xi^{-1}(\psi^{-1}(\phi^{-1}(F')))$

2.2 Content Rewritten Rules

We now present some rewritten rules, called Content Rewrite Rules (CRR), that are necessary to take into account some differences based on types, degrees or scale expression of attributes.

Rule 6 Consider the following definition:
$[\ell_1\colon \mathsf{t}_1; \ldots; \ell_k\colon \mathsf{t}_k; \ldots; \ell_n\colon \mathsf{t}_n]$
It could be rewritten as:
$[\ell_1\colon \mathsf{t}_1; \ldots; \ell_k\colon \mathsf{t}'_k; \ldots; \ell_n\colon \mathsf{t}_n]$

Rule 7 Consider the following definition:
$[\ell_1\colon \mathsf{t}_1; \ldots; \ell_k\colon \mathsf{t}_k; \ldots; \ell_n\colon \mathsf{t}_n]$
It could be replaced by:
$[\ell_1\colon \mathsf{t}_1; \ldots; \ell_k\colon \mathsf{t}'_k; \ldots; \ell_n\colon \mathsf{t}_n; \ell_{n+1}\colon \mathsf{t}_{n+1}]$

Rule 8 Consider the following definition:

$[\ell_1 \colon \mathsf{t}_1; \ldots; \ell_k \colon \mathsf{t}_k; \ldots; \ell_{k+1} \colon \mathsf{t}_{k+1}; \ldots; \ell_n \colon \mathsf{t}_n]$

If ℓ_k and ℓ_{k+1} are linked by a calculus relation, named ϕ, then we could rewrite the definition as:

$[\ell_1 \colon \mathsf{t}_1; \ldots; \ell_k \colon \mathsf{t}_k; \ldots; \phi(\ell_k) \colon \mathsf{t}_{k+1}; \ldots; \ell_n \colon \mathsf{t}_n]$

Rule 9 Assume the following definition:

$[\ell_1 \colon \mathsf{t}_1; \ldots; \ell_k \colon \mathsf{t}_k; \ldots; \ell_n \colon \mathsf{t}_n]$

It could be rewritten as:

$[\ell_1 \colon \mathsf{t}_1; \ldots; \phi(\ell_k) \colon \mathsf{t}_k; \ldots; \ell_n \colon \mathsf{t}_n]$

Rule 10 Assume the following definition:

$[\ell_1 \colon \mathsf{t}_1; \ldots; \ell_{k-2} \colon \mathsf{t}_{k-2}; \ell_{k-1} \colon \mathsf{t}_{k-1}; \ell_k \colon \mathsf{t}_k; \ell_{k+1} \colon \mathsf{t}_{k+1}; \ldots; \ell_n \colon \mathsf{t}_n]$

It could be rewritten as:

$[\ell_1 \colon \mathsf{t}_1; \ldots; \ell_{k-2} \colon \mathsf{t}_{k-2}; \ell'_k \colon \mathsf{t}'_k; \ell_{k+1} \colon \mathsf{t}_{k+1}; \ldots; \ell_n \colon \mathsf{t}_n]$

2.3 The operators

We introduce in this section ten operators derived from the rules previously defined. For each operator, we present the rule it corresponds to, and a brief overview.

FOLD and UNFOLD operators

These two operators implement the first rule. The rule 1 says that if there exists a nested construction, then this construction can be "flattened". Conversely, a flat structure can be rewrite to become a nested structure. So these two operators act upon structures.

Example: A typical example of use of such operators is based upon names. Let's assume $F = [\mathit{Name} \colon [\mathit{LastName} \colon \mathsf{string}; \mathit{FirstName} \colon \mathsf{string}]]$. Then, we say that the UNFOLD operator transforms F into F' as:

$$F' = \textsc{unfold}(F, \mathit{Name}) = [\mathit{LastName} \colon \mathsf{string}; \mathit{FirstName} \colon \mathsf{string}]$$

COMPOSE and UNCOMPOSE operators

Rule 2 says that it is possible to "uncompose" the union of attributes already enclosed into a union. Conversely, it says that it is possible to compose several attributes that can be view generically the same in an union. This is the goal of the COMPOSE and UNCOMPOSE operators.

Example: Given $F = [\mathit{Locality} \colon \langle \mathit{BirthCity} \colon \mathsf{string}; \mathit{BirthCountry} \colon \mathsf{string} \rangle]$, the UNCOMPOSE operator transforms F into F' as:

$$F' = \textsc{uncompose}(F, \mathit{Locality}) = [\mathit{Locality} \colon \mathsf{string}]$$

DISTRIBUTE and UNDISTRIBUTE operators
The third rule says that it is possible to "rise" the union constructor through a definition. The application of this rule gives the DISTRIBUTE and UNDISTRIBUTE operators. Each attribute of a union is "distribute" through the given definition.

Example: Given $F = [Name: \mathsf{string};\ BirthDate: \mathsf{date};\ Locality: \langle BirthCity: \mathsf{string};\ BirthCountry: \mathsf{string}\rangle]$, the definition can be transformed into a definition named F' defined as:

$$F' = \text{DISTRIBUTE}(F, Locality) = [\langle [Name: \mathsf{string};\ BirthDate: \mathsf{date};\ BirthCity: \mathsf{string}];\ [Name: \mathsf{string};\ BirthDate: \mathsf{date};\ BirthCountry: \mathsf{string}]\rangle]$$

SET and UNSET operators
The fourth rule takes into account sets of union of attributes. It separates the union into several sets. Therefore, SET and UNSET operators rise the union constructor through the collection constructor.

Example: Let's still consider the definition F given by $F = [SisterBrother: \{\langle SisterName: \mathsf{string};\ BrotherName: \mathsf{string}\rangle\}]$. Then the UNSET operator is decribed as:

$$F' = \text{UNSET}(F, SisterBrother) = [SisterBrother:[\{SisterName: \mathsf{string};\ BrotherName: \mathsf{string}\}]]$$

RENAME operator
This operator implements the last structural rule, rule 5. It allows to rename an attribute of a definition, in order to make attributes names more meaningful or identical to other attributes having the same meaning.

Example: A definition $F = [Birth: \mathsf{date}]$ could be replaced by the new definition F':

$$F' = \text{RENAME}(F, Birth, BirthDate) = [BirthDate: \mathsf{date}]$$

CAST operator
The CAST operator modifies an attribute's type of role r to make it compatible with another attribute of the same role (Rule 6). It means that this operator could transform (i.e. cast) the type `t` into a type `t'`, with all the modifications that may be induced on the objects defined by the given definition.

Example: Given $F = [task: \mathsf{string};\ length: \mathsf{string};\ code: \mathsf{integer}]$. The CAST operator transforms this definition into F' such as:

$$F' = \text{CAST}(F, length, \mathsf{integer}) = [task: \mathsf{string};\ length: \mathsf{integer};\ code: \mathsf{integer}]$$

At the object level, an object $o = [task= \texttt{"Design"};\ length= \texttt{"6"};\ code= 10]$ is converted into o' such that $o' = [task= \texttt{"Design"};\ length= 6;\ code= 10]$.

COMPLETE and SKIP operators
COMPLETE allows a one-to-one correspondence between two definitions. It is a way to add some attributes to a definition in order to have two definitions with the same degree. Therefore, with such an operator, it is easy to provide a complete compatibility between two definitions with different degrees. At the

object level, null values of the attribute's type are given to instances introduced in an object. SKIP allows to cut some attributes from a definition. These two operators realize the rule 7.

Example: If we suppose two definitions F and F', respectively $F = [task$: string; *length*: integer] and $F' = [task$: string; *length*: integer; *Programmer's_Name*: string]. Then, COMPLETE is defined as:

$$F' = \text{COMPLETE}(F, \textit{Programmer's\_Name}, \text{string}) = [\textit{task}: \text{string}; \textit{length}: \text{integer}; \textit{Programmer's\_Name}: \text{string}] \text{ and } F = \text{SKIP}(F', \textit{Programmer's\_Name})$$

At the object level, using the COMPLETE operator, we have, if o=[*task* = `"Design"`; *length* = `10`], then $o' = [$*task* = `"Design"`; *length* = `10`; *Programmer's_Name* = `""`].

FORMAT operator

The FORMAT operator includes differences on aggregation levels. It implements the rule 8. It is an easy way to transform the relation between two connected attributes and allows for a representation transformation of a piece of information.

Example: We introduce a definition on time to spend for a given task. Consider $F = [$*task*: string; *start_date*: date; *duration*: integer]. If we want a new definition which gives the duration of the given task with two attributes: *start_date* and *end_date*, we write:

$$F' = \text{FORMAT}(F, \textit{duration}, \textit{end\_date}, \text{date}, \textit{start\_date} + \textit{duration}) = [\textit{task}: \text{string}; \textit{start\_date}: \text{date}; \textit{end\_date}: \text{date}]$$

SCALE operator

This operator modifies the format of an attribute, e.g., a numerical value unit. Therefore, it solves the problem which can arise when two attributes are expressed in differents units. It is the application operator of rule 9. This operator can lead to a modification of the type of the considered attribute.

Example: On the time example described above, we now assume a duration expressed in hours, and another expressed in minutes. We have the following definition F: $F = [$*duration_in_hours*: integer]. We can apply the SCALE operator to transform this attribute in: $F' = [$*duration_in_minutes*: integer], and write:

$$F' = \text{SCALE}(F, \textit{duration\_in\_hours}, \textit{duration\_in\_minutes}, \text{integer}, \text{x} * 60) = [\textit{duration\_in\_minutes}: \text{integer}]$$

GROUP and UNGROUP operators

The UNGROUP and GROUP operators are related to the last rule. It allows for splitting or composition of one or several attributes. The GROUP operator solves the usual case when several attributes are grouped into a unique one in the target definition. Conversely, the UNGROUP solves the opposite problem.

Example: Consider a trivia example. We introduce a date attribute saved as a string on one hand, and as three separate strings on the other hand. So, we have

the definition F given by $F = [start\_date$: string$]$. We would like to obtain such a definition: $F' = [start\_date\_day$: string; $start\_date\_month$: string; $start\_date\_year$: string$]$. We write:

$$F' = \text{UNGROUP}(F, start\_date, (start\_date\_day, start\_date\_month, start\_date\_year)) = [start\_date\_day\text{: string; } start\_date\_month\text{: string; } start\_date\_year\text{: string}]$$

At the object level, considering an object $o = [start\_date =$ `"1/8/1992"`$]$, and using the UNGROUP operator, we have a new object $o' = [start\_date\_day =$ `"1"`; $start\_date\_month =$ `"8"`; $start\_date\_year =$ `"1992"`$]$.

3 An example

In this section, we illustrate on a short example the problems of data exchange between two tools. We first describe the context, and then give an example on the exchange of two figures that contain text data.

We assume that two tools named respectively A and B need to share some data. Both tools manage their own data, and we assume that one of them needs data from the other one. Assume that A needs to access B's data and respectively, B needs to access A's data. The process which enable the transformation of the data can be split into 3 main steps:

(*i*) get both A's and B's data definitions, as explained in Section 1 of the model. These definitions include the semantic of the data and will help in transforming the data,

(*ii*) point out the differences between definitions, by analyzing roles and types. As a result, we are able to describe differences between the two data's definitions (e.g. semantic differences, removed attributes, added attributes, type of an attribute),

(*iii*) make the definitions compatible in order to transform data. There are two sub-steps:

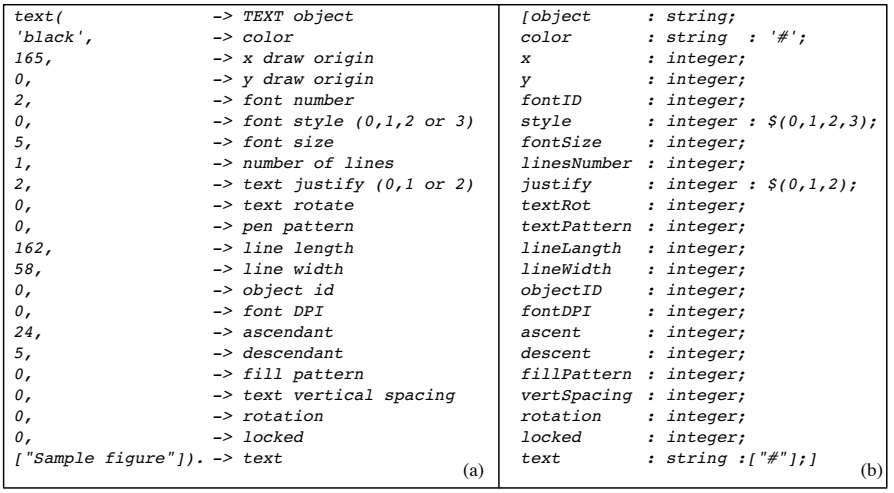
- compatibility of the definitions,
- application on the data.

After this general description of the process, we are going to describe each step in details. Before, we assume the following elements: the tool A stores its object text data as:

```
text('black',165,0,2,0,5,1,2,0,0,162,58,0,0,24,5,0,0,0,
                    0,["Sample figure"]).
```

While the second tool, B, stores its data as:

```
4 0 16 24 0 -1 0 0.000 2 24 189 164 30 Sample figure
```



```
text(                -> TEXT object
'black',             -> color
165,                 -> x draw origin
0,                   -> y draw origin
2,                   -> font number
0,                   -> font style (0,1,2 or 3)
5,                   -> font size
1,                   -> number of lines
2,                   -> text justify (0,1 or 2)
0,                   -> text rotate
0,                   -> pen pattern
162,                 -> line length
58,                  -> line width
0,                   -> object id
0,                   -> font DPI
24,                  -> ascendant
5,                   -> descendant
0,                   -> fill pattern
0,                   -> text vertical spacing
0,                   -> rotation
0,                   -> locked
["Sample figure"]). -> text
```

(a)

```
[object      : string;
color        : string  : '#';
x            : integer;
y            : integer;
fontID       : integer;
style        : integer : $(0,1,2,3);
fontSize     : integer;
linesNumber  : integer;
justify      : integer : $(0,1,2);
textRot      : integer;
textPattern  : integer;
lineLangth   : integer;
lineWidth    : integer;
objectID     : integer;
fontDPI      : integer;
ascent       : integer;
descent      : integer;
fillPattern  : integer;
vertSpacing  : integer;
rotation     : integer;
locked       : integer;
text         : string :["#"];]
```

(b)

Fig. 1. TEXT object attributes – Tool A

Figures 1(a) and 2(a) give values on the left side of the figure, and A's and B's data semantics on the right side. This is not the definition of the data, but only the semantic attached to each value. Figures 1(b) and 2(b) provide for each attribute the role name and its respective type. For instance, in figure 1(b), we notice the attribute x of type integer which corresponds to the **x draw origin** of the text. With figures 1(b) and 2(b), we retrieve object definitions introduced in the model in the first section. At this point, we assume that we get two object definitions, called F_A and F_B.

The second step concerns the matching of the definitions in order to provide a set of differences between them. To achieve this step, we use the hierarchy property proposed in section 1.4. It gives an order of inclusion between the two definitions. By analyzing them, we get the following:

- **common attributes**: *object, justify, fontID, fontSize, textPattern, color, style, lineWidth, lineLength, x, y, text, rotation, fontDPI*,
- **attributes in F_A missing in F_B**: *linesNumber, objectID, ascent, descent, fillPattern, vertSpacing, locked*,
- **attributes in F_B missing in F_A**: none.

Then, the third and last step concerns the definitions and objects transformations. To make the definitions fully compatible, we have to apply the following sequence of operators (we now assume that tool B wants to access A's data so the transformation will translate A's data in B's format).

First of all, we take into account common attributes. By ordering the definition F_A and skipping unused attributes (those which are not in the common set of attributes described above), and obtain a new definition called F'_A defined by:

```
4               -> object type (TEXT)
0               -> text justify (0,1 or 2)
16              -> font
24              -> font size
0               -> pen pattern
-1              -> color
0               -> fontDPI
0.000           -> angle
2               -> text style (1,2,4,8 or 16)
24              -> width
189             -> length
164             -> x draw origin
30              -> y draw origin
Sample figure   -> text
```

(a)

```
[object     : string;
justify     : integer : $(0,1,2);
fontID      : integer;
fontSize    : integer;
textPattern : integer : $(0,1,2,3);
color       : integer;
fontDPI     : integer;
rotation    : real;
style       : integer : $(1,2,4,8,16);
lineWidth   : integer;
lineLength  : integer;
x           : integer;
y           : integer;
text        : string]
```

(b)

Fig. 2. TEXT object attributes – Tool B

$$F'_A = [object\text{:string};\ justify\text{:integer};\ fontID\text{:integer};fontSize\text{:integer};\ textPattern\text{:integer};\ color\text{:string};\ fontDPI\text{:integer};\ rotation\text{:integer};\ style\text{:integer};\ lineWidth\text{:integer};\ lineLength\text{:integer};\ x\text{:integer};\ y\text{:integer};\ text\text{:string}]$$

Next, we are going to make the two definitions totally compatible (as explained in Proposition 3). To achieve this goal, we will apply the following sequence of operators:

- **Object attribute transformation**:
  $SCALE_{F'_A \to F_B}(F'_A,\ object,\ object,\ \text{integer},\ \mathcal{F}) = [object\text{: integer};\ justify\text{: integer};\ fontID\text{: integer};fontSize\text{: integer};\ textPattern\text{: integer};\ color\text{: string};\ fontDPI\text{: integer};\ rotation\text{: integer};\ style\text{: integer};\ lineWidth\text{: integer};\ lineLength\text{: integer};\ x\text{: integer};\ y\text{: integer};\ text\text{: string}]$
  where $\mathcal{F}$ is defined by:
  $\mathcal{F}$: string_value $\leftrightarrow$ integer_value,
  and the following translation table:

Tool A – String value	**Tool B – Integer value**
'oval'	1
'poly'	2
'polygon'	3
'text'	4
'arc'	5

  At the object level, we obtain the new object:
  $o = [object = 4;\ justify = 2;\ fontID = 2;fontSize = 5;\ textPattern = 1;\ color =$ 'black'; $fontDPI = 0;\ rotation = 0;\ style = 0;\ lineWidth = 58;\ lineLength = 162;\ x = 165;\ y = 0;\ text =$ 'Sample figure']
- **FontID attribute transformation**:
  $SCALE_{F'_A \to F_B}(F'_A,\ fontID,\ fontID,\ \text{integer},\ \mathcal{G}) = [object\text{: integer};\ justify\text{: integer};\ fontID\text{: integer};fontSize\text{: integer};\ textPattern\text{: integer};\ color\text{: string};\ fontDPI\text{: integer};\ rotation\text{: integer};\ style\text{: integer};\ lineWidth\text{: integer};\ lineLength\text{:}$

integer; x: integer; y: integer; *text*: string]
where $\mathcal{G}$ is defined as follows:

$\mathcal{G}$: integer_value $\rightarrow$ integer_value * 8.

At the object level, we obtain:
o = [*object*= 4; *justify*= 2; *fontID*= 16;*fontSize*= 5; *textPattern*= 1; *color*= 'black'; *fontDPI*= 0; *rotation*= 0; *style*= 0; *lineWidth*= 58; *lineLength*= 162; x= 165; y= 0; *text*= 'Sample figure']

– **FontSize attribute transformation**:
$SCALE_{F'_A \rightarrow F_B}(F'_A$,*fontSize*, *fontSize*, integer, $\mathcal{H}$) = [*object*: integer; *justify* :integer; *fontID*: integer;*fontSize*: integer; *textPattern*: integer; *color*: string; *fontDPI*: integer; *rotation*: integer; *style*: integer; *lineWidth*: integer; *lineLength*: integer; x: integer; y: integer; *text*: string]
where $\mathcal{H}$ is defined by:

$\mathcal{H}$: integer_value $\leftrightarrow$ integer_value,

and the following translation table:

Tool A – Integer value	**Tool B – Integer value**
0	8
1	10
2	12
3	14
4	18
5	20
6	24

At the object level, we have:
o = [*object*= 4; *justify*= 2; *fontID*= 16;*fontSize*= 20; *textPattern*= 1; *color*= 'black'; *fontDPI*= 0; *rotation*= 0; *style*= 0; *lineWidth*= 58; *lineLength*= 162; x= 165; y= 0; *text*= 'Sample figure']

– **Color attribute transformation**:
$SCALE_{F'_A \rightarrow F_B}(F'_A$, *color*, *color*, integer, $\mathcal{I}$) = [*object*: integer; *justify*: integer; *fontID*: integer;*fontSize*: integer; *textPattern*: integer; *color*: integer; *fontDPI*: integer; *rotation*: integer; *style*: integer;*lineWidth*: integer; *lineLength*: integer; x: integer; y: integer; *text*: string]
where $\mathcal{I}$ is defined by:

$\mathcal{I}$: string_value $\leftrightarrow$ integer_value,

and the following translation table:

Tool A – String value	**Tool B – Integer value**
'black'	-1
...	...

At the object level, we obtain the new object:
o = [*object*= 4; *justify*= 2; *fontID*= 16;*fontSize*= 20; *textPattern*= 1; *color*= -1; *fontDPI*= 0; *rotation*= 0; *style*= 0; *lineWidth*= 58; *lineLength*= 162; x= 165; y= 0; *text*= 'Sample figure']

– **Rotation attribute type cast**:
$CAST_{F'_A \rightarrow F_B}(F'_A$, *rotation*, real) = [*object*: integer; *justify*: integer; *fontID*:

integer;*fontSize*: integer; *textPattern*: integer; *color*: integer; *fontDPI*: integer; *rotation*: real; *style*: integer; *lineWidth*: integer; *lineLength*: integer; *x*: integer; *y*: integer; *text*: string]
At the object level, we have:
o = [*object*= 4; *justify*= 2; *fontID*= 16;*fontSize*= 20; *textPattern*= 1; *color*= -1; *fontDPI*= 0; *rotation*= 0.000; *style*= 0; *lineWidth*= 58; *lineLength*= 162; *x*= 165; *y*= 0; *text*= 'Sample figure']

– **Style attribute transformation**:
$SCALE_{F'_A \to F_B}(F'_A$, *style*, *style*, integer, $\mathcal{J}) =$ [*object*: integer; *justify*: integer; *fontID*: integer;*fontSize*: integer; *textPattern*: integer; *color*: string; *fontDPI*: integer; *rotation*: integer; *style*: integer; *lineWidth*: integer; *lineLength*: integer; *x*: integer; *y*: integer; *text*: string]
where $\mathcal{J}$ is defined as follows:

$$\mathcal{J}: \mathcal{J}_0 = 1$$
$$\mathcal{J}_i = \mathcal{J}_{i-1} * 2$$

At the object level, we have:
o = [*object*= 4; *justify*= 2; *fontID*= 16;*fontSize*= 20; *textPattern*= 1; *color*= -1; *fontDPI*= 0; *rotation*= 0.000; *style*= 1; *lineWidth*= 58; *lineLength*= 162; *x*= 165; *y*= 0; *text*= 'Sample figure']

This sequence of operators gives a new object compatible with B's format. By now, the o object can be accessed by tool B. Several comments can be made regarding this example:

– first of all, we only show the A to B data translation. There is no problem to do the opposite translation, but it takes longer. We give here some of the operators that should be use to do this. The COMPLETE operator should be used to make the B's definition totally compatible with A's. Some of the SCALE operators we give in the above example are conserved in the translation. In fact, only functions not using translation tables have to be changed, while bijectives functions (those with translation tables) can be preserved.
– in figures 1(b) and 2(b), there is some syntactical information. For instance, we can notice the *color* attribute describes as: *color*: integer: `'#'`. This definition indicates to the parser that the *color* attribute, known as a string attribute, is between two simple quotes. With such a method, we skip unnecessary information.
We have also introduced a way to describe default values of an attribute. This was done with the $ character. Between the parenthesis, we give default values of the attribute.
– finally, we would like to emphasize that there exists a important step that comes before all those described in this section. It concerns the need of having a unique notation of attributes of the same role. Our method supposes this unicity. A way to have this unicity is to take CDIF ([6]) semantic about attribute of text in a figure. Of course, definitions are rather poor against tool definitions, but it is quite normal because it gives only significant attributes.

We do not want to take the entire CDIF format, for the reasons explained in the following section, but only the semantic related to attributes. This allows to unify attribute's roles, and for instance to call the *style* attribute in a common manner (e.g. *FONTSTYLE*). The TEXT object definition in CDIF is presented in figure 3.

```
pic_tb              (UNBOUNDEDTEXT cdif_body pt inst_num
                                   symbol_name font_style)
cdif_body           (BODY cdif_text_body cdif_graph_body)
pt                  (PT integer integer)
inst_num            (INSTANCENUMBER integer)
symbol_name         (SYMBOLNAME string
font_style          (FONTSTYLE string)
cdif_text_body      (TEXTBODY ({string}))
cdif_graph_body     (GRAPHBODY)
string              "{legal_char}"
legal_char          abcdefghijklmnopqrstuvwxyz
                    ABCDEFGHIJKLMNOPQRSTUVWXYZ
                    0123456789
                    !@#$%^&*()-=_+`[]{}:;'\|/?.,<>
integer             0 to (2**16)-1
```

Fig. 3. TEXT object attributes – CDIF

4 Comments

We have introduced a data model to conceptually characterize objects and their definitions, where each object can be transformed by some operators. Our method for transforming data elements can be used both in *a priori* and *a posteriori* views. First, the model should be compatible with each tool without changing its internal data format. Then, for each data element definition, we construct a mapping that translates a definition into another for every situation. Finally, a set of operators allow us to build a translation procedure.

Let's make a few comments in regard of other methods. We believe that a common data format (like CDIF) does not solve all the problems. Such a solution implies that each tool changes its internal data format to manage a commonly accepted format. Such a process is advantageous in an *a priori* situation, but has drawbacks in an *a posteriori* case (for instance, if the tool vendor does not

want to change the tool format). So, integration could be difficult in such cases. With our model, each tool keeps its own data format and the translation is vendor independent. We believe this is well suited for an *a posteriori* situation and for moving a set of tools to a fine integrated system. Another strong point is the ability to control data without creating and managing a new common format. The format definition step is often very long, because of getting each participant an agreement on each part of the data format. The consensus step may be long, and is only *a priori* oriented. As emphasized before, our model allows for working in both *a posteriori* and *a priori* situations. So, it is useful to integrate old tools with new tools or to provide a way to change a tool in order to allow environment's evolution. These two ways of implementing data integration clearly complement each other, as they cover different steps of data integration problem solving.

We have seen in our model how each tool continues to manage its data and how there exists an "on the fly" translation when another tool wants to access these data. This characteristic provides simplicity and efficiency (data irredundancy). The identification and the representation of data elements, and the semantic interpretation are already made, and so, data translation and integration process are quite easy. Moreover, some level of automation is allowed. Our model presents solutions on current problems described before: the *synonyms* and the *homonyms* problems are dealt with (definition of each object, compatibility levels, GROUP and UNGROUP operators) but not entirely solved, we also suggest a solution to the *loss of information* problem (COMPLETE operator), and then take into account the *different level of scale* problem (FORMAT operator). We provide answers about syntactic constraints, but all the syntactic and semantic constraints part needs further research. Then, the approach offers a preferred extensibility and evolution path. It allows continued operation of existing applications to remain unchanged. Moreover, we feel that our approach makes changes and modification processes faster[1] than the adoption of a common data format as suggested by the work around CDIF, for instance.

An essential part of future research devoted to several unsolved problems, will be as described below. There exists at least two kinds of improvements: first, we distinguish model improvements (quoted with $\diamond$), and then we present implementation improvements (quoted with $\bullet$):

- $\diamond$ *the "semantic constraints" problem.* If a tool has to manage data produced by another tool, it must respect semantic constraints related by the owner tool. For instance, when a tool A considers a constraint saying that the value of an attribute x is smaller than the value of an attribute y, another tool B which has to manage A's data should ensure this constraint. So we must provide in our model such a property to enforce semantic constraints,
- $\diamond$ *the "world representation" problem.* We need to investigate all the possible cases of usual semantic that can occur. In particular, we have to account

[1] More work has to be done on the common data format and on changing tools internal data format on the other hand.

for specific cases. A starting point to solve this problem could be to get evolutions of CDIF format to provide classes of common attribute sets,

- *the "getting representation" problem.* Each tool must provide its data format in a common manner. It is important for the definitions to be described with the same model, so it can be useful to provide a frame to enforce tools to use the same symbol semantic description. Analyzis and comparison steps will be easier if we can use a complete and a common semantic description,
- *the data integrity problem.* We must provide a consistency implementation mechanism to ensure the global consistency criteria. One of the main problem is to track multiple data updates, in various format, and to provide a mechanism which respects the semantic constraints of the proprietary tool. Our approach currently does not support concurrent access to data by various tools. A transaction mechanism may be needed. But, for the moment, we consider that these problems are implementation dependent and we do not deal with them in the model.

As said before, this model needs further improvements. Rather than the classical tool oriented view, we think that a data oriented view will be a great challenge to solve some conflicting situations such as homonymy and synonymy problems. "*One data file, various tools able to manage these data*" scheme is preferred to the classical "*one tool, one data format*" scheme. To provide this control, we will introduce a group (or class) concept ([14]), where various tools with the same functionalities are regrouped into classes. Each tool in a class is closely related to other tools in the same class. For each class, there exists a common data semantic which allows for a high automation of the translation process. The new scheme can be describe as: "*given data, give me a list of authorized tools I could use to manage these data with maximum automation*". Our main effort is currently to find an efficient way to solve some of these problems (in adding new pieces of information in the model itself, i.e. *constraints*).

From a practical point of view, there exists an ongoing UNIX[2] implementation, which will be implemented on a PCTE [5] platform.

References

1. S. Abiteboul, P.C. Fischer, and H.J. Schek Eds. *Nested Relations and Complex Objects in Databases*, volume 361 of *Lecture Notes in Computer Science*. Springer-verlag, 1989.
2. S. Abiteboul and R. Hull. Restructuring Hierarchical Database Objects. *Theorical Computer Science*, 62:3–38, 1988.
3. N. Boudjlida and H. Basson. Integration Mechanisms in ALF, a Process Model-Based Project Support. In IEEE Computer Society Press, editor, *Proceedings of the 2nd International Conference on System Integration, "Managing Large-Scale Integration in the 1990s"*, Morristown, NJ, June 1992.
4. L. Cardelli and P. Wegner. On Understanding types, Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[2] UNIX is a registered trademark by AT&T.

5. ECMA. A Reference Model for Frameworks of Computer-assisted Software Engineering Environments. Technical report, ECMA TR/55 (2nd edition), December 1991.
6. EIA. CDIF Organization and procedure manual. EIA/PN-2329, January 1990.
7. J. V. Guttag. *The Specification and Application to Programming of Abstract Data Types*. PhD thesis, University of Toronto – Computer Science Departement, 1975. Report CSRG-59.
8. J. V. Guttag. Notes on type abstraction. *IEEE Transaction on Software Engineering*, 6(1):13–23, January 1980.
9. R. Hull and C.K. Yap. The Format Model: A Theory of Database Organization. *Journal of the ACM*, 31(3):518–537, July 1984.
10. C. Lécluse and P. Richard. The O_2 Data Model. Technical report, Altair 39-89, October 1989.
11. A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering – Vol. SE-13, No. 7*, pages 785–798, July 1987.
12. S. Navathe, E. Ramez, and J. Larson. Integrating User Views in Database Design. *IEEE Computer*, pages 50–62, January 1986.
13. A. Ohori. Semantics of Types for Database Objects. *Theorical Computer Science*, 76:53–91, 1990.
14. O. Perrin and N. Boudjlida. Tool Integration in Integrated Software-Engineering Environments: data dimension. In *Fifth International Conference on Software Engineering and its applications*, pages 95–105, Toulouse, December 1992.
15. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.
16. J.M. Smith and D.C.P. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133, June 1977.
17. I. Thomas and B. A. Nejmeh. Definitions of Tool Integration for Environments. *IEEE Software*, pages 29–35, March 1992.

Chapitre 8

A Model to Support Collaborative Work in Virtual Enterprises

Auteurs : Olivier Perrin, Claude Godart

Référence : Data and Knowledge Engineering Journal.

Abstract Virtual enterprises gather partners distributed in space, time and organizations, in order to achieve a common goal. Their business process realization needs the coordination of their distributed interactions. This paper presents the *Synchronization Point* model. It provides support for cooperative process management and coordination. It offers pertinent information about work progress while maintaining adequate privacy of information, and supports both long-time transactions and dynamic process definition. Then, its data repository and activity manager helps human interactions in cross-organizational applications.

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 50 (2004) 63–86
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

A model to support collaborative work in virtual enterprises

Olivier Perrin, Claude Godart

LORIA-INRIA-UMR 7503, BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France

Available online 25 January 2004

Abstract

Virtual enterprises gather partners distributed in space, time and organizations, in order to achieve a common goal. Their business process realization needs the coordination of their distributed interactions. This paper presents the *Synchronization Point* model. It provides support for cooperative process management and coordination. It offers pertinent information about work progress while maintaining adequate privacy of information, and supports both long-time transactions and dynamic process definition. Then, its data repository and activity manager helps human interactions in cross-organizational applications.

Keywords: Virtual enterprises; Cooperative activity; Coordination; Collaboration; Contracts

1. Introduction

Economic pressures (margin erosion, development costs, time-based competition) are placing emphasis on how organizations operate and interoperate with other enterprises to achieve a common business goal [8]. Business-to-Business (B2B) interactions must simply take place, and organizations must work more directly with their suppliers and customers to respond more quickly to changes. At the same time, rapid growth of web technologies is beginning to transform traditional inter-enterprise business models and allows virtual enterprise creation.

To enable organizations to adapt to this new business environment, middleware is required for providing dynamic and flexible integration between partners in the value chain. Although new technologies are needed to enable such integration, they have to work seamlessly with both intra-enterprise and inter-enterprise business processes, while maintaining the privacy of information and the autonomy of participants [6,9]. In this paper, we propose a concept that tries to answer these features and the corresponding middleware implementation in the context of Web services.

E-mail addresses: olivier.perrin@loria.fr (O. Perrin), claude.godart@loria.fr (C. Godart).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2004.01.004

Section 2 gives a definition of a virtual enterprise and a list of awaited features for cross-organizational processes. Section 3 introduces the main concepts our approach is based on: process services, synchronization points and contracts. Section 4 details the concept of synchronization points. An application example is described in Section 5 while Section 6 gives implementation details. Section 7 gives a comparison to related work. Section 8 concludes.

2. Problem statement

This paper is about cooperative processes for virtual enterprises. A virtual enterprise (VE) is an organization that allows enterprises to create a partnership for a specific project. To work on this project, trading partners have to share competences, resources and processes (or fragments of processes) and make members work together. However, these members belong to different physical organizations that can be in different places and different time zones. Contrary to classical organizational structure based on long-standing business partnerships that can rely on predefined processes to determine what should be done, the VE organizational model is more focused on what can be done to achieve a common goal, and it has to be more dynamic and loosely coupled with the specific business partners. Another difference is that a VE achieves its objectives by defining and implementing processes that are distributed across several organizations [21]. Each partner implements a subset of activities of the overall process (partners work on different activities that can be composed) and the same activity can be implemented conjointly by several enterprises. This means that activities must be coordinated and synchronized at a global level. To summarize, virtual enterprise implementation needs to integrate resources, organizational models, and process models of participants.

The classical definition of a process consists of a network of activities and their relationships [25]. This also applies in the cooperative processes we are concerned with, where partners from different enterprises realize both atomic and composite (sub-processes) activities. However, cooperative processes are only partially known and dynamic changes are often needed. Moreover, a major characteristic of cooperative activities is that they are realized by parallel flows of execution, each one modeling one activity of one contributor. The strong interdependency of partners' parallel work requires intermediate results exchange and process progress management while preserving partners' privacy.

While workflow systems could be an acceptable solution for predefined activities, they are not fully adequate for the coordination of non-sequential cooperative activities. Recently, Ellis [11] declared that "current approaches for coordinating cooperative activities do not fit because implementations of workflow tend to be coercive, isolationistic, and inflexible whereas the natural interaction of people frequently incorporates flexibility, opportunistic behaviors, social awareness, and compromise". Using a traditional workflow system to model such a cooperative activity could lead to many problems. Time efficiency could be impaired: partners may have to wait for termination of all activities before being able to estimate result compatibility and before making a decision. For faster reaction to changes, collaborative partners have to exchange intermediate results before the end of their contributions. Some flexible workflow systems can manage intermediate results exchange, but only if these exchanges are anticipated and modeled before the process execution. As stated earlier, human collaboration activities are too unpredictable and

cannot be totally anticipated. As the considered collaboration is not fully automated, and the human reaction is uncertain, a VE requires the ability for dynamic process definition and change. Another consequence of the human participation is the relatively long time of their reactions and the use of traditional ACID transactions is inappropriate for long-running activities. Instead, such activities require *long-time transactions*: atomicity and isolation levels must be relaxed and intermediate results can be released before transactions end. A cooperative behavior requires relaxed forms of transactions that are hardly supported by traditional workflow systems. Another available support consists in using groupware tools that provide implicit coordination means. They support multiple partners parallel work by managing divergence through version storage, but since there is no explicit control of this divergence, the global data integration risks to be delayed until the last phase. This is problematic, because the divergence can be so high that the global integration is generally hard to achieve, if possible at all. To manage this problem, they provide group awareness means, which allow participants auto-coordination [10]. So, the main requirements for supporting VE cooperative processes are:

- management of complex interactions, such as coordination of long-running multi-organizations processes, synchronous and asynchronous event-driven interactions, and distribution and share of processes, resources and participants;
- coordination of data and control flows;
- management of long-running business transactions (access to intermediate results) and support of flexible and knowledge intensive business processes;
- integration with implicit coordination means (communication, awareness, ...).

3. Synchronization Point: overview and concepts

3.1. Overview

The *Synchronization Point* (SP) approach tries to combine the advantages of workflow and groupware tools. By adding the flexibility of group awareness and implicit coordination to the explicit coordination of workflow systems, we provide both a model and a tool that allow partners to coordinate themselves and to control the coordination as work progresses. An SP is a cooperative cross-organizational activity that provides facilities for managing composition, coordination and synchronization of cross-organizational processes spanning multiple organizations. It provides as much interaction and parallelism as possible while trying to guarantee the execution correctness of the overall cooperative process. It also provides a flexible definition of coordination and synchronization in order to allow cooperative process revision (addition, suppression or modification) as the work progresses.

Fig. 1 overviews the SP model and introduces *process services* and SPs. An organization manages its process (private process), using applications and/or workflows systems. It externalizes fragments of this process by publishing *process services*. A process service abstracts a business process fragment. An SP aims at coordinating the data flow, the control flow and the transaction flow between a set of process services. It is dedicated to the management of the cross-organizational exchanges and it controls the contract terms (deadlines, outcomes guarantees, ...) defined

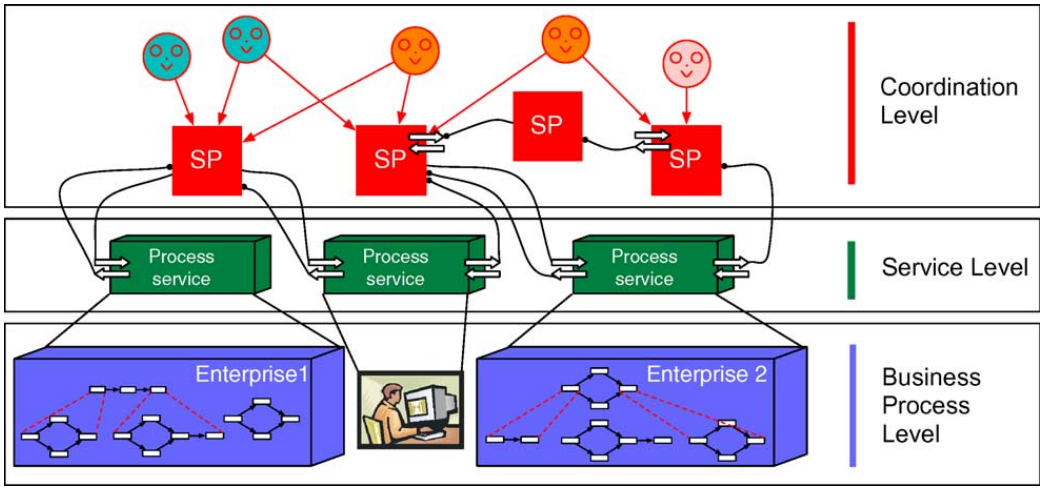


Fig. 1. Process service model and synchronization points.

in order to achieve the common goal. Due to the creative nature of process services synchronization, an SP is a cooperative activity. With such a model, a cross-organizational process is a partially ordered set of SPs.

3.2. Concept of process service

In order to manage the coordination between partners, an SP is using *process services*. To take part in cooperative projects, an organization must declare what it can offer to partners, using the description of the offered products and services. On the other hand, business processes need feedback, in order to control their own work progress. However, business processes are part of the enterprise strategic core, as they represent the organization know-how and contain a lot of proprietary information. Since all the outcomes are the result of an internal business process, partners must not have direct access to this information. A *process service*, conceptually based on the Service Oriented Architecture, is an abstract representation of an enterprise private business process. The *process service* model is a layered model, where the *process service layer* is a public abstraction of the outcomes the enterprise is able to deliver, and the *business process layer* is the internal process flow in the enterprise. The model clearly separates the enterprise offer from its implementation in order to respect the enterprises privacy needs: no direct access to internal processes and no restriction for their implementation. Besides the outcomes (products and events) description, a *process service* definition (close to definition of [20]) contains the conditions surrounding the offers (inputs, guarantees, observable states, ...), as well as the provider's information (access information, communications modes, ...), and includes information dedicated to data and activities synchronization, transactional management, and retrieval of services state [4]. Our *process service* definition is an extension of the WSDL standard [23].

Using the process service abstraction level, the cooperative process realization boils down to the problem of process services composition and integration. A cooperative process becomes a set of cooperative activities called SPs that coordinate partners work and interactions between process services. To achieve this coordination, an SP implements several functions supporting computer

mediated cooperation for controlling, deciding and evolving the cross-organizational cooperative process.

3.3. Concept of an Synchronization Point

An SP is both a cooperative activity and a cooperative process manager. As a cooperative activity, an SP must manage object sharing between contributing organizations, communication, and coordination. As a dynamic cooperative process manager, it must also manage process control, action decision, and process replanning.

3.3.1. Cooperative activity

An SP manages object sharing, communication, and coordination.

- *Object sharing*. This is the basic service to provide for cooperation support. It includes file version management and workspace management for respecting partner's privacy. It is important in the context of cross-organizational processes to preserve and ensure privacy and autonomy of participants. We must reduce objects' inter-visibility (i.e. visibility of a participant's object by another participant) as much as cooperation needs. This can directly influence the definition of SPs: for instance, it is not because an organization A shares an object with participant B and B shares another object with participant C that A accepts to share the object with C. Moreover, it is not because A shares the same object with B and C that it wants to synchronize simultaneously with B and C.
- *Communication*. When partners are coordinating their processes and reconciling their process differences, they need to interact with each other, to discuss, to share artifacts. This potentially includes chat, email, video-conferencing, white boards and so on. An SP must also support different communication policies (request/response, solicit response, one-way, notification, ...).
- *Coordination*. A major difficulty introduced by distribution is the coordination of the activities of team members. We address this problem by integrating both the *task coordination* approach (explicit coordination) with the *group awareness* approach (implicit coordination). Task coordination is based on the hypothesis that it is possible to agree on a process and to enforce this process on working sites. We address this problem using a high-level specialized process inspired from the Deming cycle (*Plan*, *Do*, *Check*, *Act*), keeping only three activities: check, act, plan. Group awareness (being aware of the partners work) is a good way to trigger communication between partners and auto-coordination in order to achieve the business goal. To achieve this, an SP notifies partners of all events concerning shared data and activities (i.e. creation, versioning, deletion, ...).

3.3.2. Cooperative process manager

To synchronize cross-organizational processes, an SP provides three main features.

- *Check*. To be able to synchronize processes, an SP needs information on their work progress. This means that an SP enforces and monitors work progress of partners using constraints on intermediate results, events (e.g. end of specific task, document reception, ...), deadlines or activity termination. Of course, to respect privacy, minimal information on internal processes

is needed, and SP can only use information that is associated to process services by contract. The role of an SP is to monitor and evaluate activities regarding the set of predefined criteria (presence of document, quality, schedule, objective fulfillment, ...). Depending on the result of check, the SP provides a link to control flow using two other related features: act and plan. Given these features, partners implied in an SP can decide actions to be undertaken and/or replanning.

- *Act*. Depending on the result of *check*, *actions* are decided upon by partners. The goal is either to validate the executed process, or to react and to fill the gap between the initially forecasted process and the executed one. A *decision* relates an event (why), an action (what), the entity in charge of its realization (who) and a schedule (when). Events can be either the result of the normal process execution, or an exception. An action can concern one or several organizations. Decisions are specified using *action rules*. For instance, what should be done when one partner does not respond? However, most important actions consist in replanning the process.
- *Plan*. The plan feature allows to deal with dynamic process changes of the SP instance. It allows us to revise the SP instance (e.g. deadline change, action rule modification, ...) as well as cross-organizational process adjustment (by adding a new SP for instance). This adjustment includes the shift of the end date of an activity, shift of an activity in a process plan, switch of two activities, the addition of an activity, the replacement of an activity by another, the suppression of an activity and so on. SP plan adjustment is based on workflow evolution rules as introduced in $\text{Adept}_{\text{flex}}$ [18]. Of course, cross-organizational process adjustment must be agreed upon by partners.

3.4. Concept of contract

In order to define the interactions between partners of the virtual enterprise that will be monitored and enforced by SPs, we use the concept of a contract. A contract is an agreement between two or more partners, defining the set of obligations and guarantees in a business process [7]. The definition of an *e-contract* is a hot topic within the Web services context. In [3], a classification of all the steps that are needed to define such a contract has been proposed. If we refer to this classification, our work focuses on the post-contractual/settlement phase called the "Contract fulfillment": contract execution, execution monitoring, re-negotiation and contract settlement evaluation. In particular, our contract model has the following objectives [17]:

- the contract definition helps to deploy SPs beyond the boundaries of the organizations;
- the contract definition allows partners to define process services and partners' needs and obligations;
- the last objective is to ensure that the SP execution is consistent with the contract definition.

In order to create an SP network as illustrated in Fig. 1, we propose the contract model presented in Fig. 2. The model relies on two concepts: relationships and *Sat* functions. A relationship, denoted *x rel y*, is a non-directional declarative constraint between a pair of objects *x* and *y*. Objects are entities that are shared in the cooperation context: processes, roles, or artifacts (e.g. documents). The order of the relationships is not important. A relationship will be validated by one (or several) satisfaction functions (called *Sat* functions). A *Sat* function takes objects or events

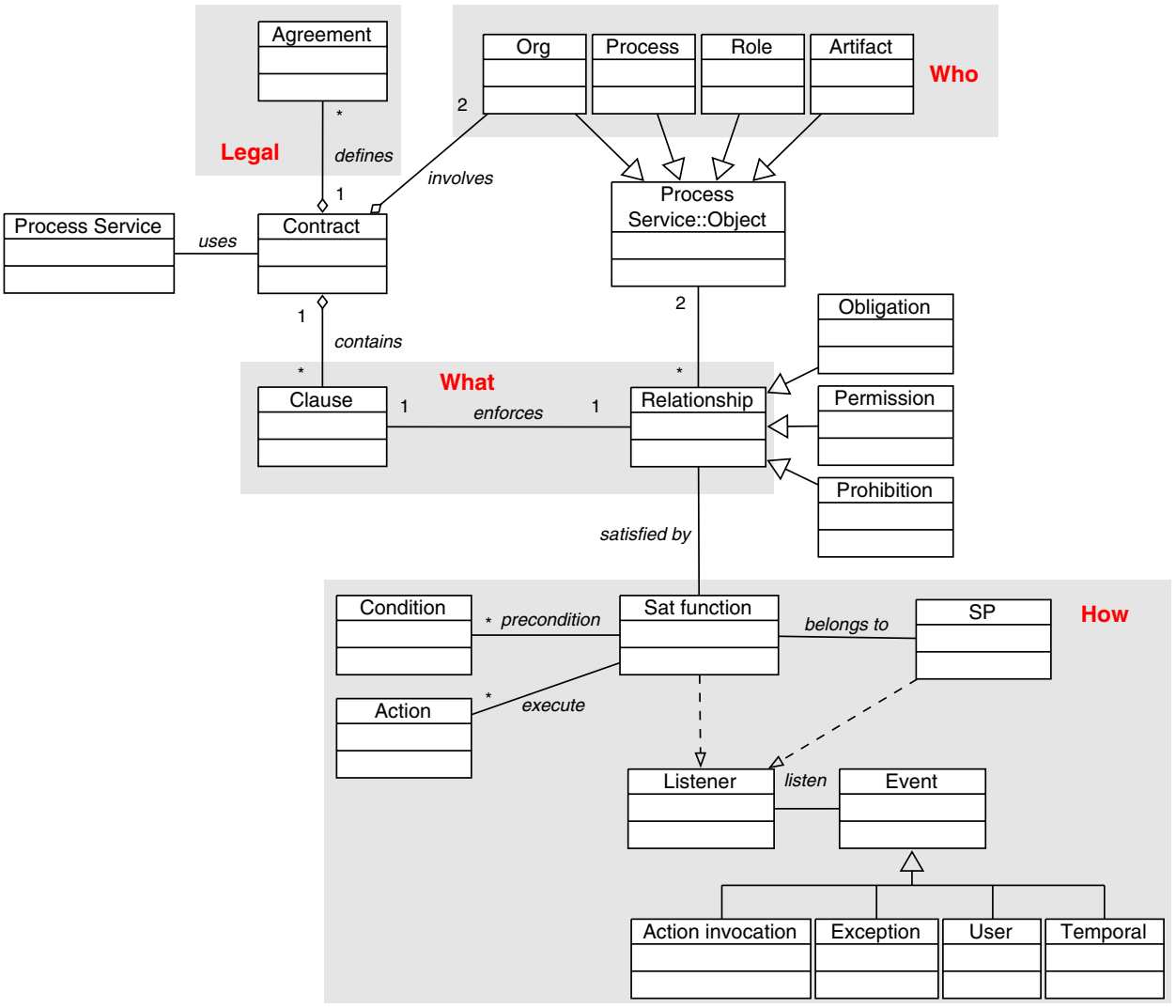


Fig. 2. The contract model.

as input. The result of the function is a boolean that determines whether the relationship between object *x* and object *y* holds. For example, a relationship can have as input a document and a date in order to ensure if a deadline is respected or not. This is written "organization A *delivers* document d". The relationship *delivers* is associated to a *Sat* function describing that a "document d" must be delivered by the organization A before June 23, 2003. Given relationships and *Sat* functions, our contract model is composed of four main parts:

- The *Who* part defines the involved partners. A partner is involved within a contract thanks to the process services it has published and especially using the objects these process services operate on. A partner is instantiated as a legal entity and can be represented by a person or an organization. An organization is involved with a given role that can be defined later in the contract.
- The *What* part contains the information about the contract subject, i.e. a set of clauses describing the core of the contract. This part defines the *relationships* to be satisfied. Relation-

ships describe respectively obligations, permissions and prohibitions ([9]) of the involved partners. During the negotiation phase (which is not taken into account in our work), relationships are discussed and negotiated between partners. Then, satisfying the set of relationships is the goal of an SP that is created using contract definition. We use both these relationships and the *Sat* functions to define, deploy and execute synchronization activities.

- The *How* part describes the execution steps of a contract, i.e. the activities that have to be achieved. Complementary to the textual description of the relationships done in the *What* part, we define the relationships between process services' objects and the way to satisfy them (the *Sat* functions). This can be for instance the results (data or messages) to be delivered, deadlines, rules to apply in case of failure, access rights based on organizations, roles, persons, and so on. Relationships and *Sat* functions are used to generate SP activities, ECA rules, data transfers, deadlines and termination conditions associated to these activities (generally, several *Sat* functions are associated to one relationship).
- Then, the last part of a contract, the *Legal* part, contains the textual clauses that address general agreements and conditions bound to a paper contract. It can be for instance references to existing applicable contracts or legislation texts.

An *organization* can take several different roles within a contract, depending on the process services it proposes. This allows to abstract roles. Moreover, a person (not represented in Fig. 2) is bound to a role and contains information such as email, phone, address, etc. An organization can also describe a workflow management system, an application or any other end-point of a given activity. Every *relationship* is bound to a contract clause and defines an execution step. As said before, a relationship belongs to one of the three following classes: obligation, permission or prohibition and is related to objects published by process services. A *Sat* function belongs to a SP, is associated to a relationship, is constrained by a *condition* and can execute an *action*. A condition is analyzed when an *event* is received by the SP and by the *Sat* function. The different event types are temporal events (deadlines for instance), user events (deliver event for instance), action invocation event (raised by a *Sat* function for instance) and exception events (failure). Exploiting these information, the SP ECA rules are in charge of the contract execution.

4. Synchronization point specification

4.1. Definitions

In the following, we denote a process service with upper case letter such as P, Q or R and we introduce $O(P)$ the set of objects manipulated by the process service P and denote its elements with lower case letters such as x, y or z. Objects of a given process service can be files, data, goals, but also sub-processes and/or activities.

4.1.1. General definitions

The WFMC defines a *business process* as a set of one ore more linked procedures or activities that collectively realize a business objective or policy goal, usually within the context of an

organizational structure defining functional roles and relationships. In our view, a *collaborative business process* represents the different activities among multiple organizations required to achieve a common given goal.

4.1.2. Relationships

We say that two process services cooperate if there exists a relationship between at least one of their respective objects, i.e. there exists a non-directional declarative constraint between a pair of objects. We denote a relationship $x\ rel\ y$ where x and y are objects and can be either documents, processes or sub-processes. This allows for instance to control the divergence of two versions of the same document in two organizations. Relationships help to ensure that a consistent version of the document will be produced by the cooperative activity.

4.1.3. Sat functions

Every relationship has at least one associated *Sat* function, denoted $Sat(x\ rel\ y)$, that checks whether the relationship is satisfied or not. If the function is not satisfied, it would exist actions such that the relationship is satisfied afterwards.

$$\forall x, y\ (x\ rel\ y) \Rightarrow \exists\ Sat(x\ rel\ y) | Range(Sat(x\ rel\ y)) = \{true, false\} \tag{1}$$

As objects can be processes, we can have

$$\forall P, Q\ (P\ rel\ Q) \Rightarrow \exists\ Sat(P\ rel\ Q) | Range(Sat(P\ rel\ Q)) = \{true, false\} \tag{2}$$

4.2. Synchronization point

Given two processes P and Q, and their objects $O(P)$ and $O(Q)$, an SP between P and Q is a particular process denoted $P{\infty}Q$ (pronounced "P synchronized with Q") such that:

$$\forall x \in O(P),\ y \in O(Q) \text{ and } (x\ rel\ y),$$
$$P{\infty}Q \Rightarrow Sat(x\ rel\ y) \in O(P{\infty}Q) \wedge Range(Sat(x\ rel\ y)) = \{true, false\} \tag{3}$$

Thus, if some relationship relates two objects from two processes, then its associated *Sat* function resides in the SP of these processes. An SP facilitates the synchronization of one cooperation between two or more processes. It includes every *Sat* function that resolves relationships between their objects or processes themselves. There exists a one-to-one relationship between the existence of an SP and the existence of a relationship among process objects. In other words, we assume that (1) we cannot have an SP without having at least one object relationship and (2) all the *Sat* functions that resolve relationship among process objects of two given processes P and Q are located in a $P{\infty}Q$ SP. We separate individual behaviors of processes P and Q and the behavior of their common SP.

An SP can also be used to manage the process of only one organization. If a single process P of one organization owns the objects involved in a relationship, we can consider the corresponding *Sat* function as a sub-process of $P{\infty}P$:

$$\forall x, y \in O(P)\ (x\ rel\ y) \Rightarrow P{\infty}P\ exists\ (\text{with } P{\infty}P \neq P). \tag{4}$$

Thus, if a relationship relates two objects of the same process, then this process should have an SP. This allows the process P to define and achieve the work, while $P{\infty}P$ controls. So, an SP can be used as a substitute for a WfMS.

4.2.1. Synchronization activities

A synchronization activity is an activity that tries to resolve a *Sat* function. It is included in an SP and is denoted $\frac{P\infty Q}{Sat(x\ rel\ y)}$ where $Sat(x\ rel\ y)$ is the *Sat* function that resolves the relationship of two objects $x \in O(P)$ and $y \in O(Q)$. An SP includes several synchronization activities that are executed in parallel. Then, the object set of an SP $P\infty Q$, denoted $O(P\infty Q)$, as the set of all shared objects of P and Q for which it exists both a relationship and a *Sat* function:

$$O(P\infty Q) = \bigcup_{i=1}^{n} x_i \in O(P) \cup \bigcup_{j=1}^{m} y_j \in O(Q) \cup \bigcup_{k=1}^{n* m} Sat_k(x_i\ rel_k\ y_j) | (x_i\ rel_k\ y_j) \tag{5}$$

When a synchronization activity fails (the *check* failed, i.e. one relationship *Sat* function cannot be satisfied), a first option can be to accept the *Sat* function to be unsatisfied. A human decision is needed and partners of the collaborative process are notified that a divergence can exist between the objects. A second option is to start the replan activity.

4.2.2. Privacy rules ($P\infty Q\infty R$)

4.2.2.1. Multi-process relationships. Let us consider relationships among three processes in order to evaluate the meaning of $P\infty Q\infty R$. First, we must keep in mind that processes can be part of different organizations. Initially, we have as many SPs as process relationships. Fig. 3a shows a situation where process Q is involved in two cooperations. Thus, P and R are not aware of each other, preserving their autonomy. In fact, we define a privacy rule where a process is involved in an SP if and only if a relationship exists among one of its objects and at least one object of the SP. We write

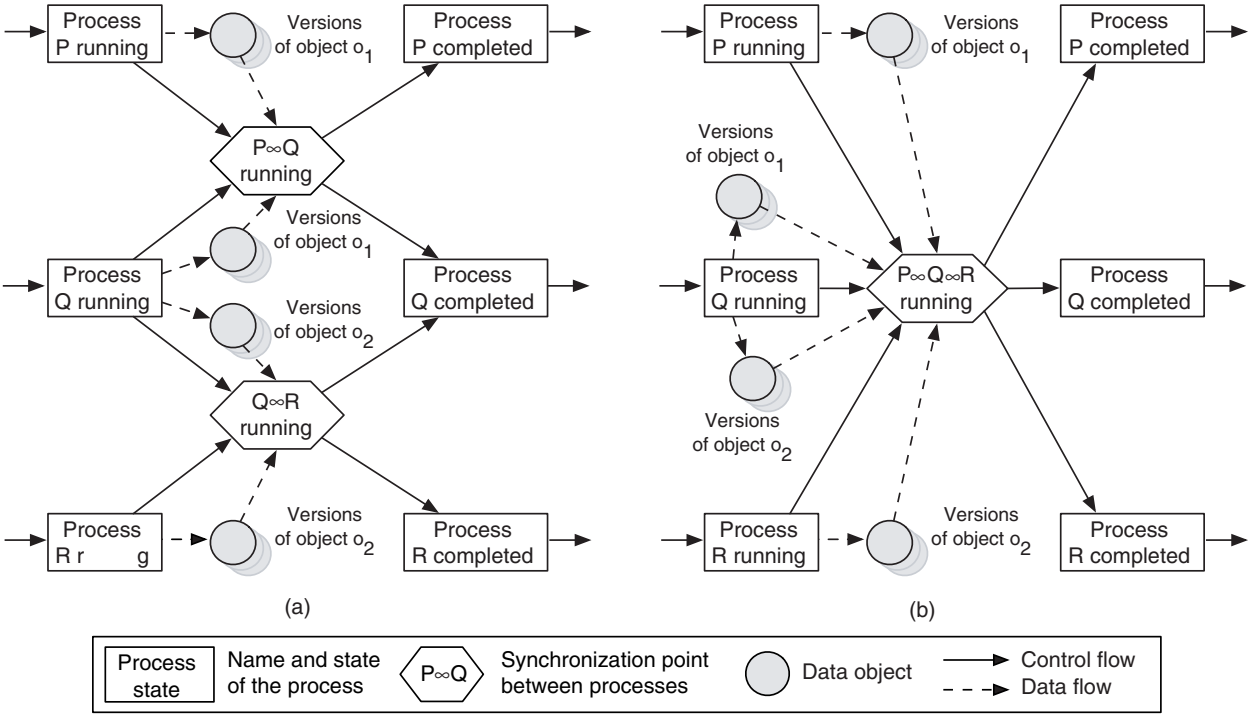


Fig. 3. Two synchronization points: (a) two synchronization points and (b) one merged synchronization point.

$$\forall x \in O(P_1), \quad x \in O(P_1 \infty P_2 \infty \ldots \infty P_n) \Rightarrow (\exists y \in O(P_2 \infty \ldots \infty P_n) | x \; rel \; y) \tag{6}$$

Definition 6 expresses in a recursive way the participation of a process in a cooperation and ensures that an SP shares only the objects necessary for the cooperation. Moreover, this allows to detect when visibility agreements are needed for an incoming process. Thus, we propose an SP model where we share "all necessary but no more and no less, under the control of each participant".

Coming back to Fig. 3a, if a relationship exists among o_1 and o_2 and if P and R accept to share those objects with each other, we can merge our two SPs in a single one. Fig. 3b depicts a situation where P, Q and R are all agree to cooperate and work together.

4.2.2.2. Cooperation contracts and synchronization points. Another aspect of SP calculus is given now. A necessary condition for the existence of an SP is the existence of at least one object relationship. But with more than two objects, relationships may be more complex and can generate more than really necessary synchronization activities. When an object is involved in several relationships at the same time, must we synchronize it in a single SP or must we define two or more SPs in order to serialize the modifications on the object? We assume that

$$\begin{aligned} &\forall x \in O(P), \; y \in O(Q), \; z \in O(R), \\ &\quad x \; rel \; y \wedge x \; rel \; z \Rightarrow P \infty Q \infty R \vee (P \infty Q) \infty R \vee (P \infty R) \infty Q \end{aligned} \tag{7}$$

Therefore, if $x \; rel \; y$, $x \; rel \; z$ and Q must process while R is running, we can propose a global SP if and only if both Q and R participants accept to work together. Otherwise, we must serialize two different SPs, selecting for example $P \infty Q$ and pushing its result in a cooperation with R such as $(P \infty Q) \infty R$ (different from $(P \infty Q \infty R)$ where only one SP exists grouping P, Q, and R).

For example, in order to perform a three-part cooperation, we take interest in Q and R object inter-visibility. Relationships $x \; rel \; y$ and $x \; rel \; z$ mean that P and Q on one hand and P and R on the other hand have accepted to share information but this is not yet true for Q and R; we assume here that a contract exists between P and Q and another exists between P and R. If those contracts include privacy protection clauses (e.g. protection of a part of shared documents), we must not provide entire visibility to R in Q and vice versa.

Thus, faced to a multi-relational definition, we apply the following algorithm:

(1) If a contract exists between Q and R and if P, Q and R can be grouped in a single SP, we create the SP.
(2) If contracts do not exist, we create SPs for each independent cooperation.
(3) If contracts do not exist and the process participants agree, we define new contracts in order to reduce the number of SP, and we apply case (1).

4.2.2.3. Process and sub-process. In order to ensure privacy and autonomy of process participants, we must reduce process object inter-visibility as tiny as cooperation needs. In a relationship between two processes P and Q such that P is a sub-process of a process R, $P \in O(R)$, and Q is a sub-process of a process S, $Q \in O(S)$, we may have $(P \; rel \; Q) \Rightarrow P \infty Q$ but not $R \infty S$ in order to preserve autonomy of processes R and S.

Considering a workflow management system activity, we could have the same visibility problem. In fact, we cannot differentiate its behavior and the behavior of objects it manipulates. That is why our model considers a workflow management system activity as a black box. Thus, to build a relationship among a workflow activity *wa* and a process *P*, we must first encapsulate *wa* in an object of a process *Q* and then declare the cooperation such as

$$\forall P, wa\ (wa\ rel\ P) \Rightarrow \exists Q | wa \in O(Q) \wedge Sat(wa\ rel\ P) \in O(P \infty Q) \tag{8}$$

4.3. State model

Our SP model is a state-aware coordination model and it includes a state model composed of two parts: first, an *object state* model for dealing with artifacts, second, a *process state* model for dealing with states of SP and processes.

The *process state* model (see Fig. 4a) is WfMC compliant [25]. We use the following states: *Initiated* (I), *Running* (R), *Suspended* (S), *Terminated* (T), *Activate* (A) and *Complete* (C). The only original transition is the *Deliver* transition intented to manage publication of intermediate versions within the SP repository.

The *object state* model (see Fig. 4b) is composed of the following states: *Defined* (d), *Instantiated* (i), *Modified* (m), *Read* (r), and *Delivered* (v). An object is first defined. At this time, we can define relationships including this object and therefore create SPs. The object is then instantiated, so it exists in a process repository, and it can be referenced within the scope of processes or SPs. The object can be modified by a process. For that, the process has read the object and writes it to its own repository. The object can also be in read mode. The last state is delivered. The object is uploaded to the SP repository. Once modified or delivered, an object can be modify or read again using a new version of this object. We distinguish (1) the *save* transition that consists in storing the object in the process' repository and (2) the *deliver* transition that stores

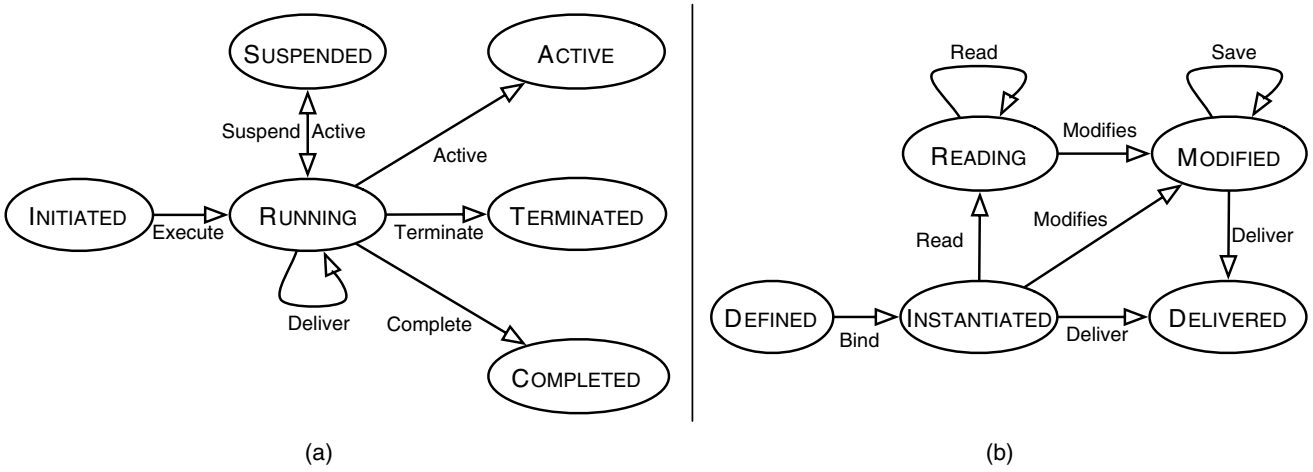


Fig. 4. State models: (a) process state model and (b) object state model.

an intermediate version into the SP's repository. Then, there exists cases where some transitions can be unavailable, depending the object types (e.g. workflow activities cannot be modified).

Given these two state models, we are able to deal with processes and SP orchestration. We use the notation $(entity)^{State} \Rightarrow Transition(entity)$ to express the fact that a transition depends on an object or process state. Using "·" for logical *and* and "+" for logical *or*, we are able to express WfMC AND-* and OR-* operators (for instance, $P^{Terminated} \Rightarrow Execute(Q \cdot R)$ represents a AND-Split). $P^{Complete} \Rightarrow Execute(Q)$ expresses that process Q must run when process P completes (sequential routing). Of course, the following properties should be checked:

(P1) *Complete* is an available state for process P,
(P2) *Execute* is a valid transition for process Q,
(P3) Q must be in a state compatible with an *Execute* transition.

These properties are checked using the following policy. At orchestration design time, we control states and transitions compatibility regarding the state model (P1 and P2). At runtime, we control the compatibility between the object state and the transition (P3). This policy facilitates validation of consistency of the system and capture of exceptions.

5. Putting it into practice

In this section, we describe in practice how to generate an SP network using contracts and process services. We first detail the deployment of the network. Then, we show how contracts' clauses are translated into relationships, and how these relationships are transformed into ECA rules in an SP.

5.1. Synchronization point deployment

In order to deploy the SPs network according to the contracts (given the link between relationships among artifacts and contracts among organizations), we have to create SPs using the following algorithm:

1. *read* the contract
2. *extract* process services information
3. *create* an SP object
4. *foreach* relationship and its *Sat* functions **do**
   // translate the relationship and its Sat functions into ECA rules
   - depending on event type, create a *listener* within the SP
   - translate the *Sat* functions into conditions of ECA rule
   - foreach action created, update the control flow (source and target processes)
5. reorganize the SP network according to (7).

We are able to validate the cooperation on three different levels:

- *static checks*, analyzing the contract definition for partners privacy policy,
- *deployment checks*, allowing to define the SPs network and to match data flow, control flow and transactional flow between process services,
- *runtime checks*, allowing to monitor and dynamically enforce the SP ECA rules.

5.2. ECA rules generation (by example)

5.2.1. From contract clauses to relationships

A contract definition identifies the cooperative activities. For instance, a contract will manage the following cooperative activity:

> CarBuilder1, CarBuilder2, EquipmentSupplier1 and EquipmentSupplier2 collaborate to the design of a front automatic wiper system. This activity is collaborative and the result is a requirements document. The system will use the speed and the rain intensity to adapt the swiping frequency.

In this contract definition, we identify the following information. First, participants (partners) are named without being precisely defined and bound to persons. The cooperative activity and its objectives are clearly identified. However, we must refine this description in order to complete the contract. This means that we must define the activities of each partner, the participants and their roles, the deadlines and quality criteria, a list of documents to be produced (both final and intermediate), the relationships that can exist between these documents, the documents available at the partners and their associated access rights, obligations, permissions and prohibitions of each partner, and finally what each participant is likely to offer. An important part of a contract concerns who is involved within the cooperative activity: activities, participants and roles are defined using contract clauses. A clause can take into account deadlines, documents to be produced, documents available to partners, and obligations, permissions and prohibitions. We now refine the description of the cooperative activity:

> EquipmentSupplier1 and EquipmentSupplier2 propose a joint requirements document that is enhanced or validated by CarBuilder1 and CarBuilder2. EquipmentSupplier1 and EquipmentSupplier2 are represented by two engineers, while CarBuilder1 and CarBuilder2 are represented by a wiper system project head. The validated document must be provided by the May 15, 2003. Process service offered by partners is identified as "Requirement Edition" (RE in short).

Then, we transform contract clauses into relationships. Considering the EquipmentSupplier1, it will propose a "Requirements document". The clause will be ensured using $r1$ and $r2$ relationships. The first relationship expresses the fact that the process service $\mathrm{RE}_{EquipmentSupplier1}$ is provided by EquipmentSupplier1, while the second says that the "Requirements document" will be delivered by the process service $\mathrm{RE}_{EquipmentSupplier1}$. This is written

r1: EquipmentSupplier1 *publishes* EquipmentSupplier1.$\mathrm{RE}_{EquipmentSupplier1}$
r2: EquipmentSupplier1.$\mathrm{RE}_{EquipmentSupplier1}$ *delivers*
EquipmentSupplier1.$\mathrm{RE}_{EquipmentSupplier1}$.Requirements

A second organization, EquipmentSupplier2, can proceed in the same way, and provides its own "Requirements document". This is written:

r3: EquipmentSupplier2 *publishes* EquipmentSupplier2.$\mathrm{RE}_{EquipmentSupplier2}$
r4: EquipmentSupplier2. $\mathrm{RE}_{EquipmentSupplier2}$ *delivers*
EquipmentSupplier2.$\mathrm{RE}_{EquipmentSupplier2}$.Requirements

One of the objective of the cooperation between the two organizations is to obtain a common version of the "Requirements document". This criteria is described in our model by a new relationship named $r5$. We write:

r5: EquipmentSupplier1.$\mathrm{RE}_{EquipmentSupplier1}$.Requirements *consistent with*
EquipmentSupplier2.$\mathrm{RE}_{EquipmentSupplier2}$.Requirements

Equipment suppliers propose solutions that meet their own internal developments. In order to access available processes, we use the process services of the organizations. This means that both EquipmentSupplier1 and EquipmentSupplier2 provide a process service that is in charge of an activity "Requirements edition". This activity is an abstract activity that can be connected to an application, a workflow management system or a human activity. In our example, the process service is bound to the corresponding activity of each equipment supplier, and the result of this process service is the "Requirements document". A third process service common to the two organizations is dedicated to the review of the two documents.

Once relationships are defined, we associate *Sat* functions to them. The contract stipulates that the "Requirements document" should be delivered until May 15, 2003, so relationships $r2$ and $r4$ are bound to the following functions:

Sat($r2$): EquipmentSupplier1.$\mathrm{RE}_{EquipmentSupplier1}$.Requirements.delivered
$\leqslant$ "05/15/2003"
Sat($r4$): EquipmentSupplier2.$\mathrm{RE}_{EquipmentSupplier2}$.Requirements.delivered
$\leqslant$ "05/15/2003"

Relationship $r5$ is bound to an obligation that describes the fact that partners must achieved the edition of the document before May 22, 2003 (i.e. the SP must be terminated and must deliver the version before this date). This obligation is enforced using the following *Sat* function onto relationship $r5$:

Sat($r5$): SP_{RE}.RE. Requirements.delivered $<$ "05/22/2003"

Using relationships, a contract becomes a set of ECA rules that will be used within an SP. Now, we describe how to translate relationships and their *Sat* functions into ECA rules.

5.2.2. From relationships and functions to ECA rules

Events. By using events it is possible to specify the changes occurring within an SP as well as the reaction to undertake, e.g., a document that has not been delivered in time is an event. Another type of event occurs when a document is correctly delivered. A third type of event is the execution of filters. When a Message Exchange Pattern (MEP—see Section 6 for details) is executed, several events are produced and events are caught by the SP using listeners to which the SP subscribes (see Fig. 2). Recovered events trigger actions. For instance, there are several possible *Sat* functions that can be assigned to one relationship. Relationships and their *Sat* functions are translated into ECA rules in order to be executed by the SP.

Obligations. Let us suppose it is May 16, 2003 and that the ''Requirements document'' was not delivered by EquipmentSupplier1. The *Sat* function associated to *r*5 cannot be satisfied, and an exception is raised. The event is a *temporal* event, where the condition checks whether the document was delivered or not. An action is executed. It can be either a replanning action (e.g., modification of the delivery date), a corrective action, or a preventive action. A corrective action is useful when replanning is not really needed, and a preventive action makes it possible to get a view of the situation before the deadline:

Event: user.check(SPrepository.Requirements)
Condition: delivered(SPrepository.Requirements)
Action: check(SPrepository.Requirements)

The rule works as follows: a user wants to check the ''Requirements document'' within the SP repository. Clicking a button fires an event. The condition associated is that the document is ready to be checked, implies that the document has been delivered. If the deliver has been done, the action to be executed is the effective consistency check (see *r*5) in order to accept the document. Another example is the following immediate corrective action:

Event: user.check(SPrepository.Requirements)
Condition: not available(SPrepository.Requirements)
Action: user.urgentRequest(SPrepository.Requirements)

Permissions and prohibitions. With regards to permissions and prohibitions, one can model the access to the documents, such as for example: CarBuilder1 can access the ''Requirements document''. The corresponding ECA rule can be written:

Event: user.read(SPrepository.Requirements)
Condition: hasRights(SPrepository.Requirements)
Action: download(SPrepository.Requirements)

Filters. A filter is a transformation applied to an object depending on the requester. The requester within the SP context is either a role (this means a user) or a process service. This is the reason why we have two types of filters: role filters that are applied for a given role and process filters that are applied for a given process. We do not propose organization filters because they are

subsumed by process filters, nor user filters because the role is an abstraction that alleviated from all the problems of directly managing users within the SP. We can show a simple example of the filtering function. Let us suppose three filters:

f1: $CarBuilder1.RE_{CarBuilder1}.x \rightarrow CarBuilder2.RE_{CarBuilder2}$
f2: $CarBuilder1.RE_{CarBuilder1}.x \rightarrow CarBuilder2.RE_{CarBuilder2}.projectmanager$
f3: $CarBuilder1.RE_{CarBuilder1}.x \rightarrow CarBuilder2.RE_{CarBuilder2}.engineer$

The first filter applies when the process $\mathrm{RE}_{CarBuilder2}$ belonging to CarBuilder2 desires to access the artifact x belonging the CarBuilder1's process $\mathrm{RE}_{CarBuilder1}$. That is a process filter. The second filter applies when the user with role *project manager* in the process $\mathrm{RE}_{CarBuilder2}$ belonging to CarBuilder2 desires to access the artifact x belonging to the CarBuilder1's process $\mathrm{RE}_{CarBuilder1}$. This is a role filter, as the third filter $f3$.

5.2.3. Discussion

To summarize the behavior of the SP including ECA rules, we have the following scenario. Let us suppose an SP denoted SP_{RE} coordinating two process services $RE_{EquipmentSupplier1}$ and $RE_{EquipmentSupplier2}$. These two process services respectively work on artifacts x and y. We can have the following relationships set $\mathscr{R}$ (to simplify the example, we only express a subset of the overall relationships set):

$\mathscr{R} = \{$
$rel1 = \langle P, x\rangle, Sat_1(rel1) =$ "P has the rights to deliver the artifact x"
$rel2 = \langle Q, y\rangle, Sat_1(rel2) =$ "Q has the rights to deliver the artifact y"
$rel3 = \langle x, y\rangle, Sat_1(rel3) = rel1 \wedge rel2, Sat_2(rel3) = merged(x, y)$
$rel4 = \langle x, date\rangle$, $Sat_1(rel4) = (SP_{RE}).x$ *deadline* $date(05/15/2003)$
$rel5 = \langle y, date\rangle$, $Sat_1(rel5) = (SP_{RE}).y$ *deadline* $date(05/15/2003)$
$rel6 = \langle rel3, date\rangle$, $Sat_1(rel6) = (SP_{RE}).rel3$ *deadline* $date(05/22/2003)$
$\}$

The SP will ensure that the relationships set $\mathscr{R}$ will be satisfied, or if not, it will help to replan or to adapt the cooperative context. *Sat* functions are translated as ECA rules. As shown on Fig. 5, an action can generate an event. For instance, if the condition of $R6$ is evaluated to false (documents are not in the SP repository and it is the 05/23/2003), the replanning event is generated. Then, this event can be treated in a new ECA rule. For instance, if evaluation of $R4$ results brings false value, we can generate a new action that enforces the delivering of the object x. In the same way, the condition can be optional. Thus, we can have a new ECA rule:

$Action.R4$	$res = (not(R4))$	$urgentRequest(RE_{EquipmentSupplier2}.x)$

We classify the events according to three categories: temporal events (e.g. a date), user events (e.g. the user initiates an event such as a deliver), and action events (e.g. events that are issued by an action such as a notification or a request). In our example, the $User.processService.deliver(x)$

Event	Condition	Action
$User.RE_{ES1}.deliver(x)$	$res = permission(RE_{ES1}, deliver, x)$	$Notify\ R1 = res$
$User.RE_{ES2}.deliver(y)$	$res = permission(RE_{ES2}, deliver, y)$	$Notify\ R2 = res$
$Action.R1, R2$	$res = R1 \wedge R2 \wedge merged(x, y)$	$Notify\ R3 = res$ $merged(x, y) \in SP_{RE}$
$Temporal.SP_{RE}.date1$	$res = x \in SP_{RE} \wedge (date1 < 05/15/2003)$	$Notify\ R4 = res$
$Temporal.SP_{RE}.date2$	$res = y \in SP_{RE} \wedge (date2 < 05/15/2003)$	$Notify\ R5 = res$
$Temporal.SP_{RE}.date3$	$res = (R3 == false) \wedge (date3 \geq 05/22/2003)$	$Replanning(SP_{RE})$

Fig. 5. ECA rules.

event is an event that pertains to the **user** class. On the other hand, the event *Action.R1,R2* is an event that pertains to the **action** class. Finally, the event *Temporal:SP.date3* is a **temporal** event.

6. Implementation overview

In our architecture, an SP starts when partners decide to start a cooperative activity in order to fulfill a given objective. Each partner only describes its activities, and contracts are established to express conditions and terms of data exchange and share. Then, a partner works in autonomy, having the ability to deliver ongoing version of its work to others and coordinating its activities using the SP. The current implementation is based on a distributed architecture that uses the Web services technology. Each partner hosts a part of the SP repository and exchanges are done using SOAP [22] messages. Using the level of abstraction introduced for describing organization processes, the architecture does not require any specific information about the organizational model or the process model. Then, "workflow enabled" applications and an explicit enterprise model are not mandatory. We use late binding to couple one activity to respectively an application, a participant, a workflow engine or a back-end process. The only requirement is the description of *process services* using an WSDL [23] extended version for describing processes properties.

An SP is managed by a tier (which can be viewed as a *broker*). The tier stores organizations end-points, projects, abstract descriptions of processes, contracts, roles and all the information about SPs. A contract is a XML document that helps us to filter what is the right information to provide to the right participant at the right time by setting up the exchange between two or more partners. A contract references a set of filters that are XML documents used to describe information that can be shared with others. Once being processed, the reference of the original document is delivered to the SP repository using a SOAP message, describing the organization end-point that delivers it. As the SP repository is distributed over all the partners of the cooperative activity, when an organization wants to access shared data, it requests the tier which partner (organization end-point) in the cooperation is able to deliver this data. Once the tier found the owner, it gives to the provider the end-point of the requester and the associated filter. Then, a SOAP based peer-to-peer conversation is engaged between the requester and the provider. The conversation is compliant with the existing contract between the two partners.

Fig. 6 depicts the architecture. We see that the private part of the process can be coupled with any internal application. Then, the process involved in the SP is published as a process

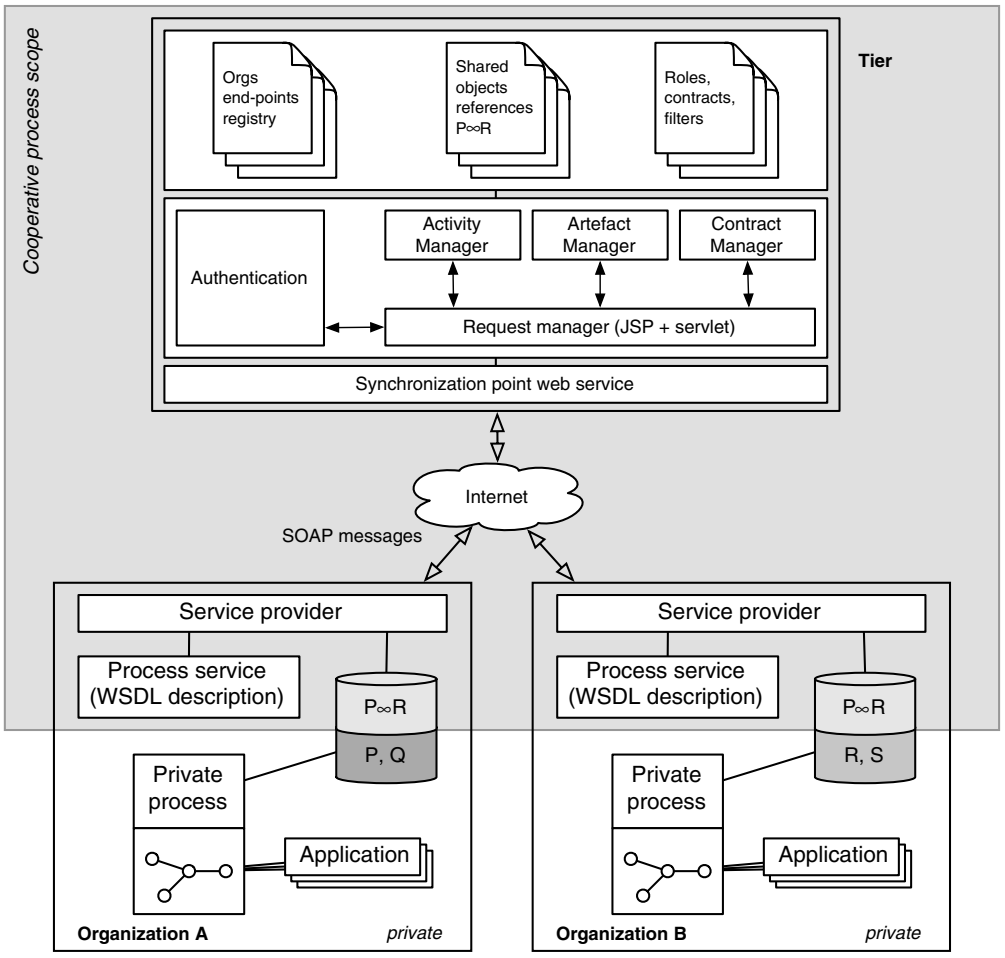


Fig. 6. Architecture diagram.

service. In the same time, a public repository is dedicated to the data stored within the SP repository for this particular organization ($P \infty R$ in Fig. 6). When an object is delivered to the SP, it is in fact stored in this particular repository and the reference of the object is sent and stored in the tier repository. The service provider is a web service that gives the ability to communicate with others using messages that are compliant to the W3C MEPs we have defined: the MEP *deliver*, which implies publishing a document to the SP (the storage of the version in the SP shared space consists in storing the document in the organization and to refer it into the tiers), the MEP *download* (which uses the Java Network Launch Protocol technology) for downloading the document in the organization workspace (applying the existing filters), and the MEP *read* for reading the document without downloading it. Control flow is synchronized using synchronization activities (*check*, *act*, and *plan*) belonging to the SP. Once an activity is achieved (the service provider notifies the tier that a given *process service* is completed), its results are delivered to the SP repository and when coordination is done, the next activity starts if and only if all the checks (as defined in the contracts) are valid and the reconciling operations on shared objects are done. Data flow is managed using a configuration and version management system named Toxic [12]. During execution of a cooperative process, we manage all the versions of artifacts.

For each process (i.e. P), we define a repository where objects involved in the process (i.e. $O(P)$) are stored. Process activities use this repository to manage objects. It also exists an SP repository that allows all the process participants involved in the cooperation to share objects. So, given two processes P and Q that collaborate we may have three repositories: $O(P)$, $O(Q)$ and $O(P \infty Q)$. Given the multi-organizational context, it is possible to define privacy policies for the repositories.

Different versioning policies are existing for SP and for processes. In fact, an SP repository is considered as a contractual view of one or more organization version spaces while processes and activities repositories are considered as workspaces dedicated to evolution of data. That is, SPs, individual processes (those which are not involved in a cooperation) or merely processes that manipulate some relationship independent objects (objects that are not shared) have a direct read and/or write access to organizational repositories. Therefore, they use the versioning policy of the organizational repository when checking in or checking out. On the other hand, processes manage their own versions independently in their own repository. So, version mismatches induced by this policy are hidden by privacy rules on process repositories. Moreover, this avoids an exponential amount of version synchronization messages between organizations. When an SP completes, version spaces of the processes involved are reconciled, and the SP's version space is reconciled with the organizational repository. This schema for version management also applies for nested processes and their respective repositories and long-term transactions.

By providing a shared space for versions associated with the SP (where versions of artifacts are delivered), we are able to keep one participant of this SP in touch with work progress of others. As objects are under version control, and by having access to intermediate versions, each participant can see what others did on shared objects, revisions, differences between versions and who changed what and when. Of course, this asynchronous awareness can be considered as simple, but we believe that it can help to ensure that everyone has up-to-date information about how the project and the collaborative activities make progress. Moreover, each participant keeps the complete control on its own objects, a mandatory feature for all partners.

Finally, our prototype provides the ability to trace the work done during both autonomy periods and collaborative periods. By storing main states of all activities, duration of these activities, missed deadlines or decisions taken, we provide a view of what happened and when, and where the process failed or succeeded.

7. Related work

Current work in the research field on automating processes is not directly applicable in the virtual enterprise domain. Most of current propositions lead to tightly couple one organization with another both at the architecture and process levels. Moreover, transactions models are not accurate. For instance, WfMC XPDL [25] can be used to define processes but compensation cannot be modeled and locking access to shared resources is not desirable as one organization risks to loss its autonomy. Then, recovery operations cannot be under the responsibility of only one organization [19]. In fact, two contradictory needs coexist: the autonomy of partners and the need to get some information about processes hold by the others. Our proposition is different from work such as CrossFlow [14], where organizational model must be replicated and where applications must be ''workflow enabled'' in order to be able to extract any necessary data the system may need.

In [1], van der Aalst proposes a model compliant to the WfMC model. Workflows are modeled using high-level coloured petri nets. A model can be split into different parts and provides an expressive notation that allows to model aggregation, loops, branches, concurrency, and time constraints. The approach also builds on a lifecycle model, and algorithms are introduced to determine the soundness of a given net. However, the approach is top-to-bottom and it does not allow to model exceptions. It also lacks the notion of roles and the communication between the partners.

Georgakopoulos [20] presents in CMI (Collaboration Management Infrastructure) the concept of window of opportunity which is used for conciliating prescribed activities inherited from WfMS and optional activities inherited from groupware applications. Our SP concept helps to provide the same kind of feature, but it wants to be more general and flexible, and gives the opportunity to exchange intermediate results. Some ideas for the process service concept came from [21].

In Weske [24], an approach to cope with flexibility in workflows management systems is presented but multi-enterprises processes are not in the scope of this reflection. One interesting idea is the definition of a meta-schema for workflows and the use of graphs for defining workflows.

The work of Casati [5] describes data exchange and process interoperability, but in a B2B context, where exchanges are limited to peer-to-peer conversations. We consider their concept of traces very interesting and we forecast to include this kind of mechanism in a future version of the prototype.

In BPEL4WS [2], it is possible to query the states and control the execution of process instances. BPEL relies on WS-Coordination [15] and WS-Transaction (WS-Tx) [16] to handle long-running processes, but some limitations remain. Business Activities of WS-Tx are said to be able to manage long duration activities. However, they leave the definition of the behavior (the coordination logic) to the developers of one organization, which means that one participant should define and implement the expected behavior for all participants before the cooperation

starts (it can be a non-trivial task). Moreover, the specification has an odd compensation model (see [13] for details). So we claim that these propositions fail to provide solutions for either long term transactions, coordination phases, or binding to internal processes.

8. Conclusion

In this paper, we propose a model and an architecture that support collaboration and cooperation for multi-enterprises processes. The SP model tries to offer advantages of both workflow management systems and groupware systems, and different partners can work together once they defined cooperation rules corresponding to contract specification. Then, an SP is able to coordinate the work progress, and it provides all the participants with accurate information on work evolution. By including data and control flow management functions in SP, we allow a flexible process definition, and we can adjust the collaboration process definition by updating an SP during work progress.

The first feature of the SP model is the platform and system independence. Thus, it is easy to integrate the model with existing back-end systems of organizations and it is not tightly coupled with workflow-enabled systems. Another feature is the adherence to component-based design and composability. It is possible to compose several processes viewed as components in order to obtain an overall composed process and to easily manage multi-enterprises processes. Then, if you need to cooperate to achieve a common objective, you need to exchange and to provide to others intermediate results, breaking the ACID properties of classical transactional models used by traditional workflow management systems. The SP model tries to break these limits.

Flexibility, portability, and interoperability are also supported and the SP model offers an effortless enterprise information systems integration, while preserving autonomy of each partner. We do not impose to be compliant to any model (both process or organizational models) in order to use our model and our architecture. This is hardly possible when we try to do this using workflow management systems. Then, we adopt a service-oriented architecture, and we rely on standards such as SOAP, WSDL and XML. This is used to accommodate and fit both the architecture and the model without to throw away all the work done and to benefit from future enhancements.

Acknowledgements

Some of these contributions are parts of a cooperation between LORIA and France Telecom R&D *e-Process* project, and are also inspired from the Eureka ITEA-EAST project.

References

[1] W. van der Aalst, K. van Hee, Workflow Management: Models, Methods, and Systems, MIT Press, Cambridge, MA, 2002.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leyman, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Business Process Execution Language for Web Services, BEA, IBM, Microsoft, SAP, Siebel, 2003.

[3] S. Angelov, P. Grefen, B2B eContract handling—a survey of projects, papers and standards, CTIT Technical Report 01-21, University of Twente, 2001.
[4] J. Bitcheva, O. Perrin, C. Godart. Cross-organizational processes coordination, Technical Report LORIA A02-R-046, Nancy, May 2002.
[5] F. Casati, M. Sayal, U. Dayal, M.C. Shan, Integrating workflow management systems with B2B interaction standards, in: Proceedings of the 18th International Conference on Data Engineering (ICDE), February 2002, IEEE Computer Society, San Jose, CA, 2002, pp. 287–296.
[6] Q. Chen, U. Dayal, M. Hsu, M. Griss, Dynamic agents, workflow and XML for e-commerce automation, in: First International Conference on E-Commerce and Web-Technology (EC), UK, 2000.
[7] D. Chiu, S. Cheung, S. Till, A three layer architecture for e-contract enforcement in an e-service environment, in: Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36), January 2003, IEEE Computer Society, Silver Spring, MD, 2003.
[8] A. Dan, F. Parr, Long running application models and cooperating monitors, in: High Performance Transaction Processing Workshop, Asilomar, CA, 1999.
[9] U. Dayal, M. Hsu, R. Ladin, Business process coordination: state of the art, trends, and open issues, in: Proceedings of 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy, September 2001, pp. 3–13.
[10] P. Dourish, V. Bellotti, Awareness and coordination in shared workspaces, in: Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW), October 1992, ACM Press, Toronto, Ontario, 1992, pp. 107–114.
[11] C. Ellis, Net models supporting human and humane behaviors, Invited talk, in: Conference on Business Process Management (BPM), Eindhoven, The Netherlands, June 2003.
[12] C. Godart, P. Molli, G. Oster, O. Perrin, H. Skaf-Molli, P. Ray, F. Rabhi, The ToxicFarm integrated cooperation framework for virtual teams. Distributed and Parallel Databases: Special Issue on Teamware Technologies 15 (1) (2004) 67–88.
[13] P. Greenfield, A. Fekete, J. Jang, D. Kuo, Compensation is Not Enough. In 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC), Brisbane, Australia, September 2003.
[14] P. Grefen, K. Aberer, Y. Hoffner, H. Ludwig, CrossFlow: cross-organizational workflow management in dynamic virtual enterprises, International Journal of Computer Systems Science and Engineering 15 (5) (2000) 277–290.
[15] D. Langworthy, F. Cabrera, G. Copeland, W. Cox, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, A. Nadalin, D. Orchard, I. Robinson, J. Shewchuk, T. Storey, Web Services Coordination, BEA, IBM, Microsoft, September 2003.
[16] D. Langworthy, F. Cabrera, G. Copeland, W. Cox, M. Feingold, T. Freund, J. Johnson, C. Kaler, J. Klein, A. Nadalin, I. Robinson, T. Storey, S. Thatte, Web Services Atomic Transaction (WS-AtomicTransaction), BEA, IBM, Microsoft, 2003.
[17] O. Perrin, C. Godart, A contract model to deploy and control cooperative processes, in: 4th International Workshop, Technologies for E-Services (TES), September 2003, LNCS, 2819, Springer Verlag, Berlin Germany, 2003, pp. 78–90.
[18] M. Reichert, P. Dadam, ADEPTflex—Supporting dynamic changes of workflows without losing control, Journal of Intelligent Information Systems, Special Issue on Workflow Managament 10 (2) (1998) 93–129.
[19] H. Schuldt, G. Alonso, C. Beeri, H.J. Schek, Atomicity and isolation for transactional processes, ACM Transactions on Database Systems 27 (1) (March 2002) 63–116.
[20] H. Schuster, D. Baker, A. Cichocki, D. Georgakopoulos, M. Rusinkiewicz, The collaboration management infrastructure, in: Proceedings of the 16th International Conference on Data Engineering (ICDE), February 2000, IEEE Computer Society, San Diego, CA, 2000, pp. 677–678.
[21] H. Schuster, D. Georgakopoulos, A. Cichocki, D. Baker, Modeling and composing service-based and reference process-based multi-enterprise processes, in: Proceedings of the 12th Conference on Advanced Information Systems Engineering (CAiSE), June 2000, LNCS, 1789, Springer Verlag, Stockholm, Sweden, 2000, pp. 247–263.
[22] Simple Object Access Protocol (SOAP) 1.1. W3C, May 2000.

[23] Web Services Description Language (WSDL) 1.1. W3C, March 2001.
[24] M. Weske, Formal foundation and conceptual design of dynamic adaptations in a workflow management system, in: Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34), January 2001, IEEE Computer Society, Silver Spring, MD, 2001.
[25] WfMC. Workflow process definition interface—XML process definition language (XPDL) (WFMCTC-1025). Technical Report, Workflow Management Coalition, 2002.

Olivier Perrin is an assistant professor at University Nancy 2 since 1995, and works at LORIA in the ECOO project. He received his Ph.D. from University Henri Poincaré Nancy 1 in 1994. His research interests include various topics in virtual organizations, Web services technologies, workflow management, and enterprise application integration. He has published several articles in international conferences, and he was involved in many international and european projects. His current work is on extending Web services and Grid computing with transactional properties for supporting virtual organizations, and he is currently involved in the European ITEA-EAST project with automotive partners.

Claude Godart is a professor at University Henri Poincaré Nancy 1. He is the scientific director of the ECOO project of LORIA, a joint venture between Universities of Nancy, INRIA and CNRS. The ECOO project is interested in the development of services for hosting virtual teams and organizations. His proper center of interest is the consistency maintenance of the data supporting cooperation between several partners. This encompasses user centric advanced transaction and workflow models. He has been implicated in several transfer project in cooperation with international, European and French companies.

Chapitre 9

Ensuring Required Failure Atomicity of Composite Web Services

Auteurs : Sami Bhiri, Olivier Perrin, Claude Godart

Référence : Proceedings of World Wide Web Conference 2005.

Abstract The recent evolution of Internet, driven by the Web services technology, is extending the role of the Web from a support of information interaction to a middleware for B2B interactions. Indeed, the Web services technology allows enterprises to outsource parts of their business processes using Web services. And it also provides the opportunity to dynamically offer new value-added services through the composition of pre-existing Web services. In spite of the growing interest in Web services, current technologies are found lacking efficient transactional support for composite Web services (CSs). In this paper, we propose a transactional approach to ensure the failure atomicity, of a CS, required by partners. We use the Accepted Termination States (ATS) property as a mean to express the required failure atomicity. Partners specify their CS, mainly its control flow, and the required ATS. Then, we use a set of transactional rules to assist designers to compose a valid CS with regards to the specified ATS.

Ensuring Required Failure Atomicity of Composite Web Services

Sami Bhiri
LORIA-INRIA
BP 239, F-54506
Vandoeuvre-les-Nancy Cedex, France
bhiri@loria.fr

Olivier Perrin
LORIA-INRIA
BP 239, F-54506
Vandoeuvre-les-Nancy Cedex, France
operrin@loria.fr

Claude Godart
LORIA-INRIA
BP 239, F-54506
Vandoeuvre-les-Nancy Cedex, France
godart@loria.fr

ABSTRACT

The recent evolution of Internet, driven by the Web services technology, is extending the role of the Web from a support of information interaction to a middleware for B2B interactions.

Indeed, the Web services technology allows enterprises to outsource parts of their business processes using Web services. And it also provides the opportunity to dynamically offer new value-added services through the composition of pre-existing Web services.

In spite of the growing interest in Web services, current technologies are found lacking efficient transactional support for composite Web services (CSs).

In this paper, we propose a transactional approach to ensure the failure atomicity, of a CS, required by partners. We use the Accepted Termination States (ATS) property as a mean to express the required failure atomicity.

Partners specify their CS, mainly its control flow, and the required ATS. Then, we use a set of transactional rules to assist designers to compose a valid CS with regards to the specified ATS.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.2.4 [**Database Management**]: Systems—*Transaction Processing*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Distributed commercial transactions*

General Terms

Design, Reliability

Keywords

Reliable Web services compositions, Failure atomicity, Transactional models

1. INTRODUCTION

Web services are emerging as a promising technology for automating B2B interactions. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can dynamically combine individual services to provide new value-added services.

A main problem that remains is how to ensure a correct composition and a reliable execution of a composite service(CS) with regards to partners transactional requirements.

Despite growing interest, Web services middleware is still rather primitive in terms of functionality, far from what current EAI middleware can provide for intra-enterprise applications [2].

The current Web services technologies ensure communication interoperability which is a part of the problem when considering the building of reliable Web services compositions [16]. Indeed, unlike activities in traditional workflows, services are defined independently of any computing context. Thereafter, the task of building composite Web services requires mechanisms to deal with the inherent *autonomy*, and *heterogeneity* of Web services.

Although powerful, *Advanced Transaction Models* (ATMs) [6] are found lacking functionality and performance when used for applications that involve dynamic composition of heterogenous services in a peer-to-peer context.

Their limitations come mainly from their inflexibility to incorporate different transactional semantics as well as different interactions patterns into the same structured transaction [8].

In this paper, we propose a transactional approach for reliable Web services compositions by ensuring the failure atomicity required by the designers.

From a transactional point of view, we consider a CS as a structured transaction, Web services as sub transactions and interactions as inter sub transactions dependencies. We use the Accepted Termination States (ATS) property as a correctness criteria to relax atomicity.

To the best of our knowledge, defining a transaction with a particular set of properties, in particulary ATS, and ensuring that every execution will preserve these properties remains a difficult and open problem [18].

The paper is organized as follows. Section 2 introduces a motivating example and gives the main points which has driven our approach. In section 3, we explain the notion of transactional web service and show how we express its transactional properties. Section 4 presents the notion of Transactional Composite (Web) Service (TCS) and explains our

WWW2005,, May 10–15, 2005, Chiba, Japan.
ACM 1-59593-046-9/05/0005.

transactional point of view. Section 5 presents the notion of Accepted Termination States (ATS) as a mean to express the required failure atomicity. Section 6 illustrates how our approach proceeds (using a set of transactional rules) to assist designers to compose valid TCSs. In section 7, we discuss some related work. Section 8 concludes our paper.

2. MOTIVATING EXAMPLE AND METHODOLOGY

Let us first present a motivating example. We consider an application dedicated to the online purchase of personal computer. This application is carried out by a composite service as illustrated in figure 1. Services involved in this application are: the **Customer Requirements Specification (CRS)** service used to receive the customer order and to review the customer requirements, the **Order Items (OI)** service used to order the computer components if the online store does not have all of it, the **Payment by Credit Card (PCC)** service used to guarantee the payment by credit card, the **Computer Assembly (CA)** service used to ensure the computer assembly once the payment is done and the required components are available, and the **Deliver Computer (DC)** service used to deliver the computer to the customer (provided either by Fedex (DC_{Fed}) or TNT (DC_{TNT})).

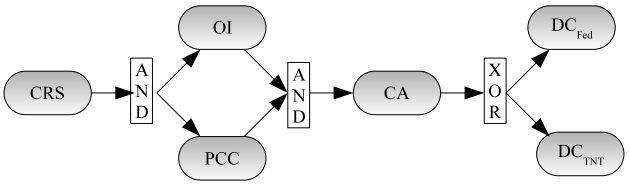


Figure 1: A composite service for online computer purchase.

When a user designs a composite service, he expects the service execution to be reliable. That means he particularly pays attention to failure handling. In our example, the designer may want to be sure: that one of the two delivery services will succeed, that the service CA is sure to complete, and that it is possible for the service OI to undo its effects (for instance when the payment fails). These properties define what we call the *transactional behavior* of the service. This behavior is specified using a set of transactional requirements. Since these requirements may vary from one context to another, the transactional behavior will vary too. For instance, a designer may accept the failure of the DC_{Fed} service in a context while in another one, he may not tolerate such a failure at such an advanced stage. So the mean of a reliable execution is tightly related to transactional requirements and it may vary according to designers.

In the same time, in order to ensure a reliable execution, we have to be sure that a specified transactional behavior is consistent with the set of selected services and the transactional requirements. Back to our example, we can easily notice that since the OI service is not sure to complete, the payment service PCC have to be compensatable (and it must be compensated when the OI service fails).

The following points introduce our approach and its concepts for supporting this kind of scenarios.

First, we believe that we must enhance Web services description for a better characterization of their transactional behavior. This can be done by enhancing WSDL interface with transactional properties.

Second, we have to model services composition and choreography, in particular mechanisms for failure handling and recovery.

Third, we have to provide designers with a mean to express their transactional requirements, in particular their required failure atomicity level. Finally, we have to support composite service validation with regards to designers' requirements.

In the rest of the paper we detail each of these issues and especially our set of transactional management rules for composite service validation.

3. TRANSACTIONAL WEB SERVICE DESCRIPTION

A Web service is a self-contained modular program that can be discovered and invoked across the Internet. Web services are typically built with XML, SOAP, WSDL and UDDI specifications [4] [17]. A transactional Web service is a Web service that emphasizes transactional properties for its characterization and correct usage.

The main transactional properties of a Web service that we are considering are *retriable, compensatable and pivot* [14]. A service s is said to be ***retriable*** if it is sure to complete after several finite activations. s is said to be ***compensatable*** if it offers compensation policies to semantically undo its effects. Then, s is said to be ***pivot*** if once it successfully completes, its effects remains for ever and cannot be semantically undone. Naturally, a service can combine properties, and the set of all possible combinations is $\{r; cp; p; (r, cp); (r, p)\}$.

In order to model the internal behavior of a service, we have adopted a states/transitions model. A service has a minimal set of states (*initial*, *active*, *aborted*, *cancelled*, *failed* and *completed*), and it also includes a set of transitions (*activate()*, *abort()*, *cancel()*, *fail()*, and *complete()*). The figure 2.a shows the internal states/transitions diagram of a pivot service.

When a service is instantiated, the state of the instance is *initial*. Then this instance can be either *aborted* or *activated*. Once it is *active*, the instance can normally continues its execution or it can be *cancelled* during its execution. In the first case, it can achieve its objective and successfully *completes* or it can *fail*.

The requested transactional properties can be expressed by extending the service states and transitions. For instance, for a compensatable service, a new state *compensated* and a new transition *compensate()* are introduced (e.g. service in figure 2.b). Figure 2 illustrates the states/transitions diagram of a retriable service (figure 2.c) and states/transitions diagrams of services combining different transactional properties (figures 2.d and 2.e).

Within a transactional service, we also distinguish between external and internal transitions.

External transitions are fired by external entities. Typically they allow a service to interact with the outside and to specify composite services choreographies (see next section). The external transitions we are considering are *activate(), abort(), cancel(), and compensate()*. Internal transitions are

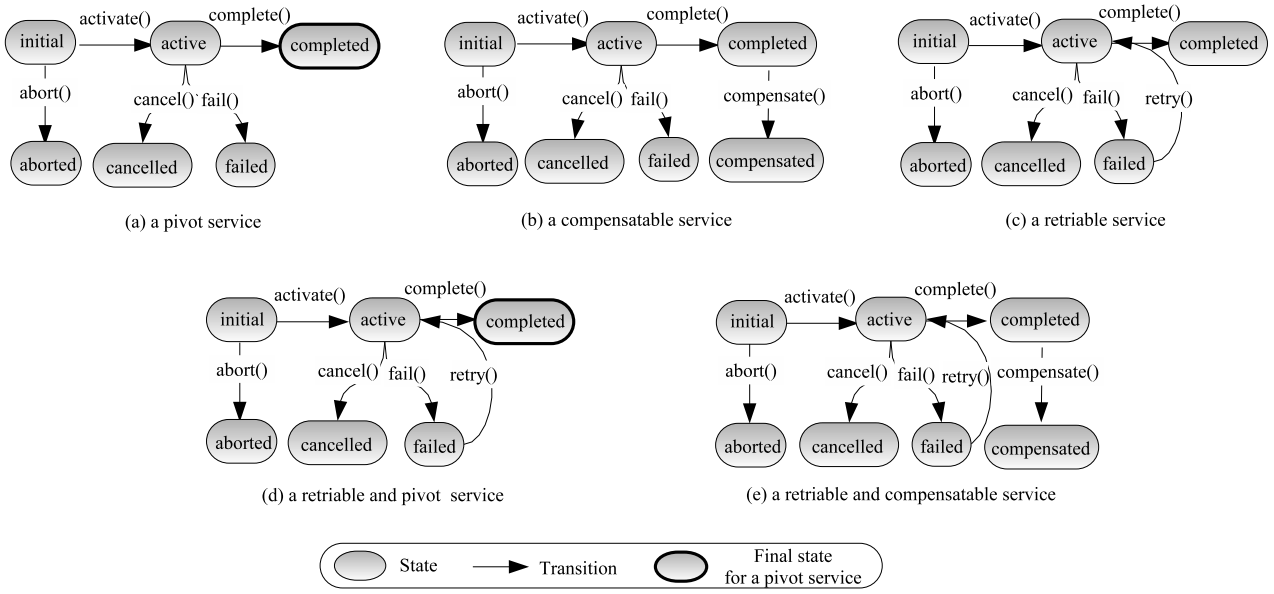


Figure 2: Services states/transitions diagrams according to different transactional properties.

fired by the service itself (the service agent). Internal transitions we are considering are *complete(), fail(), and retry()*.

4. COMPOSITION OF TRANSACTIONAL WEB SERVICES

A composite Web service is a conglomeration of existing Web services working in tandem to offer a new value-added service[13]. It coordinates a set of services as a cohesive unit of work to achieve common goals.

A Transactional Composite (Web) Service (TCS) emphasizes transactional properties for composition and synchronization of component Web services. It takes advantage of services transactional properties to specify mechanisms for failure handling and recovery.

4.1 Dependencies between services

A TCS defines services orchestration by specifying dependencies between services. They specify how services are coupled and how the behavior of certain service(s) influence the behavior of other service(s).

DEFINITION 4.1 (DEPENDENCY FROM s_1 TO s_2). *A dependency from s_1 to s_2 exists if a **transition** of s_1 can fire an external **transition** of s_2.*

A dependency defines for each external transition of a service a precondition to be enforced before this transition can be fired.

In our approach, we consider the following dependencies between services:

Activation dependency from s_1 to s_2: There is an activation dependency from s_1 to s_2 if the completion of s_1 can fire the activation of s_2.

We can tailor activation dependencies between services by specifying the activation condition, $ActCond(s)$, of each service s. $ActCond(s)$ defines the precondition to be enforced before the service s can be activated (only after the completion of other service(s)). There is an activation dependency from s_1 to s_2 iff $s_1.completed \in ActCond(s_2)$. Reciprocally for each service $s_1 \in ActCond(s_2)$, there is an activation dependency from s_1 to s_2 according to $ActCond(s_2)$.

For example, the composite services defined in figure 3 define an activation dependency from OI and PCC, to CA such that CA will be activated after the completion of OI and PCC. That means $ActCond(CA) = OI.completed \bigwedge PCC.completed$.

Alternative dependency from s_1 to s_2: There is an alternative dependency from s_1 to s_2 if the failure of s_1 can fire the activation of s_2.

We can tailor alternative dependencies between services by specifying the alternative condition, $AltCond(s)$, of each service s. $AltCond(s)$ defines the precondition to be enforced before the service s can be activated as an alternative of other service(s). There is an alternative dependency from s_1 to s_2 iff $s_1.failed \in AltCond(s)$. Reciprocally for each service $s_1 \in AltCond(s_2)$, there is an alternative dependency from s_1 to s_2 according to $AltCond(s_2)$

For instance the composite service cs_1 in figure 3.b defines an alternative dependency from DC_{Fed} to DC_{TNT} such that DC_{TNT} will be activated when DC_{Fed} fails. That means $AltCond(DC_{TNT}) = DC_{Fed}.failed$.

Abortion dependency from s_1 to s_2: There is an abortion dependency from s_1 to s_2 if the failure, cancellation or the abortion of s_1 can fire the abortion of s_2.

We can tailor abortion dependencies between services by specifying the abortion condition, $AbtCond(s)$, of each service s. $AbtCond(s)$ defines the precondition to be enforced before the service s can be aborted. There is an abortion dependency from s_1 to s_2 iff $s_1.aborted \in AbtCond(s_2)) \bigvee s_1.failed \in AbtCond(s_2) \bigvee s_1.cancelled \in AbtCond(s_2)$. Reciprocally for each service $s_1 \in AbtCond(s_2)$, there is an abortion dependency from s_1 to s_2 according to $AbtCond(s_2)$.

Compensation dependency from s_1 to s_2: There is

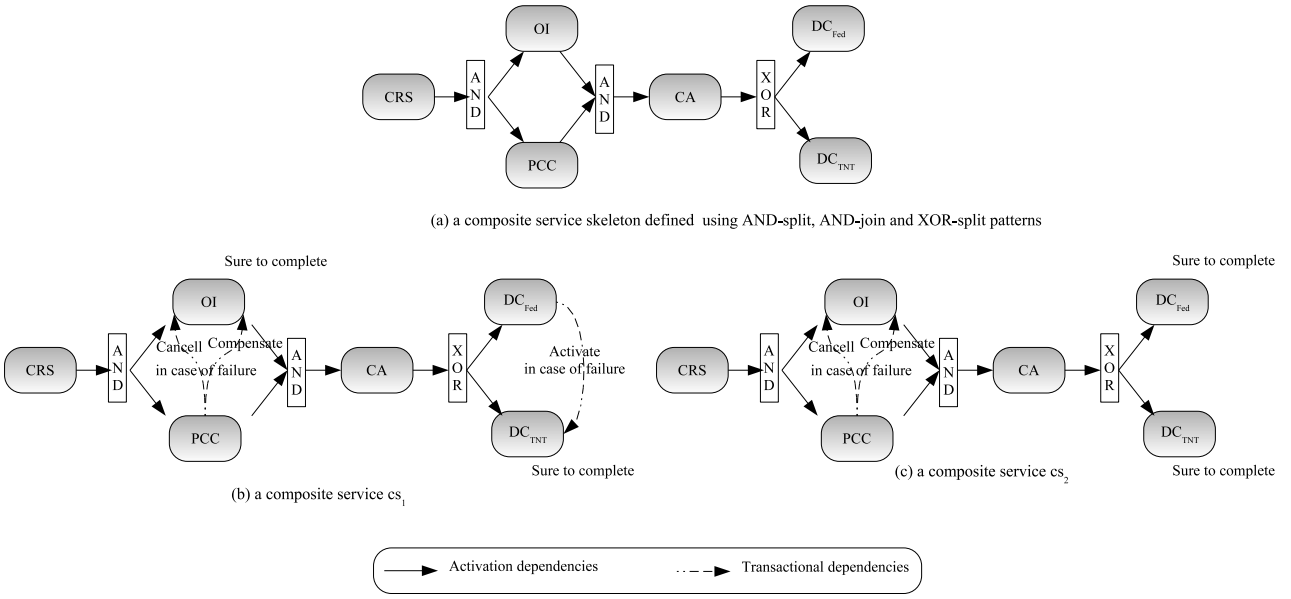


Figure 3: Two composite services defined according to the same skeleton.

a compensation dependency from s_1 to s_2 if the the failure or the compensation of s_1 can fire the compensation of s_2.

We can tailor compensation dependencies between services by specifying the compensation condition, $CpsCond(s)$, of each service s. $CpsCond(s)$ defines the precondition to be enforced before the service s can be compensated. There is a compensation dependency from s_1 to s_2 iff $s_1.failed \in CpsCond(s_2) \bigvee s_1.compensated \in CpsCond(s_2)$. Reciprocally for each service $s_1 \in CpsCond(S_2)$, there is a compensation dependency from s_1 to s_2 according to $CpsCond(s_2)$.

Composite services in figure 3 define a compensation dependency from PCC to OI such that OI will be compensated when PCC fails. That means $CpsCond(OI) = PCC.failed$.

Cancellation dependency from s_1 to s_2: There is a cancellation dependency from s_1 to s_2 if the failure of s_1 can fire the cancellation of s_2.

We can tailor cancellation dependencies between services by specifying the cancellation condition, $CnlCond(s)$, of each service s. $CnlCond(s)$ defines the precondition to be enforced before the service s can be cancelled. There is a cancellation dependency from s_1 to s_2 iff $s_1.failed \in CnlCond(s_2)$.

Reciprocally for each service $s_1 \in CnlCond(s_2)$, there is a cancellation dependency from s_1 to s_2 according to $CnlCond(s_2)$.

Composite services in figure 3 define a cancellation dependency from PCC to OI such that OI will be cancelled when PCC fails. That means $CnlCond(OI) = PCC.failed$.

For clarity reasons, we do not deal with abortion dependencies. We call transactional dependencies the compensation, cancellation and alternative dependencies.

4.2 Relations between dependencies

Dependencies specification must respect some semantic restrictions. Indeed, transactional dependencies depend on activation dependencies according to the following relations:

R_1 : An abortion dependency from s_1 to s_2 can exist only if there is an activation dependency from s_1 to s_2.

R_2 : A compensation dependency from s_1 to s_2 can exist only if there is an activation dependency from s_2 to s_1, or s_1 and s_2 execute in parallel and are synchronized.

R_3 : A cancellation dependency from s_1 to s_2 can exist only if s_1 and s_2 execute in parallel and are synchronized.

R_4 : An alternative dependency from s_1 to s_2 can exist only if s_1 and s_2 are exclusive.

Section 4.4 shows how these relations define potential dependencies induced by given activation dependencies.

4.3 Control and transactional flow of a TCS

Within a transactional composite service, we distinguish between the TCS control flow and the TCS transactional flow.

Control flow: The control flow (or skeleton) of a TCS specifies the partial ordering of component services activations. Activation dependencies between component services define the corresponding TCS control flow.

We use (workflow-like) patterns to define a composite service skeleton. As defined in [7], a pattern "is the abstraction from a concrete form which keeps recurring in specific non arbitrary contexts". A workflow pattern can be seen as an abstract description of a recurrent class of interactions based on (primitive) activation dependency. For example, the *AND-join* pattern [21] (see figure 3.a) describes an abstract services choreography by specifying services interactions as following: *a service is activated after the completion of several other services.*

Example: Figure 3.a illustrates a TCS skeleton defined using an AND-split, an AND-join and an XOR-split patterns.

Transactional flow: The transactional flow of a TCS specifies mechanisms for failures handling and recovery.

Transactional dependencies (like compensation, cancellation and alternative) define the TCS transactional flow.

4.4 Pattern's potential dependencies

Several TCSs can be defined based on a skeleton. Each TCS adopts the activation dependencies defined by the skeleton's patterns and may extend them by specifying additional transactional dependencies.

Example: Figure 3 shows two TCSs, cs_1 and cs_2, defined using the same skeleton. Each of these TCSs adopts this skeleton (figure 3.a) and refines it with an additional transactional flow.

Additional transactional dependencies are a subset of potential transactional dependencies defined by the skeleton's patterns. Indeed, a pattern defines in addition to the default activation dependencies, a set of potential transactional dependencies.

A *potential dependency* is a dependency that is not initially defined by the pattern but that can be added by TCSs using this pattern. Potential dependencies are directly related to the pattern's activation dependencies according to the relations we have introduced in section 4.2.

We have shown above that dependencies between services can be tailored by specifying preconditions on services' external transitions. And potential transactional dependencies are not an exception to this fact. So a TCS skeleton defines for each service the potential conditions corresponding to the potential dependencies. A pattern defines for each service s it is connected with:

- $ptCpsCond(s)$: its potential compensation condition,
- $ptAltCond(s)$: its potential alternative condition,
- $ptCnlCond(s)$: its potential cancellation condition.

We can write each of these conditions in exclusive disjunctive normal form. For instance, we can write the potential compensation condition of a service s as follows: $ptCpsCond(s) = \bigoplus_i ptCpsCond_i(s)$. Then $ptCpsCond_i(s)$ is one potential compensation condition of s.

Example: The TCS skeleton illustrated in figure 3.a uses an AND-join pattern to define activation dependencies between services OI, PCC and CA. According to the relation R_2 given in section 4.2, a TCS based on this skeleton can eventually specifies the following compensation dependencies: from OI to PCC, from PCC to OI, from CA to OI and from CA to PCC. Similarly, according to the relation R_3, this pattern defines the following potential cancellation dependencies: from OI to PCC, and from PCC to OI. That means, among other, that *ptCpsCond(PCC)=OI.failed* $\bigoplus$ *CA.failed* $\bigoplus$ *CA.compensated* and *ptCnlCond(OI)=PCC.failed*.

In the same way, according to the relation R_4, the XOR-split pattern connecting CA, DC_{Fed} and DC_{TNT} defines the following potential alternative dependencies: from DC_{TNT} to DC_{Fed} and from DC_{Fed} to DC_{TNT}.

That means that $ptAltCond(DC_{TNT}) = DC_{Fed}.failed$ and that $DC_{TNT}.failed = ptAltCond(DC_{Fed})$.

Finally, note that both TCSs cs_1 and cs_2 define their transactional flow as a subset of the potential transactional flow presented above. Both define compensation and cancellation dependencies from PCC to OI. cs_1 defines an alternative dependency from DC_{Fed} to DC_{TNT}.

5. FAILURE ATOMICITY REQUIREMENTS OF A TCS

Several executions can be instantiated according to the same TCS. The state of an instance of a TCS composed of n services is the tuple (x_1, x_2, ..., x_n), where x_i is the state of service s_i at a given time. The set of termination states of a TCS cs, $STS(cs)$, is the set of all possible termination states of its instances.

Back to our motivating example limited to the three services CRS, OI and PCC, we can have the following set of termination states: (*CRS.completed, OI.completed, PCC.completed*); (*CRS.completed, OI.failed, PCC.completed*); (*CRS.completed, OI.completed, PCC.failed*); (*CRS.compensated, OI.failed, PCC.initial*); (*CRS.compensated, OI.initial, PCC.failed*); (*CRS.compensated, OI.failed, PCC.failed*).

In order to express the designer's requirements for failure atomicity, we use the notion of *Accepted Termination States* ([18]). In other word, the concept of ATS represents our notion of correction.

Definition 5.1 (Accepted Termination States). *An accepted termination state, ats, of a composite service cs is a state for which designers accept the termination of cs. We define ATS the set of all Accepted Termination States required by designers.*

An execution is correct iff it leads the CS into an accepted termination state. A CS reaches an ats if (i) it completes successfully or (ii) it fails and undoes all undesirable effects of partial execution in accordance with designer failure-atomicity requirements [18].

Back to our example, a designer may choose the following ATS: *ATS(CS)={(CRS.completed, OI.completed, PCC.completed); (CRS.compensated, OI.failed, PCC.failed)}* that means that an execution is correct when all of the services complete, or when CRS is compensated (given the failure of OI and PCC). Obviously, we note that a composite service transactional behavior may vary according to The required ATS.

6. TRANSACTIONAL RULES

To explain the rules and to illustrate how they are working, we go back to our motivating example of personal computer online purchase. We suppose in addition that designers specify the ATS illustrated in the figure 5 to express their required failure atomicity.

Intuitively, the execution of a composite service can generate various termination states. A composite service is not valid if it exists some termination states that do not belong to the ATS specified by the designers.

Definition 6.1 (Validity according to an ATS). *A CS cs is said to be **valid** according to ATS* iff *its set of termination states is included in ATS, written* $STS(cs) \subseteq ATS$.

Example: The composite service cs_1 (illustrated in figure 3.b) is valid because $STS(cs_1) \in ATS$. However regarding the composite service cs_2 (illustrated in figure 3.c), we

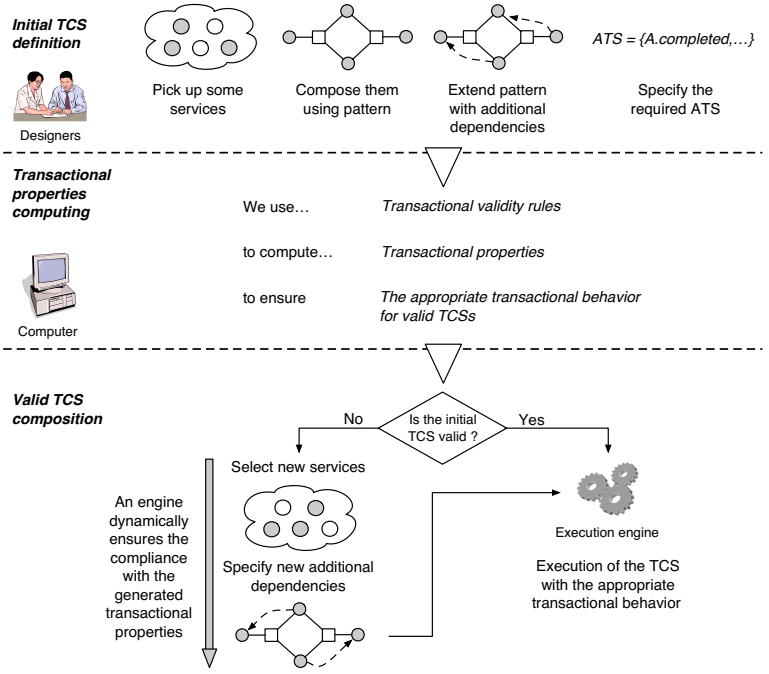


Figure 4: Objective and overview of our approach.

services / ats	CRS	OI	PCC	CA	DC_{Fed}	DC_{TNT}
ats_1	(completed,	completed,	completed,	completed,	initial,	completed)
ats_2	(completed,	completed,	completed,	completed,	completed,	initial)
ats_3	(completed,	compensated,	failed,	aborted,	aborted,	aborted)
ats_4	(completed,	failed,	compensated,	aborted,	aborted,	aborted)
ats_5	(completed,	cancelled,	failed,	aborted,	aborted,	aborted)
ats_6	(completed,	completed,	completed,	completed,	failed,	completed)

Figure 5: ATS used in our example of PC online purchase.

note that $STS(cs_2)$ contains the following termination state, $(CRS.completed, OI.failed, PCC.completed, CA.aborted, DC_{Fed}.aborted, DC_{TNT}.aborted)$, which is not an accepted termination state. Thereafter cs_2 is not valid.

6.1 Objective and overview

As illustrated in figure 4, our approach applies in a top-down manner.

Definition of an initial TCS: First, designers dynamically choose some available services and combine them to offer a new value added service. They compose the new service using a set of interactions patterns (sequence, AND-split, AND-join,...).

They can augment this skeleton by new dependencies selected from the potential dependencies. Then they express their required failure atomicity by specifying the required ATS.

Compute validity transactional properties: We use a set of rules, independent from skeletons and designers' ATS, to compute from the TCS skeleton and the required ATS a set of transactional properties.

These transactional properties tailor the appropriate transactional behavior for valid TCSs. A TCS must satisfy these transactional properties to be valid.

Definition of a valid TCS: If the initial TCS is not valid, designers can (i) select new services (with eventually new transactional properties) and (ii) augment the same skeleton with new dependencies. During this phase an engine assist designers to compose a valid TCS by respecting the generated transactional properties.

Once a valid TCS is reached, it can be deployed and executed.

6.2 Extracting services conditions

Tailoring the appropriate transactional behavior for valid composite services is equivalent to identify the appropriate

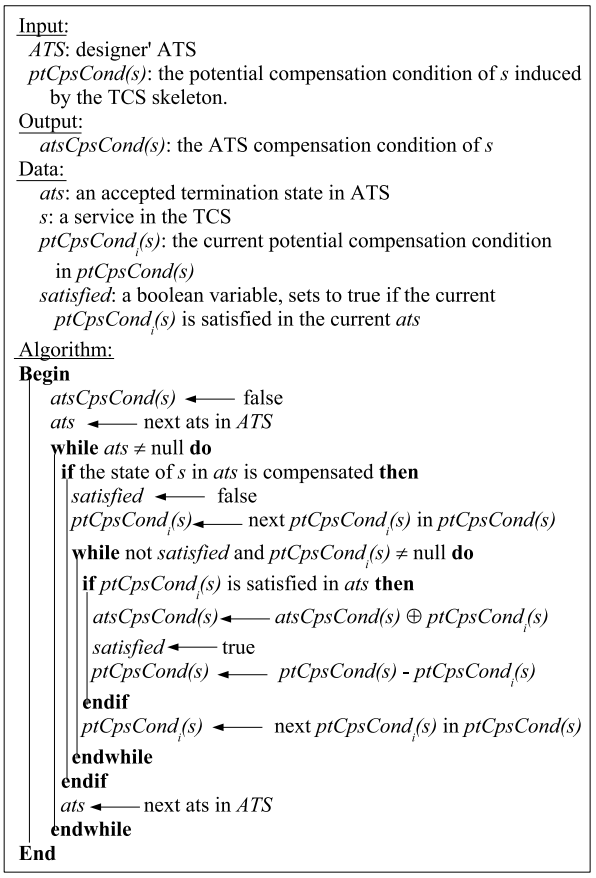


Figure 6: The algorithm for extracting ATS compensation conditions of a service s from the specified ATS and the TCS skeleton.

dependencies between services. We can deduce from the specified ATS and the TCS skeleton the services' conditions corresponding to these dependencies.

For each service s we distinguish (i) $atsCpsCond(s)$, the ATS compensation condition deduced from ATS, (ii) the ATS cancellation condition, $atsCnlCond(s)$, deduced from ATS, and (iii) $atsAltCond(s)$, the ATS alternative condition deduced from ATS. Below, we explain how we can deduce these conditions.

The algorithm given in figure 6 allows to extract the ATS compensation condition for a given service s from the composite service skeleton and the required ATS. The principle is: a potential compensation condition of s becomes an ATS compensation condition if it is satisfied in an $ats \in ATS$ such that the state of s in ats is compensated. We proceed similarly to deduce ATS alternative and cancellation conditions of each service.

For instance in our example, the potential compensation condition of PCC, $OI.failed$, becomes an ATS compensation condition because it is satisfied in $ats4$ (in which the state of PCC is compensated). And since $ats4$ is the only ats in which PCC is compensated then we can deduce that $atsCpsCond(PCC) = OI.failed$. Similarly we can extract the ATS cancellation condition for OI. $ats5$ is the only ats in which OI is cancelled. Furthermore, the potential cancellation condition of OI, $PCC.failed$ is satisfied in $ats5$. Then we can deduce that $atsCnlCond(s) = PCC.failed$. Finally, we can deduce in the same way, from $ats6$ and $ptAltCond(DC_{TNT})$, that $atsAltCond(DC_{TNT}) = DC_{Fed}.failed$.

It is important to note that the ATS specified by the designers must be consistent with the pattern semantics. An ATS is consistent if it satisfies the following two conditions.

First, each of its ats must be well-formed. An $ats \in ATS$ is not well-formed if it exists a service s such that none of its potential (or activation) conditions (corresponding to its state in ats) is satisfied (in ats). We can easily modify the previous algorithm to detect if it exists a not well-formed $ats \in ATS$.

Second, the set of all ats must be consistent. Such inconsistency can be detected after the generation of transactional properties ensuring CSs validity.

For instance given the following termination state (limited to the three services OI, PCC and CA) ts = *(OI.completed, PCC.compensated, CA.aborted)*, we note, among other, that none of the service PCC potential conditions ($OI.failed$, $CA.failed$ and $CA.compensated$) is satisfied in ts. So we can deduce that ts is not well-formed.

To illustrate an ATS inconsistency, let us consider the following ATS=$\{ats_1$ = *(OI.completed, PCC.completed, CA.completed)*; ats_2 = *(OI.completed, PCC.failed, CA.aborted)*; ats_3 = *(OI.compensated, PCC.failed, CA.aborted)*$\}$. Note that ats_1 and ats_2 are contradictory because the service OI once it is completed (in ats_2) and once it is compensated (in ats_3) for the same condition ($PCC.failed \bigwedge CA.aborted$). Thereafter the given ATS is not consistent although each of its ats is well-formed.

6.3 Transactional validity rules

An ATS also defines the accepted termination states of each component service. We denote $ATS(s)$ the set of accepted termination states of a component service s. Regarding our illustrative example, we can deduce, for instance, that *ATS(PCC)* = {*completed, failed, compensated*} and *ATS(CA)* = {*completed, aborted*}.

We can now introduce validity rules we are using to generate transactional properties that ensure validity (we suppose that $\Diamond$F means that F is eventually true):

$\forall s|$ *s is a component service in TCS*

1. $s.failed \notin ATS(s) \Longrightarrow$ generate the following transactional property TP_s^r : *s must be retriable*
2. $s.compensated \notin ATS(s) \Longrightarrow$ generate the following transactional property TP_s^p : *there is no need for s to be compensatable*
3. $\forall atsCpsCond_i(s) \in atsCpsCond(s)$, generate the following transactional property $TP_s^{cp_i}$: $\Diamond(atsCpsCond_i(s)) \Longrightarrow$
   (a) s must be compensatable and
   (b) $atsCpsCond_i(s) \in CpsCond(s)$.
4. $\forall atsCnlCond_i(s) \in atsCnlCond(s)$, generate the following transactional property $TP_s^{cl_i}$: $\Diamond(atsCnlCond_i(s)) \Longrightarrow$ $atsCnlCond_i(s) \in CnlCond(s)$.

5. $\forall atsAltCond_i(s) \in atsAltCond(s)$, generate the following transactional property $TP_s^{at_i}$: $\Diamond(atsAltCond_i(s)) \Longrightarrow atsAltCond_i(s) \in AltCond(s)$.

The first rule postulates that if the state *failed* does not belong to the ATS of s, then it exists a transactional property saying that s must be *retriable*.

The second rule postulates that if the state *compensated* does not belong to the ATS of s, then it exists a transactional property saying that there is no need for s to be compensatable.

The third rule postulates that for each ATS compensation condition of s, $atsCpsCond_i(s)$, it exists a transactional property saying that: if this condition is eventually true then s must be compensatable and $atsCpsCond(s)$ becomes a compensation condition of s. That means $\forall s' \in atsCpsCond_i(s)$ add a compensation dependency from s' to s according to $atsCpsCond_i(s)$.

The fourth rule postulates that for each ATS cancellation condition of s, $atsCnlCond_i(s)$, it exists a transactional property saying that: if this condition is eventually true then $atsCnlCond(s)$ becomes a cancellation condition of s. That means $\forall s' \in atsCnlCond_i(s)$ add a cancellation dependency from s' to s according to $atsCnlCond_i(s)$.

The fifth rule postulates that for each ATS alternative condition of s, $atsAltCond_i(s)$, it exists a transactional property saying that: if this condition is eventually true then $atsAltCond(s)$ becomes an alternative condition of s. That means $\forall s' \in atsAltCond_i(s)$ add an alternative dependency from s' to s according to $atsAltCond_i(s)$.

Example Back to our example, we can compute the following transactional properties: $\mathcal{TPV}(ATS, CSskeleton) = \{\boldsymbol{TP}^r_{CRS}, \boldsymbol{TP}^r_{CA}, \boldsymbol{TP}^r_{DC_{TNT}}, \boldsymbol{TP}^p_{CA}, \boldsymbol{TP}^p_{CRS}, \boldsymbol{TP}^p_{DC_{TNT}}, \boldsymbol{TP}^p_{DC_{Fed}}, \boldsymbol{TP}^{cp_1}_{PCC}, \boldsymbol{TP}^{cp_1}_{OI}, \boldsymbol{TP}^{cl_1}_{OI}, \boldsymbol{TP}^{at_1}_{DC_{TNT}}\}$

- By applying the first rule and since the state *failed* does not belong to $ATS(CRS)$ we get the transactional property $\boldsymbol{TP}^r_{CRS}$: *CRS must be retriable.* Similarly, we can compute the following transactional properties: $\boldsymbol{TP}^r_{CA}$: *CA must be retriable* and $\boldsymbol{TP}^r_{DC_{TNT}}$: DC_{TNT} *must be retriable.*

- By applying the second rule and since the state *compensated* does not belong to $ATS(CA)$ we get the transactional property $\boldsymbol{TP}^p_{CA}$: *there is no need for CA to be compensatable.* Similarly we can compute the following transactional properties: $\boldsymbol{TP}^p_{CRS}$, $\boldsymbol{TP}^p_{DC_{TNT}}$, and $\boldsymbol{TP}^p_{DC_{Fed}}$: *there is no need for CRS,* DC_{TNT}, DC_{Fed} *to be compensatable.*

- By applying the third rule and since *atsCpsCond(PCC) = OI.failed* we get the transactional property $\boldsymbol{TP}^{cp_1}_{PCC}$: $\Diamond(OI.failed)$ (means that OI is not retriable) $\Longrightarrow$
  (a) *PCC must be compensatable* and
  (b) *PCC must be compensated when OI fails.*

- By applying the third rule and since *atsCpsCond(OI) = PCC.failed* we get the transactional property $\boldsymbol{TP}^{cp_1}_{OI}$: $\Diamond(PCC.failed)$ (means that PCC is not retriable) $\Longrightarrow$
  (a) *OI must be compensatable* and
  (b) *OI must be compensated when PCC fails.*

- By applying the fourth rule and since *atsCnlCond(OI) = PCC.failed* we get the transactional property $\boldsymbol{TP}^{cl_1}_{OI}$: $\Diamond(PCC.failed)$ (means that PCC is not retriable) $\Longrightarrow$ *OI must be cancelled when PCC fails.*

- By applying the fifth rule and since $DC_{Fed}.failed =$ *atsAltCond(*DC_{TNT}*)* we get the transactional property $\boldsymbol{TP}^{At_1}_{PCC}$: $\Diamond(DC_{Fed}.failed)$ (means that DC_{Fed} is not retriable) $\Longrightarrow$ DC_{TNT} *must be activated when* DC_{FED} *fails.*

The composite service cs_1 verifies all the validity rules and thereafter it is valid. However the composite service cs_2 verifies all the validity rules except the $R^{cp_1}_{PCC}$ rule. This rule postulates that if the compensation condition of PCC (which is the failure of OI) is eventually true (which is the case in cs_2) then PCC must be compensatable and must be compensated when OI fails (which is not the case in cs_2). The composite service cs_1 respects this rule since OI is sure to complete and thereafter never fails.

6.4 Validity rules proof

We use the following lemma (the proof is not shown due to lack of space).

Lemma A TCS termination state ts is not an accepted termination state iff $\exists$ a service s such that

- the termination state of s in $ts \notin ATS(s)$ or
- none of its ATS potential conditions (corresponding to its state in ts) is satisfied (in ts).

Proving that: (cs satisfies all validity rules $\Longleftrightarrow$ cs is valid) is equivalent to proof that: (cs is not valid $\Longleftrightarrow$ $\exists$ a rule such thats cs does not satisfy this rule).

1) $\Longrightarrow$: cs is not valid means that it has a not valid termination state. That means (using the lemma above) either (a) it exists a service s which terminates in a not valid state or (b) it exists a service s which terminates in a valid state but without satisfying one of its ATS conditions corresponding to this termination state. (a) means that cs does not verify the validity rules 1 or 2. (b) means that cs does not verify one of the validity rules 3, 4 or 5.

2) $\Longleftarrow$: (a) If cs does not verify one of the validity rules 1 or 2 then it exists a service s which will terminate in a non valid termination state. (b) if cs does not verify one of the validity rules 3, 4 or 5 then it exists a service s which will terminate in a valid state without satisfying none of its corresponding ATS conditions. (a) and (b) means that (using the lemma above) cs is not valid.

6.5 Implementation

We are currently developing a prototype that supports this work. Our prototype is written in Java.

The first part of the prototype is the transactional engine. It allows the user to select the services (with transactional properties), to define the TCS skeleton using patterns, and to specify the required ATS. The engine uses the transactional rules to compute the appropriate transactional properties for valid TCSs. Then, it assists designers to compose a valid TCS by respecting these transactional properties.

Window 1 of figure 7 shows how designers can choose services from the "Web services" scroll panel. It typically shows the transactional properties of the chosen service.

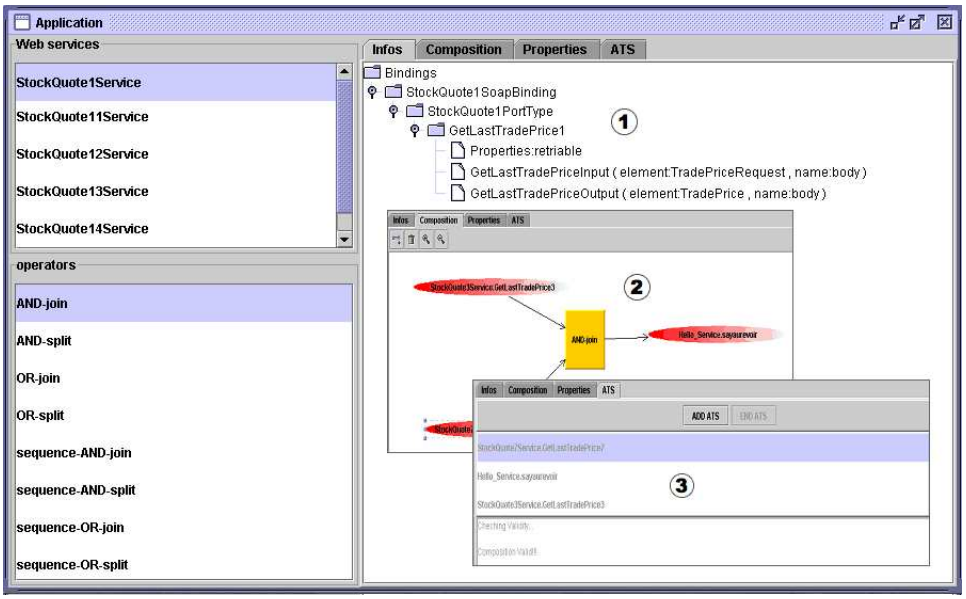


Figure 7: A screen shot illustrating the application of our transactional approach.

Window 2 of figure 7 illustrates how designers can specify the TCS skeleton using patterns from the "operators" panel.

Finally, window 3 of figure 7 illustrates how the transactional engine computes the appropriate transactional properties from the required ATS.

The second part of the prototype is a workflow engine that is able to execute the composite service. Our workflow engine is Bonita, a workflow engine supported by the Object Web consortium ([20]). Bonita is a cooperative workflow system supporting the specification, the execution, the monitoring, and the coordination of the processes. The main features of Bonita are: a third-generation workflow engine that can be **parameterized by an activity model**, a web interface to control workflow processes (accessing workflow methods as J2EE-based web services), an implementation using J2EE Enterprise Java Beans, the possibility to execute code in the server side for different events (e.g. start and cancel activities) by means of hooks (hooks can be for instance Java programs, and may be assigned to process and node events), and the availability of a graphical user interface to design and control workflow processes, based on Java JFC/Swing. Of course, for our concern, the most interesting feature is related to the ability to define a specific model of services, including transactional states.

7. RELATED WORK

Advanced Transaction Models (ATMs) have been proposed to support new database applications by relaxing transaction isolation and atomicity to better match the new requirements. As workflows in the past, services composition requirements either exceed or significantly differ from those of ATMs [6] in terms of modelling, coordination [22] and transactional requirements. Their limitations come mainly from their inflexibility to incorporate different transactional semantics as well as different behavioral patterns into the same structured transaction [8]. To overcome these limitations, [18] proposed a transactional Workflows system supporting multitask, multisystem activities where: (a) different tasks may have different execution behaviors or properties, (b) application or user defined coordination of the different tasks, and (c) application or user defined failure and execution atomicity are supported. In this approach, failure atomicity requirement is defined by specifying a set of ATS. Unfortunately, no transaction management support is provided to ensure this correctness criteria. Accepted termination states as a mean to relax atomicity has been discussed in many previous works [1, 5, 18]. In fact, ATS property has been always implicitly included in most of transactional models. For example, atomicity property implicitly defines ATS for traditional transactions; *all* (success state) and *nothing* (correct failure state). Also, when an advanced transaction model specifies global transaction structure, sub transactions properties, inter sub transaction dependencies, mechanisms of handing-over, success and failure criteria, and so on, it implicitly defines its ATS. In the same way, when [19, 23] define rules to form a well defined flexible transaction, they implicitly define the appropriate ATS for flexible transaction model.

Emerging standards such as BTP [15], WS-transaction (WS-AtomicTransaction and WS-BusinessActivity [11, 12]), and WS-TXM (Acid, BP, LRA)[3] define models to support a two-phase coordination of web services. These proposals are based on a set of extended transactional models to specify coordinations between services. Participants agree to a specific model before starting interactions. Then the corresponding coordination layer technologies support the appropriate messages exchange according to the chosen transactional model. These propositions inherit the extended transactional models rigidity. Besides, there is a potentiel problem of transactional interoperability between services implemented with different approaches. Our approach can complement these efforts and overcome these two gaps. Indeed, our approach allows for reliable, more complex, and

more flexible compositions. In addition, it can coordinate services implemented with different technologies since we use only services transactional features (and not interested in how they are implemented). So, we can use our approach to specify flexible and reliable composite services, while component services can be implemented by one of the above technologies. Once a valid TCS is reached, it can be considered as a coordination protocol and can be plugged in one of the existing coordination technology to be executed.

8. CONCLUSION

In this paper, we have proposed a transactional approach for reliable Web services compositions by ensuring the failure atomicity required by the designers.

Contrary to ATMs, our approach follows the opposite direction by starting from designers requirements to provide correctness rules. Like in [9, 18] (for transactions), designers define the global composite service structure, using patterns, and specifies required ATS as a correctness criteria. Then, we use a set of transactional rules to assist designers to compose a valid CS with regards to the specified ATS.

The main contribution of our approach is that is able to incorporate different interactions patterns into the same structured transaction, and besides it can validate CSs according to designers transactional requirements.

Acknowledgment: We would like to thank Laura Lozano for its implementation efforts.

9. REFERENCES

[1] M. Ansari, L. Ness, M. Rusinkiewicz, A.Sheth. Using Flexible Transactions to Support Multi-System Telecommunication Applications. In *Proc. of the 18th VLDB Conference*, 65-76, August 1992.

[2] B. Benatallah, F. Casasti, F. Toumani. Web Service Conversation Modeling: A Cornerstone for E-Business Automation. In *IEEE Internet Computing*, 8(1), 46-54, January/February 2004.

[3] D. Bunting and al. Web Services Transaction Management (WS-TXM) Version 1.0. Arjuna, Fujitsu, IONA, Oracle, and Sun, July 28, 2003.

[4] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana. Unraveling the Web services web: an Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 6(2), 86-93, March/April 2002.

[5] A. Elmagarmid, Y. Leu, W. Litwin, M. Rusinkiewicz. A multidatabase transaction model for Interbase. In *Proc. of the 16th VLDB Conference*, Brisbane, Australia 1990.

[6] A. Elmagarmid, (Ed.). Transaction Models for Advanced Database Applications. Morgan-Kaufmann, 1992.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley, Reading, Massachussets, 1995.

[8] N. Gioldasis, S. Christodoulakis. UTML: Unified Transaction Modeling Language. In *Third International Conference on Web Information Systems Engineering (WISE'02)*, 115-126, IEEE Computer Society, December 2002.

[9] N. Krishnakumar, A. Sheth. Managing Heterogenous Multi-system Tasks to Support Enterprise-wide Operations.. Distributed and Parallel Databases, 3(2), 155-186. Kluwer Academic Publishers, 1995.

[10] D. Langworthy and al. Web Services Coordination (WS-Coordination). BEA, IBM, Microsoft, 2003.

[11] D. Langworthy and al. Web Services Atomic Transaction (WS-AtomicTransaction). BEA, IBM, Microsoft, 2003.

[12] D. Langworthy and al. Web Services Business Activity Framework (WS-BusinessActivity). BEA, IBM, Microsoft, 2004.

[13] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, A. K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. VLDB Journal: The International Journal on Very Large Data Bases, 12(4), 59-85, April 2003.

[14] S. Mehrotra, R. Rastogi, A. Silberschatz, H. Korth. A transaction model for multidatabase systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems (ICDCS92)* (Yokohama, Japan), IEEE Computer Society Press, 56-63, June 1992.

[15] OASIS Committee Specification. Business Transaction Protocol, Version 1.0 (June 2002).

[16] P. F. Pires. WebTransact: A Framework For Specifying And Coordinating Reliable Web Services Compositions. *Technical report ES-578/02*, Coppe federal university of Rio De Janeiro, Brazil, April 2002.

[17] P. F. Pires, M. R. Benevides, M. Mattoso. Building Reliable Web Services Compositions. In *Web, Web-Services, and Database Systems*, LNCS 2593, 59-72, Springer, 2003.

[18] M. Rusinkiewicz, A. Sheth. Specification and Execution of Transactional Workflows. In *Modern Database Systems: The Object Model, Interoperability, and Beyond.*, W. Kim Ed., ACM Press and Addison-Wesley, 1995.

[19] H. Schuldt, G. Alonso, C. Beeri, H.J. Schek. Atomicity and isolation for transactional processes. In *ACM Transactions on Database Systems*, 27(1), 63-116, March 2002.

[20] M. Valdès, F. Charoy. Bonita: Workflow Cooperative System. ObjectWeb consortium, http://bonita.objectweb.org/, 2004.

[21] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros. Advanced Workflow Patterns. In *the 7th International Conference on Cooperative Information Systems (CoopIS 2000)*, LNCS 1901, 18-29. Springer-Verlag, Berlin, 2000.

[22] D. Worah, A. Sheth. Transactions In Transactional Workflows. In *Advanced Transaction Models and Architectures*, S. Jajodia and L. Kerschberg Eds., chapter 1, pages 3-45. Kluwer Academic Publishers, 1997.

[23] A. Zhang, M. Nodine, B. Bhargava, O. Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. In *Proc. ACM SIGMOD International Conference on Management of Data*, 22(3), 67-78, 1994.

Chapitre 10

Timed Specification For Web Services Compatibility Analysis

Auteurs : Nawal Guermouche, Olivier Perrin, Christophe Ringeissen

Référence : Proceedings of WWV 2007.

Abstract Web services are becoming one of the main technologies for designing and building complex inter-enterprise business applications. Usually, a business application cannot be fulfilled by one Web service but by coordinating a set of them. In particular, to perform a coordination, one of the important investigations is the compatibility analysis. Two Web services are said compatible if they can interact correctly. In the literature, the proposed frameworks for the services compatibility checking rely on the supported sequences of messages. The interaction of services depends also on other properties, such that the exchanged data flow. Thus, considering only supported sequences of messages seems to be insufficient. Other properties on which the services interaction can rely on, are the temporal constraints. In this paper, we focus our interest on the compatibility analysis of Web services regarding their (1) supported sequences of messages, (2) the exchanged data flow, (3) constraints related to the exchanged data flow and (4) the temporal requirements. Based on these properties, we study three compatibility classes : (i) absolute compatibility, (ii) likely compatibility and (iii) absolute incompatibility.

Timed Specification For Web Services Compatibility Analysis

Nawal Guermouche[1]

LORIA, INRIA Lorraine, Campus scientifique
BP 239, 54506 Vandoeuvre-Lès-Nancy

Olivier Perrin[2]

LORIA, INRIA Lorraine, Campus scientifique
BP 239, 54506 Vandoeuvre-Lès-Nancy

Christophe Ringeissen[3]

LORIA, INRIA Lorraine, Campus scientifique
BP 239, 54506 Vandoeuvre-Lès-Nancy

Abstract

Web services are becoming one of the main technologies for designing and building complex inter-enterprise business applications. Usually, a business application cannot be fulfilled by one Web service but by coordinating a set of them. In particular, to perform a coordination, one of the important investigations is the compatibility analysis. Two Web services are said *compatible* if they can interact correctly. In the literature, the proposed frameworks for the services compatibility checking rely on the supported sequences of messages. The interaction of services depends also on other properties, such that the exchanged data flow. Thus, considering only supported sequences of messages seems to be insufficient. Other properties on which the services interaction can rely on, are the temporal constraints. In this paper, we focus our interest on the compatibility analysis of Web services regarding their (1) supported sequences of messages, (2) the exchanged data flow, (3) constraints related to the exchanged data flow and (4) the temporal requirements. Based on these properties, we study three compatibility classes: (i) *absolute compatibility*, (ii) *likely compatibility* and (iii) *absolute incompatibility*.

Keywords: compatibility analysis, Web services, temporal constraints.

1 Introduction

Web services are one of the main technologies adopted by organizations to build their business applications. Generally, a Web service is a set of related functionalities that can be programmatically accessed through the Web. These functionalities

[1] Email: Nawal.Guermouche@loria.fr
[2] Email: Olivier.Perrin@loria.fr
[3] Email: Christophe.Ringeissen@loria.fr

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

represent the different operations made available by the Web service and are described in its service description using the WSDL standard language [10]. Autonomy and heterogeneity of services induce several interoperability problems. The interoperability is the key for several issues such as the service composition which involves the synthesis problem (i.e., how to coordinate the component services to fulfill a given goal) [1]. In this context, the *compatibility analysis* plays a crucial role. This analysis consists in checking if a client and a provider can properly interact.

The interaction between a client and a provider depends not only on functionalities the provider can supply, but also on other properties representing requirements of each of them. Hence, service descriptions must be augmented by all the aspects that can represent the client's and the provider's requirements. Such enriched descriptions are useful to keep services informed if they can interact together by considering their requirements. For the best of our knowledge, all the related compatibility checking frameworks consider only the supported messages sequences [4,8]. Other works consider some forms of temporal constraints that cannot express all the service's temporal requirements [3].

Since the exchanged messages can involve data, considering only the supported sequences of messages seems not sufficient. Thus, considering data flow in Web services analysis, and especially in the compatibility analysis is important. Moreover, to receive or send a message, a Web service may express some requirements. Among these requirements, we focus on data and time related conditions. For example, a service that allows us to book a flight ticket (TB service) can supply an electronic ticket if the *credit card number ccn* of the client *is valid*. Moreover, the TB service cannot send the electronical ticket before 60 minutes starting from the payment.

Regarding the temporal aspects, a service can have its own requirements associated to its internal running. Such requirements are called *internal constraints*. For example, if a client that books a flight ticket does not send its credit card number (*ccn*) within 30 minutes, the service cancels. The internal constraints can be, implicitly, mutually dependent. Thus implicit incompatibilities can arise. To cater for such problematical issue, we propose to infer from the internal constraints, the potential implicit constraints. Those inferred constraints can be exposed to the other services, thus we call such constraints *external constraints*. For example, once the client sends its *ccn*, the provider must supply the electronic ticket within 1 hour.

In this paper, we present a timed compatibility analysis framework that deals with the *implicit incompatibilities* that can arise. In our approach, Web services are modeled as finite state machines. This kind of formal representation has been already used in a series of papers [9,3,5,15]. The model we consider in this paper integrates multiple aspects to have a fine-grained abstraction of Web services [13]. Theses aspects are (a) *operations*, which can require input parameters and provide output parameters, (b) *message* passing exchanged via channels able to hold at most one message at a time, (c) *data* that are parameters of the messages and operations, (d) constraints over these data, (e) temporal constraints, and (f) the behavior of Web services (which may involve multiple operations and message-passing activities) is specified using guarded finite state transition systems. The data used

by Web services are managed via information systems. Services can get data by exchanging messages or by reading the information systems. Furthermore, the services can change the values of data in the information systems. This allows us to deal with constraints over counters. For example, if the client provides a wrong *ccn* successively three times, beyond the third attempt the system cancels. To express such requirements, we may consider counters that can be *incremented*, *decremented* or *reset* once a transition is triggered. To express temporal constraints, we rely on clocks as defined in standard timed automata [2].

The paper first recalls the model we rely on in Section 2. Section 3 presents the motivating example used to illustrate our model. The different forms of temporal constraints we define are described in Section 4. In Section 5, we show how we integrate the different forms of temporal constraints by extending to the Web Services Timed Transitions Systems (WSTTS) [15]. In section 6, we present the compatibility analysis process of Web services regarding their different constraints (constraints over data and temporal constraints). Related work is introduced in Section 7, and Section 8 concludes.

2 Overview

In this paper, we enhance with temporal constraints the model of Web services automatic composition introduced in [13]. In the following section, we introduce the model we proposed.

2.1 The model

In our earlier framework [13], Web services are represented as conversational automata. In these automata, the alphabet consists of actions and exchanges of messages. A message is either sent or received. An output message is denoted by $!m$, whilst an input one is denoted by $?m$. A message containing a list of parameters is denoted by $m(d_1, \ldots, d_n)$, or $m(\bar{d})$ for short. There are many papers dealing with conversational automata to represent web services, including the *Colombo* model [6]. Our model extends the Roman model [7,5] by introducing communication and data capabilities, and can be considered as an extension of the conversational model introduced in [9], where data are exchanged via messages, and can be used to express guards of transitions. An action a with a list of input parameters $d_1, \ldots, d_m$ and list of output parameters $d'_1, \ldots, d'_n$ is denoted by $a(d_1, \ldots, d_m; d'_1, \ldots, d'_m)$. Throughout the paper, we only consider "well-formed" Web services, in which we assume that, when a parameter is used, possibly in a guard, it has been previously initialized. There are three ways to initialize a parameter: either it is the output of a previous action, it is retrieved via a web service exchange of message, or via reading the information system. Constraints used in guarded transitions can be defined as conjunctions of atomic formulas $d \bowtie val$, where val is a possible value for the parameter d, and $\bowtie$ is binary relation to compare the value of d and val.

Definition 2.1 A *Web service* Q is a tuple (S, s_0, F, M, A, C, T) where S is a set of states, s_0 is the initial state, F is a set of final states ($F \subseteq S$), M is a set of

messages, A is a set of actions, C is a set of constraints, and the set of transitions $T \subseteq S \times (M^{?!} \cup A) \times C \times S$, such that, from a source state, the service exchanges a message or performs an action in A, with a constraint to reach a target state, where $M^{?!} = \{?m | m \in M\} \cup \{!m | m \in M\}$ is the set of input/output messages. An input (resp. output) transition is a transition of the form $(\_, ?m, \_)$ (resp. $(\_, !m, \_)$).

3 Motivating example

To illustrate our model, we consider a simple application for an online flight ticket booking. This application is carried out by two Web services: the *ticket booking service* (TB) and the *payment by credit card* service (PCC). The requester specifies the destination (city) and the date of the travel. Then the TB service suggests a list of tickets description (price, class, timing). The requester makes a choice, and he/she provides a credit card number *ccn* to complete the booking. Then the TB service invokes the (PCC) service to check the credit card number (*ccn*) validity in order to guarantee the payment.

The TB service has several requirements such as:

(i) once TB service sends the description list of tickets, if the client does not confirm the travel booking by providing his *ccn* in 1800 seconds, the ticket booking is cancelled.

(ii) if the client provides a wrong *ccn* successively three times, beyond the third attempt the system cancels.

(iii) if the client books a ticket during the *Christmas holiday*, i.e., in the period [25 *december*, 05 *january*], TB service performs a rebate on the price of tickets.

To establish a conversation between the two services, requirements that are strongly bound to temporal features must be checked. Thus, we believe that Web services description interfaces (for example WSDL interface) must be enhanced by *time-related properties*. The time-related properties can have several forms. In the following sections, we detail the issues related to such properties.

4 Temporal constraints

From our point of view, each service can have *time-related requirements* to express the internal process of the service. Such requirements are called *internal constraints*. In order to establish an interaction between a client service and a provider service, each one of them exposes to the other requirements that must be checked before initializing the interaction. Such exposed requirements are called *external constraints*.

4.1 Internal constraints

The internal constraints are specified when the Web service is designed. They relate for example to the temporal requirements needed to exchange messages and fulfill an operation. Especially, an internal constraint allows to express the fact that triggering a transition may depend on other timed transitions. In our work, we distinguish two kinds of internal constraints: (i) *activation constraints* that

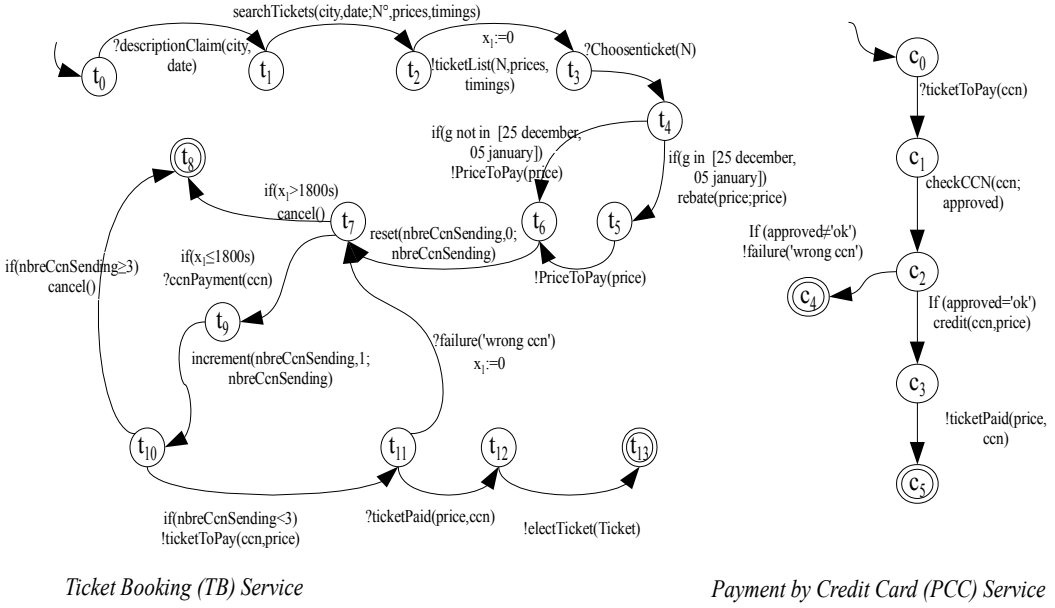


Fig. 1. The online flight booking ticket.

correspond to the cancellation transitions and (ii) *dependency constraints* used to specify requirements related to ordinary transitions (i.e., non-cancellation ones).

Example 4.1 To illustrate an activation constraint over a sequence of transitions, we consider the requirement (i) of the motivating example which is: *once the TB service sends the description list of tickets, if the client does not confirm the travel booking by providing his ccn in 1800 seconds, the tickets booking will be cancelled.*

To model such feature, we use the standard clocks of timed automata [2]. As seen in Fig. 1, we reset a clock x_1 with the transition that sends the ticket description ($!ticketList(N, price, timing)$), and we guard the transition that cancels the booking with a temporal constraint $x_1 > 1800s$.

Furthermore, an internal constraint can be specified over a period that can be expressed via a global clock providing the absolute time. An example of such constraints is presented by the requirement (iii): *if the client books a ticket during the christmas holiday, i.e., the period* $[25\ december, 05\ january]$*, TB service performs a rebate on the price of tickets.* To be able to express such time-related requirements, we propose to use temporal constraints over a *global clock*, which cannot be reset in transitions.

Let X be a set of clocks. The set of *constraints* over X, denoted $\Psi(X)$, is defined as follows:

true $\mid x \bowtie c \mid \psi_1 \wedge \psi_2$, where $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, $x \in X$, and c is a constant.

The internal constraints can also have another form, such as depending on the number of times[4] transitions can be triggered. Back to the requirement (ii) of the motivating example, the system must cancel after the third wrong attempt

[4] carrying out an operation or exchanging a message a specific number of times

to provide the *ccn*. To consider such a feature in the Web services model, we propose to enhance the Web services specification with constraints over *counters*. To manage such constraints and for the sake of simplicity of the model, we propose to consider counters as data that can be stored in the information system. Those data (counters) can be *incremented*, *decremented* and *reset*.

To summarize, internal constraints are expressed via (1) *local clocks* used in timed automata theory [2] to specify the relative period in which a transition must be triggered, and (2) a *global clock* to specify the absolute date (or period) in which a transition must be triggered, and (3) constraints over data such that constraints over *counters* to specify the number of times in which a transition must be triggered.

The internal constraints related to the cancellation transitions are called *activation constraints*, whilst those related to non-cancellation transitions are called *dependency constraints*.

4.2 External constraints

To initialize a conversation between a client service and a provider service, each one can expose requirements that must be checked. As presented above, each service has its internal constraints. So we propose to infer from those internal constraints the external constraints, that can be exposed to the other services. For example, we can infer from the internal constraints of the *TB* service, depicted in Fig. 1, that TB cancels after at least $1800s$.

In Section 6.2, we illustrate how we infer external constraints from internal ones, and we will show the importance of the inference of the external constraints in the compatibility checking.

An external constraint can be defined as: *Once pre-conditions are satisfied, an input/output message must be performed in a given period.*

Definition 4.2 An external constraint e is a tuple (p, ϱ, d), where ϱ denotes an input/output message in $M^{?!}$, p is the pre-condition that must hold to perform ϱ, such that d is the period in which ϱ can be performed. If ϱ is an input (resp. output) message, the external constraint is called an *input* (resp. *output*) external constraint.

In the following, we restrict our attention to two forms of preconditions:

- A precondition equals to *true*, which means that ϱ must be performed in a period starting from the invocation of the service (see Section 6.2.2).
- A precondition equals to an *input/output* message, which means that ϱ must be performed in a period starting from the exchange of this message (see Section 6.2.1).

Example 4.3 The client can claim to receive the electronic ticket not later than 60 minutes after sending his *ccn*.

Once the CCN is sent ($!ccnPayment(ccn)$*), the electronic ticket must be received (*$?electTicket(ticket)$*) in* 60 *minutes (*$[0, 3600s]$*).*

In this example, the related external constraint of the client is

$(!ccnPayment(ccn),\ ?electTicket(ticket),\ [0, 3600s])$.

To summarize, as shown in Fig. 2, we distinguish (1) *internal constraints* and (2) *external constraints*. Internal constraints can be specified by (i) *temporal constraints* and (ii) *constraints over data*. The temporal constraints can be specified over *local clocks* and *a global clock*. The local clocks rely on standard timed automata clocks [2], whereas, the global clock relies on an absolute time. The constraints over data involve *parameters* of the exchanged messages, or *counters*. The internal constraints specify the activation constraints (cancellation) and *dependency* constraints (non-cancellation). The external constraints are constraints exposed by the client and the provider service, that must be checked before initializing the interaction. Those external constraints are inferred from the internal constraints (see Section 6.2).

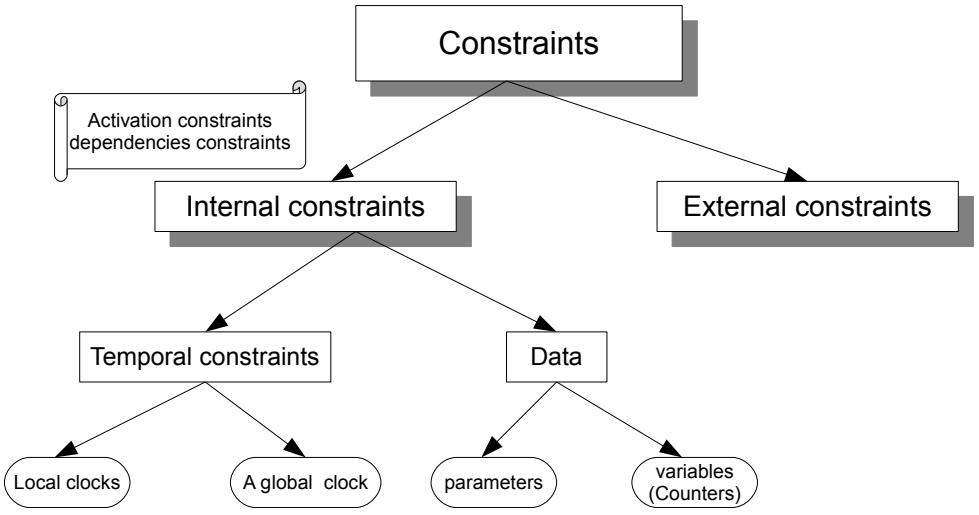


Fig. 2. Constraints specification.

5 Web services timed conversations

In this section, we show how we integrate temporal constraints presented above into web service specifications. To this aim, we propose to extend Web Service Timed Transition Systems (WSTTS) which are timed conversational automata [15]. A WSTTS is a finite state machine equipped with a set of clock variables and transitions guarded by constraints over clock variables. WSTTS use the standard form of constraints used in timed automata [2]. To consider constraints presented previously, WSTTS are replaced by *Extended Web Service Timed Transition Systems* (EWSTTS). In addition, in EWSTTS we cater for data capabilities used in our previous work [13], that are not considered in WSTTS. For the data management, we use information systems which are relational structures. In the following, we introduce the information systems and EWSTTS.

5.1 Information system

An information system is used to manage data. It is characterized by a set of objects defined by a set of attributes that can change their values by performing three atomic operations: *add*, *update*, and *delete*. Information can also be *read*. These atomic operations are effects of actions performed by services. To enforce constraints over counters, we consider a set of data that represent the counters. The value of counters can be *incremented*, *decremented* or *reset*.

5.2 Extended Web Service Timed Transition Systems (EWSTTS)

EWSTTS introduces several aspects such that: (1) data flow capabilities, (2) constraints over data, (3) a global clock and (4) constraints over counters. Moreover, in WSTTS there are no final states, whereas in the Extended WSTTS we propose (EWSTTS), we consider final states (component F) as usual in timed automata. An EWSTTS is a finite-state machine in which a transition performs an action having an effect on the information system, send or receive a message. Services can get data from the information systems. By sending and receiving messages, services can also exchange data. An EWSTTS is equipped with a set of clocks and data variables, and transitions are guarded by constraints over *clock* and *data* variables. We consider two kinds of data: (i) parameters of messages/operations, and (ii) counters that allow us to express constraints. Transitions are labeled by *constraints over data*, *timed constraints* and resets of *local clocks*.

Definition 5.1 An EWSTTS is a tuple $P = (S, s_0, F, M, A, C, X, T)$ such that

- S is a set of states, s_0 is the initial state and $F \subseteq S$ is a set of final states.
- A is a set of actions.
- M is a set of messages.
- C is a set of constraints over data (including counters).
- X is the set of clocks (local clocks and a global clock).
- A set of transitions $T \subseteq S \times (M^{?!} \cup A) \times C \times \Psi(X) \times 2^{X \setminus \{g\}} \times S$, with an element of the alphabet (action or an exchanged message), a constraint over data, a guard over clocks, and the clocks, except the global clock (g), to be reset.

We define now the extended semantic of WSTTS [15]. The semantic of an EWSTTS is defined using a transition relation over configurations made of a state, a clock valuation and a data valuation. The clock valuation is a mapping $u : X \to \mathbb{T}$ from a set of clocks to the domain of time values. The data valuation is a mapping $v : D \to \mathbb{V}$ from a set of data D to the domain of data values. The mapping u_0 denotes the (initial) clock valuation, such that $\forall x \in X \setminus \{g\}, u_0(x) = 0$. The mapping v_0 denotes the (initial) data valuation.

A service remains at the same state s without triggering a transition when the time increments, if there is no transition (s, a, c, ψ_X, Y, s') such that the constraints c over data, and the temporal constraints ψ_X are both satisfied, where $\psi_X \subseteq \Psi(X)$. A service moves from state s to state s' by triggering a transition $t = (s, a, c, \psi_X, Y, s')$ if the constraints c and ψ_X are satisfied.

Definition 5.2 (Semantic of EWSTTS)
Let $P = (S, s_0, F, M, A, C, X, T)$ be a EWSTTS. The semantic is defined as a labeled transition $(\Gamma, \gamma_0, \rightarrow)$, where $\Gamma \subseteq S \times V_T \times V_C$ is the set of configurations, such that V_T is a set of temporal valuations, V_C is a set of data valuations, $\gamma_0 = (s_0, u_0, v_0)$ is the initial configuration, and $\rightarrow$ is defined as follows:

- *Elapse of time*: $(s, u, v) \overset{\text{tick}}{\rightarrow} (s, u + d, v)$
- *Location switch*: $(s, u, v) \overset{a}{\rightarrow} (s', u', v')$, if $\exists t = (s, a, c, \psi_X, Y, s')$ such that $v \wedge c$ and $u \wedge \psi_X$ are satisfiable and $\forall y \in Y, u'(y) = 0$, $\forall x \in X \backslash Y, u'(x) = u(x)$, where $Y \subseteq X \backslash \{g\}$. Informally, v' is obtained from v by updating data changed by a.

In our context, we assume that *Location switch* is applied eagerly: when both *Elapse of time* and *Location switch* can be applied, *Location switch* is chosen. Notice that a possible use of *Elapse of time* is to precede a *Location switch* in order to represent the cost in time of an operation.

6 Compatibility analysis

Two Web services are said *compatible* if they can interact without been blocked. In the literature, this may happen when a service is waiting for a message that the other service does not send. The compatibility problem was studied in several works [8,4,3,18]. In all these works, the Web services are modeled only by their messages sequences. However, the compatibility analysis should not only rely on the sequences of messages they can exchange. For example, two Web services can support the same sequences of messages but if they do not involve the same data flow, these services are incompatible.

To provide a compatibility checking framework that considers multiple aspects, we model Web services using their sequences of messages, the parameters of the messages, the constraints over their parameters, the actions, and the aforementioned temporal constraints. By considering parameters of messages and constraints over these parameters, two Web services (a provider and a client) can be not compatible for many reasons: (1) the client (resp. the provider) waits for a message that the provider (resp. the client) does not send, (2) the data flow of the input messages differs from the output message's data flow, (3) the constraints over data or (4) the temporal constraints corresponding to the output and input messages are inconsistent i.e., they have disjoint sets of solutions.

Regarding the constraints over data, two services are compatible if their respective transitions that allow to send and receive the message are consistent. Thus, we need to check if the transitions are consistent. We call this process *the local consistency checking* of transitions. However, according to the temporal constraints, performing a local consistency checking is not always adequate, since the constraints of some transitions can have an impact on the triggering of other transitions. Thus, we propose to infer external constraints that are implicit according to the internal constraints. In the following, we explain the *local consistency checking* of transitions. Then we show how and why the internal constraints are not sufficient for the compatibility analysis of services.

6.1 Local consistency of transitions

Two Web services Q_1 and Q_2 are said compatible if each message sent by the service Q_1 (resp. Q_2) is received by the service Q_2 (resp. Q_1). So the compatibility checking relies on the consistency of the transitions that correspond to the pairs (*input message, output message*). By considering constraints over data and temporal constraints, we distinguish three classes of consistency of two transitions: (1) *absolute consistency*, (2) *likely consistency* and (3) *absolute inconsistency*.

An output transition t_1 of a service Q_1 is said *absolutely consistent* with an input transition t_2 of a service Q_2, if the solutions of constraints (temporal and over data) of t_1 are solutions of constraints of t_2.

For instance, let us suppose a service Q_1 that can send the message m within the interval $[20, 30s]$, and a service Q_2 that can receive the same message within the interval $[10, 60s]$. Since $[20, 30] \subseteq [10, 60]$, once the message is sent, it will be received. Now, consider that m can be sent in $[10, 40s]$, but received only within $[10, 20s]$. If m is sent at $30s$, it cannot be received, and so for this value, transitions are clearly *inconsistent*. On the other hand, if the message is sent at $10s$, it will be received. Hence, for some solution values the transitions are consistent and for some others, they are not, i.e., they are *likely consistent*.

Definition 6.1 An output transition $t_1 = (s_1, !m, c_1, \psi_{X_1}, s'_1)$ is *(locally) absolutely consistent* with an input transition $t_2 = (s_2, ?m, c_2, \psi_{X_2}, s'_2)$, denoted by $t_1 \subseteq t_2$ if $Sol(c_1) \subseteq Sol(c_2)$ and $Sol(\psi_{X_1}) \subseteq Sol(\psi_{X_2})$, where $Sol(c_i)$ denotes the set of solutions of the constraint c_i related to the data and $Sol(\psi_{X_i})$ denotes the set of solutions of the temporal constraint ψ_{X_i}.

t_1 is *likely consistent* with t_2, denoted $t_1 \subsetneq t_2$ if $t_1 \nsubseteq t_2$ and $Sol(c_1) \cap Sol(c_2) \neq \emptyset$ and $Sol(\psi_{X_1}) \cap Sol(\psi_{X_2}) \neq \emptyset$.

t_1 is *absolutely inconsistent* with t_2 if $Sol(c_1) \cap Sol(c_2) = \emptyset$ or $Sol(\psi_{X_1}) \cap Sol(\psi_{X_2}) = \emptyset$

6.2 External constraints inference

To analyze the compatibility of two Web services, checking the local consistency of internal constraints related to pairs (*input message, output message*) of services is not always sufficient to detect incompatibilities. In fact, the temporal constraints of some transitions of a service can have an impact on other transitions of the same service. To handle this problem, we suggest to verify if the internal constraints of the service are mutually dependent. To do this, we propose to infer from the internal constraints all the potential constraints, called *external constraints*, since they will be exposed to check if they satisfy the exposed requirements of the client service.

To illustrate such a situation, let us use the fragments of two services Q_1 and Q_2 depicted in Fig. 3. According to Definition 6.1, the two transitions $(s_1, ?m_1, x < 10, s_2)$ and $(q_3, !m_1, q_4)$ are *likely consistent*, since solutions of $x < 10$ are non-disjoint with the message sending ones. However, these transitions are problematic since Q_1 must receive the message m_1 before $10s$, whilst Q_2 cannot reach the state q_3 before $20s$, i.e., it cannot send m_1 before $20s$. As a consequence, the service Q_1

cannot receive it, i.e., Q_1 and Q_2 are *incompatible*.

So, using constraints inference, we can deduce the two external constraints:

(i) service Q_1: from the receipt of the message m_0 until receiving m_1, there is at most $10s$,

(ii) service Q_2: between the sending of m_0 and the sending m_1, there is at least $20s$.

Formally, the two external constraints are:
($?m_0$, $?m_1$, $[0, 10]$)
($!m_0$, $!m_1$,$[20, 20 + t)$) where t is the time related to the run-time process.

Once we get the two external constraints, we check if their periods are consistent. In the example $[20, 20 + t') \cap [0, 10] = \emptyset$, the two transitions are *absolutely inconsistent*, hence the two services are *incompatible*.

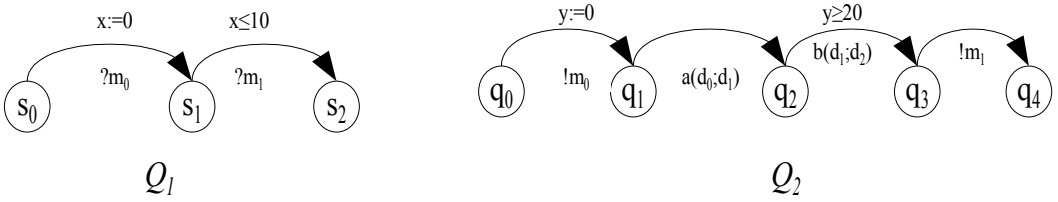


Fig. 3. Absolutely incompatible services.

Two inferred constraints (external constraints) are absolutely consistent if the period to send the message is included in the period to receive it. However, when the sending period is not included in the receiving period but the two periods have some common values, we say that constraints are *likely consistent*. When the two periods are disjoint, we say that the two constraints are *absolutely inconsistent*.

Definition 6.2 Let p_1, p_2 two pre-conditions such that ($p_1 = p_2 = true$), ($p_1 = ?m', p_2 = !m'$) or ($p_1 = !m', p_2 = ?m'$). An output external constraint $e_1 = (p_1, !m, d_1)$ is *absolutely consistent* with an input external constraint $e_2 = (p_2, ?m, d_2)$, denoted $e_1 \subseteq e_2$ if $d_1 \subseteq d_2$. e_1 is *likely consistent* with e_2, denoted $e_1 \subsetneq e_2$ if $e_1 \nsubseteq e_2$ and $d_1 \cap d_2 \neq \emptyset$. e_1 is *absolutely inconsistent* with e_2 if $d_1 \cap d_2 = \emptyset$

Inference of external constraints can be done via: (i) synchronization over messages or (ii) reference to a common clock. In the following, we explain each of them.

6.2.1 Synchronization over messages

The inference based on the synchronization over a message can be explained using the example of the two Web services depicted in Figure 3. In the service Q_1, the clock x is reset when the message m_0 is received, i.e., in the transition $(s_0, ?m_0, x, s_1)$. On the other hand, the clock y of service Q_2 is reset when the message m_0 is sent, i.e., in the transition $(q_0, !m_0, y, q_1)$. Such transitions are called *rendez-vous synchronizing transitions*, in which we can say that $x \equiv y$. Having a rendez-vous synchronizing transition, we could detect that, once the message m_0 is sent, the

service Q_2 can reach the state q_3 at least $20s$, i.e., it can send the message m_1 after $20s$. Since the service Q_1 can receive the message m_1 at most in $10s$, thus the two services are *not compatible* because their conversation will fail.

6.2.2 Global duration

When there is no rendez-vous synchronizing transition, let us assume that both services are started simultaneously. In this particular case, the idea is to infer the required period to send and receive a message. As illustrated in Fig. 4, once the service Q_1 receives the message m_0, it must receive the message m_1 within $20s$. The service Q_2 has no constraint on the transition $(q_2, !m_1, q_3)$ that enables to send the message m_1. Once the service Q_2 performs the operation $a(d_0; d_1)$, it must perform the operation $c(d_2; d_3)$ within the next $10s$. The message m_1 is sent before performing the operation $c(d_1; d_3)$, hence we can infer that the message is sent before $10s$. Then, the external constraints we can infer respectively for the service Q_1 and the service Q_2 are $(true, ?m_1, [0, 20s])$ and $(true, !m_1, [0, 10s])$

As it can be seen, $[0, 10s] \subseteq [0, 20s]$, hence we can deduce that the two services are *compatible*.

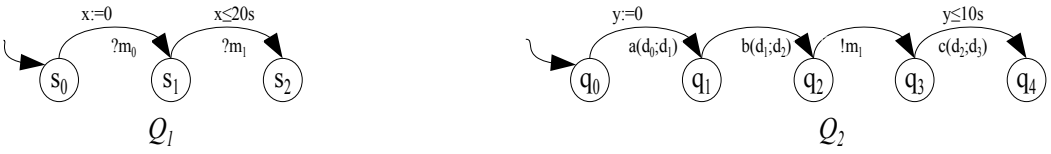


Fig. 4. An absolute compatibility.

6.3 Web services compatibility classes

As seen in Fig. 5, the compatibility of services can be checked using three steps. The local consistency allows to detect incompatibilities of Web services regarding their constraints over data (step 1). Since the local consistency checking is insufficient to detect temporal incompatibilities, the external constraints must be inferred from internal constraints (step 2). By considering all the inferred constraints, we check if services are compatible regarding their temporal constraints (step 3).

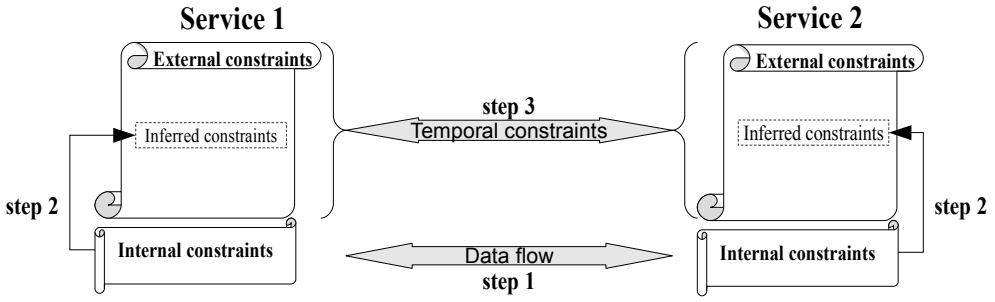


Fig. 5. Compatibility checking process.

According to our notions of consistency for external constraints and transitions, we consider three classes of compatibility for Web services: (i) *absolute compatibility*, (ii) *likely compatibility*, and (iii) *absolute incompatibility*.

Definition 6.3 Let Q_1 and Q_2 be two web services having T_1 and T_2 as respective sets of input/output transitions, and let E_1 and E_2 be their respective sets of external constraints.

Q_1 and Q_2 are *absolutely compatible* if the following holds (for $i, j \in \{1, 2\}, i \neq j$):

- $\forall t_i \in T_i$, $\exists t_j \in T_j$ such that $t_i \subseteq t_j$,
- $\forall e_i \in E_i, \exists e_j \in E_j$ such that $e_i \subseteq e_j$,

Q_1 and Q_2 are *likely compatible* if Q_1 and Q_2 are not absolutely compatible and the following holds (for $i, j \in \{1, 2\}, i \neq j$):

- $\forall t_i \in T_i, \exists t_j \in T_j$ such that $t_i \subseteq t_j$ or $t_i \subsetneq t_j$,
- $\forall e_i \in E_i, \exists e_j \in E_j$ such that $e_i \subseteq e_j$ or $e_i \subsetneq e_j$,

Q_1 and Q_2 are *absolutely incompatible* if one of the following holds (for $i, j \in \{1, 2\}, i \neq j$):

- $\exists t_i \in T_i$, $\nexists t_j \in T_j$ such that $t_i \subseteq t_j$ or $t_i \subsetneq t_j$,
- $\exists e_i \in E_i$, $\nexists e_j \in E_j$ such that $e_i \subseteq e_j$ or $e_i \subsetneq e_j$,

One can notice that the different classes of compatibility are disjoint and cover all the possible cases.

Our approach consists in analyzing the internal behavior of web services to infer external constraints used for checking the compatibility of services. We are studying methods for the inference of external constraints, and this will allow us to implement the compatibility checking presented in this section.

In the future, we want to apply the compatibility analysis framework for the composition problem. A compatibility checking algorithm will allow us to synthesize a composition by verifying the compatibility of services according to their constraints.

7 Related work

The research field on the compatibility analysis for interoperability applications is very active. A lot of works have been published on automatic service mechanisms, using automata as a formal presentation [8,4,3,18,15,16]. The fundamental issue addressed by all these works is the same: given two services, are they able to interact?

In [8], the authors consider the sequence of messages that can be exchanged between two Web services. According to our approach, we consider also the data flow and constraints over these data. Furthermore, we cater for temporal constraints.

Similarly to [8], the approach presented in [4] also considers only sequences of messages. The work presented in [4] has been extended with temporal constraints [3,18].

In [3], the temporal constraints enable to trigger transitions after an amount of time. The lack of the model presented in [3] is that the activation constraints can only be expressed over one transition. To cater for such requirements, in this paper we proposed to consider the standard clocks used in timed automata [2], to express activation constraints that cannot be expressed in [3], such that having an activation

constraint over a sequence of transitions. Contrary to our work, there is no data flow considerations, external constraints and constraints over counters. In the same area in [18] several important aspects such that data flow and temporal requirements as the external constraints, constraints over the global clock and counters are not considered.

In [12], the authors deal with analyzing and verifying properties of composite Web services specified as multiple BPEL processes. The properties are expressed via the temporal logic LTL [17]. The services are specified using an automaton-based formalism, where services are specified by the messages they can exchange asynchronously and the data flow. So, [12] investigates an approach that analyzes some given properties in a given composition, whilst we are interested in the compatibility analysis needed to build a composition.

Another work presented in [15] deals with modeling temporal requirements in a given composition. This work allows to express the same kind of constraints as in [18]. However, this model does not consider data flow, external constraints and constraints over counters and more generally constraints over data. Moreover, this work does not consider the compatibility analysis.

In the same vein, the authors of [11] propose to use timed automata to check the timed properties of a given composition. Thus they translate the descriptions written in WSCI-WSCDL [14] into timed automata. In this paper, we are interested in checking the compatibility of services, which is a key feature to synthesize a composition, whilst [11] deals with the problem of verifying a given composition.

8 Conclusion and Perspectives

In this paper, we have presented an approach to deal with the automatic compatibility checking of Web services by considering their operations, messages, data associated to messages, together with conditions on these data and temporal constraints. We have defined two forms of constraints: (i) *internal constraints* used to model the service and (ii) *external constraints* inferred from internal ones. The inference of *external constraints* allows to detect some implicit constraints that can be used to show the *incompatibility of services*. The internal constraints can be local or global. Our notion of local clock is identical to the one used in timed automata [2]. We use a global clock to specify constraints that relies on absolute dates. As the global clock relies on the absolute time, thus it is never reset. The internal constraints can express *activation* and *dependency* conditions. Moreover, we cater for data capabilities that allow us to specify guards. Then, in order to analyze the compatibility of services by considering these properties, we have proposed to extend the notion of WSTTS [15].

Our future work consists in studying how to infer automatically external constraints, and then to integrate compatibility checking mechanisms into a framework for the composition of Web services modeled as conversational automata [13].

References

[1] G. Alonso and F. Casati. Web services and service-oriented architectures. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, 2005.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] B. Benatallah, F. Casati, J. Ponge, and F. Toumani. On temporal abstractions of web service protocols. In *The 17th Conference on Advanced Information Systems Engineering (CAiSE '05). Short Paper Proceedings*, 2005.

[4] B. Benatallah, F. Casati, and F. Toumani. Analysis and management of web service protocols. *23rd International Conference on Conceptual Modeling*, November 2004.

[5] D. Berardi. *Automatic Service Composition. Models, techniques and tools.* PhD thesis, La Sapienza University, Roma, 2005.

[6] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 613–624. ACM, 2005.

[7] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2003.

[8] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella. When are two web services compatible? In *Technologies for E-Services, 5th International Workshop (TES)*, pages 15–28, 2004.

[9] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *Proceedings of the international conference on World Wide Web, WWW 2003*, pages 403–410, 2003.

[10] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. *http://www.w3.org/TR/2001/NOTE-wsdl-20010315*, 2001.

[11] G. Diaz, J.-J. Pardo, M.-E. Cambronero, V. Valero, and F. Cuartero. Verification of web services with timed automata. In *Proceedings of the International Workshop on Automated Specification and Verification of Web Sites (WWV 2005)*, volume 157 of *ENTCS*, pages 19–34, 2005.

[12] X. Fu, T. Bultan, and J. Su. Analysis of interacting bpel web services. In *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 621–630, 2004.

[13] N. Guermouche, O. Perrin, and C. Ringeissen. A mediator based approach for services composition. *INRIA-LORIA Research Report*, 2007.

[14] N. Kavanzas and al. Web service choreography description language (wscdl) 1.0. In *http://www.w3.org/TR/ws-cdl-10/*.

[15] R. Kazhamiakin, P. K. Pandya, and M. Pistore. Timed modelling and analysis in web service compositions. In *Proceedings of the The First International Conference on Availability, Reliability and Security, ARES*, pages 840–846. IEEE Computer Society, 2006.

[16] A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, volume 4423 of *LNCS*, pages 274–287, 2007.

[17] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 1981.

[18] J. Ponge. A new model for web services timed business protocols. In *Atelier (Conception des systèmes d'information et services Web) SIWS-Inforsid)*, 2006.

Chapitre 11

Dynamic Web Services Provisioning with Constraints

Auteurs : Eric Monfroy, Olivier Perrin, Christophe Ringeissen.

Référence : Proceedings of CoopIS 2008.

Abstract In this paper we consider the provisioning problem of Web services. Our framework is based on the existence of an abstract composition, i.e., the way some services of different types can be combined together in order to achieve a given task. Our approach consists in instantiating this abstract representation of a composite Web service by selecting the most appropriate concrete Web services. This instantiation is based on constraint programming techniques which allows us to match the Web services according to a given request. Our proposal performs this instantiation in a distributed manner, i.e., the solvers for each service type are solving some constraints at one level, and they are forwarding the rest of the request (modified by the local solution) to the next services. When a service cannot provision part of the composition, a distributed backtrack mechanism enables to change previous solutions (i.e., provisions). A major interest of our approach is to preserve privacy : solutions are not sent to the whole composition, services know only the services to which they are connected, and parts of the request that are already solved are removed from the next requests..

Dynamic Web Services Provisioning with Constraints*

Eric Monfroy[1,2], Olivier Perrin[3], and Christophe Ringeissen[3]

[1] Universidad Técnica Federico Santa María, Valparaíso, Chile
[2] LINA, Université de Nantes, France
[3] LORIA, INRIA Nancy Grand Est, France
Eric.Monfroy@inf.utfsm.cl,{operrin,ringeiss}@loria.fr

Abstract. In this paper we consider the provisioning problem of Web services. Our framework is based on the existence of an abstract composition, i.e., the way some services of different types can be combined together in order to achieve a given task. Our approach consists in instantiating this abstract representation of a composite Web service by selecting the most appropriate concrete Web services. This instantiation is based on constraint programming techniques which allows us to match the Web services according to a given request. Our proposal performs this instantiation in a distributed manner, i.e., the solvers for each service type are solving some constraints at one level, and they are forwarding the rest of the request (modified by the local solution) to the next services. When a service cannot provision part of the composition, a distributed backtrack mechanism enables to change previous solutions (i.e., provisions). A major interest of our approach is to preserve privacy: solutions are not sent to the whole composition, services know only the services to which they are connected, and parts of the request that are already solved are removed from the next requests.

Keywords: web service, provisioning, privacy, constraint reasoning.

1 Introduction

Composition of Web services has been recently investigated from various points of view by developing different approaches based for instance on planning techniques, logical systems and appropriate transition systems. The control of a composition can be complex. One reason is the non-deterministic behavior of services [9]. Another reason is the possible failure of services involved in the composition. Therefore, exchanged messages are difficult to manage, since they include complex data related to different aspects: temporal, security, reliability, or presentation [10,16]. The combination of all these aspects can generate a very complex design, and the resulting code can be difficult to write, to maintain,

* This work is partly funded by the INRIA-CONICYT project "CoreWeb" and the INRIA associate team "VanaWeb".

R. Meersman and Z. Tari (Eds.): OTM 2008, Part I, LNCS 5331, pp. 26–43, 2008.

and to adapt. Implementing a composition requires to take into account different aspects such as control flow, data flow, security and reliability. Languages like WS-BPEL [1] could allow us to implement a composition covering at the same time all those aspects. However, these aspects make the task very time consuming and error prone.

Our approach is based on constraint reasoning (see e.g., [4,14] for constraint programming and constraint reasoning references). Constraints are used to model in a declarative way properties of services related to various aspects: temporal requirements, reliability, security, ... The basic idea is to associate a constraint solver to each service. This solver is in charge of finding the right concrete services that are able to fulfill a given goal with respect to various requirements expressed as constraints. At that point, a natural problem arises: how the different solvers can be combined together in order to find solutions of a given client request? We address this problem in the paper by considering a simple form of composition, where (a pattern of) the control flow is already defined and the selection of services is independent from the interaction means (messages exchange, message structure...). The contribution of this paper is to present a constraint-based distributed framework for the provisioning of services. We develop the main algorithms to construct a composition thanks to solvers and backtrack mechanisms used in a distributed way. In this distributed framework, each service is building a part of the full composition.

Our approach deals with an abstract representation of a composition. Given this abstract composition, we introduce constraints defining how a type of service can interact with some other types of services. A coordination language (inspired from OWL-S [21] or the Reo language [20]) allows us to specify the orchestration of the services involved in the composition. Then, we consider the instances of services, and their own properties (used as local constraints). Given a request, the solver associated to a service enables us to build incrementally the composition. The result is a distributed constraint solving as once the solver has solved the local constraints, it forwards the request to others services and solvers. When a service s' cannot provision a part of the composition according to the solution of a service s, we have a backtrack mechanism which allows s' to ask a new solution to s. In that case, s will thus try to compute a new solution, and thus change the instantiation it has built before. Thus s' (if still part of the solution) will have a new opportunity to build the rest of the instantiation. Our approach is both dynamic and scalable, as each service is incrementally building a part of the full instantiation. Hence, this allows us to take into account the privacy of each service belonging to the composition.

The rest of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3, we detail a possible scenario. Section 4 presents the ingredients of our constraint-based framework. Section 5 shows the constraint modeling of our scenario. In Section 6, we present the mechanisms used to build a composition. The related algorithms are illustrated on the scenario in Section 7. Implementation issues are discussed in Section 8. Section 9 concludes by discussing some future works.

2 Related Work

Web service composition is nowadays a very active research direction. Many approaches have been investigated including techniques based on planning in AI [27,30], situation calculus [11,22,25], conversational transition systems [12], or symbolic model-checking applied to planning [28,29]. Our model relies on the use of constraint (logic) programming. Applying extensions of logic programming to Web service composition has been already investigated. The seminal approach presented in [22] shows how an extension of Golog (implementing a situation calculus) provides a well-suited formalism for the composition problem.

The fundamental issue addressed by all these approaches consists in automatically building a composition schema that fulfills a client query via a combination of existing services. The approach based on automata is currently very popular. In [12], Web services exchange asynchronous messages and they are modeled as Mealy machines, but there is no way to handle data, for instance as parameters of messages. In [6,8], an approach of automatic composition is presented where each service is represented as a deterministic transition system involving operations and exchanges of messages. In [7], the authors present the Colombo framework in which automata incorporate conversational aspects, parameterized operations and updates in a common database. The composition problem in this framework relies on building a mediator to perform the interaction between services needed to achieve a given goal, expressed as a goal service. In our approach, the problem appears to be less complex, since the composition of Web services is given in an abstract way. But this abstract composition still has to be instantiated, to obtain an executable composition involving concrete services.

There are few papers reporting experiments on the use of constraint reasoning for the composition problem [2,3,13,17,18,19]. We have already discussed some preliminary ideas in [24]. As in [17], we do not consider all the dimensions of the problem, since we assume that a pattern composition is already known, and we restrict us to the problem of instantiating the variables of the pattern, i.e. the different kinds of Web services. Contrary to [17], we do not consider constraints globally, but we handle constraints locally in order to build a solution gradually using a top-down mechanism. Our framework has the ability to tackle privacy requirements (as opposed to [17]) by considering only public properties of possible sub-services, by limiting exchange of data, and by keeping locally most of the knowledge. In [2], the authors use an integer programming solver by assuming that constraints and objective functions are linear. In [3], the idea is to consider the composition problem as particular constraint-based configuration problem. In that paper, a goal is expressed by a set of output messages expected by the composition. The configurator used in this context is goal-oriented and apparently proceeds by applying backward chaining techniques.

Due to similarities between constraint reasoning and AI planning [26], our approach could be compared to the use of planning systems for building a solution (a plan) to a composition problem. The Hierarchical Task Network (HTN) planning seems particularly well-suited to handle the (Hierarchy of) services involved in a pattern to be instantiated [30]. As shown in [30], the HTN planning

system SHOP2 can be applied to the composition problem. However, the planning process presented in that paper is restricted by the capabilities of the planning system, which cannot handle the concurrency. Our aim is to present a general methodology which is independent from a given solver.

3 Web Service Composition: A Portal for Printing Pictures

Let us consider a Web portal for printing numerical pictures. The components of the scenario, illustrated in Figure 1 are the client, the portal, the labs in charge of printing the pictures, and the delivery services. A printed picture has several properties, but in our example, we only retain the following properties: its size, its quality, a price.

We suppose given the abstract composition that allows a client to request a portal for printing and receiving pictures. This means that the workflow is already known, and that the process specifying the orchestration of the different activities exists (static information). However, what is unknown when the client sends the request is the concrete services instances that will fulfill the execution of the process (dynamic information). In order to provision the orchestration and to select these instances, we use a set of distributed constraint solvers that will try to find a possible assignment. This means that we do not use a unique centralized solver to instantiate the composition, but that each activity of the process is in charge of selecting the most appropriate service with respect to properties and the initial request of the client.

The starting point is the request of the client: the client is able to express some requirements to be satisfied by the composition. For instance, a client may want to get a set of printed pictures (delivery included) in a given format, a given quality, in a given deadline, e.g.:

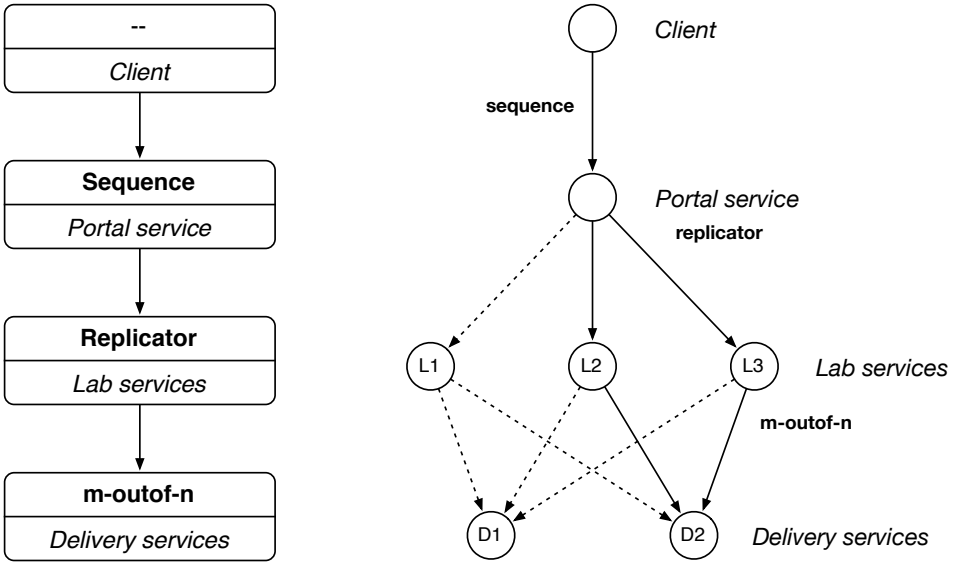


Fig. 1. The process, and one possible provision

- print and send me these 100 pictures (no temporal or quality constraints),
- print and send me these 100 pictures at the average quality and minimum cost in the 24 hours (temporal constraint, picture quality constraints, and cost constraint),
- print and send me these 100 pictures at the 6×7 format and the $8,5 \times 11$ format as soon as possible (temporal constraint, picture quality constraints).

The abstract composition (i.e. non instantiated) is described on Figure 1 (left). Using the *sequence* connector, we specify that the portal will be executed once the request of the client is sent. The portal service will replicate the request to a set of lab services (*replicator* connector), each lab having its own properties. At this level, each solver will be activated in order to select the right lab instances according to the client request. Then, we use a *m-outof-n* connector to specify that the delivery will be done by only a subset of the delivery services.

On the Figure 1 (right), we can see a possible provision of this composition. The solution returned by the solvers associated with the services instances is made of services L2 and L3 for lab services, and of service D2 for delivery.

The provisioning problem we are interested in involves multiple aspects. First, one has to specify how different types of services can be composed, i.e., what are the building blocks that are needed to orchestrate these service types in order to fulfill the user needs. This means that the ordering as well as the service types (portal, photo lab, delivery) are known. We call these building blocks the connectors, and the result is called an abstract composition.

The second aspect is to dynamically instantiate the previous abstract composition with concrete instances of services. The choice will depend on the requirements defined in the user request, but also on the possible links existing between concrete instances of services. This means, given all the constraints of the instances of services and the client request, our objective is to find one (or more) topology that satisfies the request and the constraints specified in the chosen set of services. Our proposal will achieve this objective in a distributed manner, i.e., the solvers (that equip each service) solve some constraints at one level, and they forward the rest of the request (simplified by the local solution) to the next services. This approach preserves privacy: solutions are not sent to the whole composition, services know only the services to which they are directly connected, and parts of the request that are already solved are removed from the next requests.

Our approach is both dynamic and scalable in the sense that there is no unique and centralized point of control, but rather a set of small solvers that are in charge of instantiating the abstract composition. If one solver does not find any solution, the backtrack mechanism allows us to try another solution. Then, it also scales up since it is easy to dynamically modify, add, or remove instances of services to accommodate a high number of requests, using the algorithm we provide in Section 5.

4 A Constraint-Based Approach

In this section, we present the various ingredients and concepts of our approach: abstract composition, connectors, services and their properties, requests, and exchange of constraints.

4.1 Abstract Composition

The abstract composition will guide a service for building and instantiating a part of the composition and for requesting other services. For each type of services, there is an abstract composition (that a service of another type cannot access for privacy reasons) organized w.r.t. requests: depending on a request, a service instantiates a different part of a composition. Thus, for each type S of services, there is an abstract composition that contains:

- some **choreography constraints** Cc. Given a request q, these constraints define how a service (an instance of type S) must connect and dynamically build the rest of the composition (i.e. launch the other services that are needed to execute the composite service). These constraints are based on
  - **connectors** allowed by the coordination language and
  - local constraints about **instances properties** (e.g., the use of S is "free")
- some **induced requests** corresponding to the requests the service will have to send to services it will connect to, depending on the request q it received.
- all the requests a service of a given type can perform.
- possibly, some constraints about services properties, imposing some relationships between the services appearing in the Cc constraints.

Consider a service s of type S that receives a request q. Then s can access the information of the abstract composition for type S, denoted AC_S, as follows:

- connection constraints: $Cc \leftarrow extractCc(AC_S, q)$
- requests: $extractRq(AC_S, q, s_i)$ denotes the induced request related to service s_i of type S_i depending on the abstract composition AC_S and the request q.

4.2 Connectors

To illustrate our approach, we use very few elementary connectors, namely *sequence*, *replicator* and *m-outof-n* connectors. The *sequence* connector allows two services to be executed in sequence. The *replicator* allows to duplicate the messages issued from one service to several services. The induced request w.r.t. q is the same for all the target services called by the source service. The response awaited is the first response coming from one of the target services. Then, the last connector is the *m-outof-n* connector, which allows for duplicating the messages, and for waiting the response from at least m services. The semantics of how the responses are combined is given in Section 5.1.

Our connectors can be compared to the ones introduced in OWL-S [21]. Our *sequence* connector is equivalent to the OWL-S one, since it defines a sequence control construct. Our *replicator* and *m-outof-n* connectors manage concurrency. These connectors are semantically equivalent to the $split + join$ construct of OWL-S. The difference between the two connectors deals with how results are collected at the end of the execution (results are collected from one service in the *replicator* case, while there are collected from m services in the *m-outof-n* case). The "if-then-else" and "choice" constructs of OWL-S are not directly mapped to a dedicated connector in our framework, but they could be mimic using a specific selection of solutions in the algorithm in charge of building the concrete composition (this algorithm is given in Section 6.2). Moreover, the "loop" construct could be mimic as a chain of sequences of different instances of the same service, where a counter is used to distinguish the different instances successively called at each iteration of the loop.

4.3 Services and Property Constraints

We consider a catalog of services that contains the service instances and their respective types. This catalog is accessible to each service, but for privacy reasons, we could consider to split it into sub-catalogs with restricted access.

A service instance s has its own properties or requirements that are given as some possibly fully instantiated constraints that we call the **property constraints** of s. We denote them $Cs(s)$ or Cs when s is clear from the context. Some of these properties are *public* (they can be forwarded to other services) whereas others are *private* (they are used only in the service itself). These properties are related to the requests the service s can achieve. For example, one can specify that a service s accepts to work with a service s' for a given work, but will not accept for another work ($link(s, s', q)$ for request q while $non-link(s, s', q')$ for request q'). If a service s accepts (resp. does not accept) to work with s' for any kind of request, we simply write $link(s, s')$ (resp. $non-link(s, s')$). A service s will access its *private* (respectively *public*) properties related to the request q with $extractCsPriv(s, q)$ (respectively $extractCsPub(s, q)$).

A service s which wants to connect to a service of type S' can access the public properties of a service instance s' before "really being connected" by calling: $extractCsPub(s', q)$. This allows s to get more information/constraints before choosing or refuting a service of type S'. In practice, this is important since it allows us to exclude candidate solutions without calling the machinery to explicitly connect to these services. Let us consider s, with the following Cc constraints: $replicator(s, X_1 : S_1, X_2 : S_2) \wedge version(X_1) = version(X_2)$. s will look for the instances of type S_1 (let say s_{11} to s_{1n}), and the instances of type S_2 (s_{21} to s_{2m}). s will get the constraints/properties of each s_{ij} before computing solutions corresponding to pairs of services having the same version number. Without this feature, s would have to consider each pair (s_{1i}, s_{2j}) as a candidate solution; then it would have to connect to these services; wait for their public properties; then check these constraints with respect to the constraints of the request (of s); and then backtrack if the two services do not have the

same version. This solution would drastically slow down the building process by generating, connecting, and testing impossible provisionings.

4.4 Requests

A request is a constrained message asking a service to achieve some task satisfying the constraints. The type of requests a service of type S can ask for is fixed. The following are some of the requests a user can send to a photo portal:

- $estimateRq1(100, 4 \times 6)$: the user wants 100 photos in format 4×6;
- $estimateRq2(100, 4 \times 6, 80)$: the user wants 100 photos in format 4×6 for less than 80 euros;
- $estimateRq3(100, 4 \times 6, 80, 48, lab \neq cheapfot)$: the user wants 100 photos in format 4×6 for less than 80 euros, wants to receive the pictures in less than 48 hours, and does not want the pictures to be printed by the lab "cheapfot".
- $estimateRq4(100, 10 \times 15, lab \neq cheapfot)$: the user wants 100 photos in format 10×15, and does not want the picture to be printed by the lab "cheapfot".

We use auxiliary functions to transform requests into constraints (e.g. $constraints\_of$) and to forward requests according to the abstract composition and the current local solution of constraints (e.g. $extractRq$). Hence, the function $constraints\_of(q)$ extracts the constraints contained in the request q in a form suitable to the solver of the service. For the last example of request, this could be: $Number\_pics = 100 \wedge format = \text{“}4 \times 6\text{”} \wedge Price < 80 \wedge Time < 48 \wedge lab(S) \wedge S \neq cheapfot$. To illustrate $extractRq$, consider a laboratory service: if it receives a request involving the constraint $Time < 48$, it forwards $Time < 48 - T$ to delivery services in order to take into account the duration T it needs to print the pictures.

Some request will require the service to build a new part of the composition (e.g., an estimate request that the portal receives) whereas others will only re-use already running services (e.g., a printing request that the portal receives).

4.5 Exchanging Constraints

To build a complete composition, the services need to exchange some (parts of) constraints or solutions (instantiated constraints). Only the required constraints are exchanged in order to keep privacy:

- constraints issued from a request and that must be communicated to the next service, are forwarded in the induced request. This is given in the abstract composition AC of the type of the service, and enables to strengthen privacy. E.g., consider a portal receives the request: $estimateRq4(100, 4 \times 6, lab \neq cheapfot)$. The last constraint of the request ($lab \neq cheapfot$) is treated by the portal itself: it will not ask an estimate to the lab "cheapfot". The induced request $estimateRq4 - 1$ it thus sends to the lab it will connect to is: $estimateRq4 - 1(100, 4 \times 6)$.

- property constraints: only the public property constraints of a service can be accessed by another service.
- Cc constraints are never exchanged between services: thus, to strengthen privacy, a service only knows about the services it connects to.

5 Constraints Modeling on the Photo Portal Problem

Considering our motivating example (Section 3) and the model we introduced, we obtain the following scenario. Consider 4 types of services: user, portal, laboratory, and delivery. Let assume we have one user service (called u), two portal services ($p1$ and $p2$), five laboratory services ($l1, l2, l3, l4, l5$), and three delivery services ($d1, d2, d3$) defined as follows in the catalog of service instances:

```
user(u).
portal(p1); portal(p2).
laboratory(l1); laboratory(l2); laboratory(l3);
laboratory(l4); laboratory(l5).
delivery(d1); delivery(d2); delivery(d3).
```

5.1 Abstract Compositions

The AC is defined for each type of service, and for privacy reasons, a service will only have access to the abstract composition related to its type (e.g., a portal will not see the constraints related to the construction of a photo lab). The Cc constraints are defined as follows by introducing the different types of services, their combinations, and possibly constraints over the services of a combination. For instance, the following Cc constraint states that a portal only wants to connect to labs that provide free estimates:

```
replicator(P,L*). portal(P). laboratory(L). estimate(L,free).
```

The first part of the above Cc constraint defines the connector between two types of service, i.e. a *replicator* connector applied onto a portal service type (denoted by `P`), and a set of laboratory service type (denoted by `L*`). The second part of the constraint gives the types of the services that appear in the connector. Here, `P` is a service of type portal, and `L` is a service of type laboratory. Then, the last part allows to associate conditions to the services. In the above example, the portal service only wants to connect to labs that provide free estimates.

5.2 Property Constraint of the Service Instances

Suppose that the portal $p1$ works with laboratories $l1, l3$, and $l5$, and the laboratory $l1$ only works with delivery services $d1$ and $d3$; $l1$ can print formats 4×6 and 8×12, its maximum printing time is 24h, and $l1$ provides free estimates. Then, the private part of the property constraint of $p1$ contains the following private constraints that will be used by $p1$ to compute the labs it will connect to:

```
link(p1,l1); link(p1,l3); link(p1,l5).
```

Similarly, $l1$ has the following private constraints:

```
link(l1,d1); link(l1,d3).
```

and the public constraints:

```
format(l1,4x6); format(l1,8x12).
estimate(l1,free). maxDuration(l1,24).
```

Note that a property could be a non instantiated constraint. So we can imagine to use more general constraints in Cs (i.e. not only ground unit clauses). For example, consider that $l2$ can connect to all deliveries. Its private Cs constraint would be: `link(l2,D). delivery(D).` where D is a (universally quantified) variable representing a service, and `delivery(D)` a constraint fixing its type.

5.3 Requests

Consider the user wants 100 photos in format 4×6, wants to receive the pictures in less than 48 hours, and does not want the pictures to be printed by the lab "cheapfot". The request q to a portal is: $estimateRq4(100, 4 \times 6, 48, lab \neq cheapfot)$. The function $constraints\_of(q)$ of the portal extracts the constraints contained in q, so q is translated in: $Number\_pics = 100 \wedge lab(L) \wedge format(L, 4 \times 6) \wedge Time < 48 \wedge L \neq cheapfot$.

Consider q was sent to the portal $p1$. $p1$ will build the corresponding part of the composition by connecting to laboratories (computing solutions is described in the section 6.2). $p1$ will connect (replicator connector) to laboratory services:

- that print '4×6'; this requirement is part of the request,
- that print in less that 48h; this requirement is part of the request,
- that are either $l1$, $l3$, or $l5$; this is required by the private property constraints of $p1$,
- that perform free estimates; also required by the private properties of $p1$,
- that are not the $cheapfot$ laboratory since this is prohibited by the constraints in the request.

The same induced requests will be sent to each laboratory that satisfies these requirements: $estimateRq4p(100, 4 \times 6, 48)$. Note that the constraint about the laboratory has now disappeared.

We skip some steps, up to the induced request a laboratory will send to a delivery satisfying its constraints: $estimateRq4l(pics, 200, 48 - T)$ where T is the time the laboratory will take to print the pictures, and 200 is the weight of the parcel to deliver that contains pictures ($pics$).

6 Constraint Solving and Instantiating Compositions

6.1 Solvers

Solvers we use in our framework aim at instantiating variables which denote services. The instantiation is well-typed, a variable of a given type is instantiated

by a value (a service) of this type. A solver deals with a constraint store involving different kinds of constraints.

- Cs constraints to specify the (public and private) properties of services.
- Cc constraints to specify how services are connected and constraints between services. These constraints include atoms of the form $p(s, s_1, \ldots, s_n)$ stating that s connects to $s_1, \ldots, s_n$ with a connector p of the coordination language.
- Cq constraints associated to a request q, possibly involving constraints over data.

A solution of the abstract composition Cc together with q is an assignment α of variables in Cc and Cq s.t. $\alpha(Cc \wedge Cq)$ is true with respect to the related Cs constraints.

In practice, we do not collect all the constraints of the whole composition before calling a unique solver. Instead, we solve constraints locally, for each service, and the solver of this service satisfies the 2 following features.

- the solver computes an instantiation of variables denoting sub-services, together with a solved form for constraints over data: this solved form is then propagated to sub-services by applying it to the induced requests.
- the solver finds a solution w.r.t. the current Cs plus the public part of Cs constraints of candidate sub-services. These Cs constraints may combine together differently depending on the connector p between services. For sake of simplicity and with respect to the connectors we consider in this paper, we consider the conjunction of Cs constraints in the algorithm given below. Indeed, if each property constraint is "tagged" with the name of the service instance (e.g., `estimate(l1,free)` is tagged with $l1$), then the conjunction of property constraints can be made safely for *replicator* and *m_outof_n* connectors. A conjunction is also needed between Cs constraints issued from several connectors.

6.2 Dynamic and Distributed Composition Algorithm

We now describe the algorithms of our framework. The *request_received* function and the *backtrack_request* function are triggered w.r.t. some events: reception of a request for the first function, and reception of a backtrack request for the second one. *compute_sol* and *build* are functions called by the 2 functions above.

Request Received. A service s receives a request q from s': s computes a set of solutions (Sol), each solution being a sequence of services instances that it must connect to. Then, s builds a part of the composition using these solutions.

Algorithm 1. *request_received*(q)

```
% computes the solution w.r.t. all constraints available to s
Sol ← compute_solutions(q)
% call to the function that builds an instantiated composition
build(Sol, q)
```

Computing Solutions. A service s computes a set of solutions Sol w.r.t. a request q, the abstract composition it tries to build, its own property constraints, and the property constraints of the candidate services it will build.

Algorithm 2. *compute_solutions*(q)

% s gets the constraints of the abstract composition related to the request
$Cc \leftarrow extractCc(AC_S, q)$
% s gets its property constraints
$CsPP \leftarrow extractCsPub(s, q) \wedge extractCsPriv(s, q)$
% each $sol \in Sol1$ verifies the constraints of the AC, of the request
% and of the properties of s
$Cq \leftarrow constraints\_of(q)$
$Sol1 \leftarrow solve(Cq \wedge Cc \wedge CsPP)$
% Solutions are filtered again w.r.t. the public constraints/properties
% of the services appearing in a candidate solution of $Sol1$
$Sol = \emptyset$
while $Sol1 \neq \emptyset$ **do**
  % a solution sol is selected from $Sol1$
  $sol \leftarrow select(Sol1)$
  $Sol1 \leftarrow Sol1 \setminus \{sol\}$
  % Assume sol associates $s_1, \ldots, s_n$ to s through connectors
  % get the property constraints of the services appearing in sol
  $CsPubS \leftarrow \bigwedge_{i=1}^{n} extractCsPub(s_i, q)$
  % sol is verified with the constraints of the public part of the property
  % constraints of the s_i. If it satisfiable, then sol is used to update Sol
  if $solve(Cq \wedge Cc \wedge CsPP \wedge CsPubS \wedge sol) \neq \emptyset$ **then**
    $Sol \leftarrow Sol \cup \{sol\}$
  end if
end while
return (Sol)

The solver associated to each service is based on search space reduction (eliminating values of the domains of the service variable that do not satisfy the constraint. E.g., consider the constraint $lab(L), free\_estimate(L)$; then all lab instances that do not give free estimates are removed from the possible instantiation for L.) and enumeration.

Composition Building. A service s builds a new part of the composition based on the solutions computed w.r.t. the request q that was addressed to s by s'. If s does not have any more solution, then s asks to the service that sent it the request to backtrack: this is inter service backtrack. Otherwise, s takes one solution sol and builds/connects to the corresponding services w.r.t. the connectors of the composition. Then, depending on the connectors (defined in Cc), and of the solutions, s sends some induced requests to the services it connected to.

Algorithm 3. $build(Sol, q)$

```
if Sol = ∅ then
  % s has no more solution: it asks s' to backtrack: inter service backtrack
  backtrack_request(s', q)
else
5:  % s selects a solution sol w.r.t. some criteria
  sol ← select(Sol)
  Sol ← Sol \ {sol}
  Cc ← extractCc(AC_S, q)
  % for each connector of the abstract composition
10: for all p(X : S, X_1 : S_1, ..., X_n : S_n) ∈ Cc do
    % services of the connector are instantiated w.r.t. the solution sol
    p(s, s_1, ..., s_n) ← p(sol(X), sol(X_1), ..., sol(X_n))
    % create (when necessary) and connect to the services
    for all i ∈ [1..n] do
15:   if not_running(s_i) then
        launch(s_i);
      end if
      connect(s, s_i);
    end for
20:   % create the connector if necessary
    if not_active(p(s, s_1, ..., s_n)) then
      create(p(s, s_1, ..., s_n))
    end if
    % sends the induced request to each service
25:   for all i ∈ [1..n] do
      q_i ← extractRq(AC_S, q, s_i)
      % sol(q_i) denotes q_i plus the solution over data constraints
      send(s_i, sol(q_i))
    end for
30: end for
end if
```

The *select* function can be used to optimize the composition: for example, when backtracking, one can imagine to select a solution which is as close as possible (e.g., minimizing the number of different services appearing in the new solution) from the previous solution in order to re-use already running services.

As explained below, it may happen that one can reuse an active connector which has been created for a previous request (see line 20 of Algorithm 3).

Backtrack Request. s receives a backtrack request from s' w.r.t. the request q': s is asked to backtrack. To this end, s needs to know which set of solutions Sol and which request q are related to s', and with which connector s' was connected to s: this is done with the *has_produced* function which is just a call to the memory of s. Depending on the connectors, we have chosen to trigger the backtrack as follows:

- for a sequence connector, the backtrack is always achieved;
- for a connector $replicator(s, s_1, \ldots, s', \ldots, s_n)$, the backtrack is not achieved if at least one of the s_i is still connected to s with this connector (if a s_i is still connected to s for another connector –e.g., s_i was connected twice to s for 2 tasks–, it does not count).
- for a connector $m\_outof\_n(s, s_1, \ldots, s', \ldots, s_n)$, the backtrack is not achieved if at least m of the s_i are still connected to s with this connector.

Algorithm 4. $backtrack\_request(s', q')$

$(q, Sol, p(s, s_1, \ldots, s', \ldots, s_n)) \leftarrow has\_produced(q')$
$disconnect(s')$
if $p = sequence \vee$
$\quad (p = replicator \wedge card(still\_connected(s_1, \ldots, s_n)) < 1) \vee$
$\quad (p = m\_outof\_n \wedge card(still\_connected(s_1, \ldots, s_n)) < m)$ **then**
$\quad build(Sol, q)$
end if

Note that s does not kill s' although s' will possibly not participate in the next solution of s. This, because it could be that s' is treating a request from a third service s'' (s' also participates in another part of the composition), or s' is waiting for another request from the same service. Indeed, we consider the following mechanism:

- using the abstract composition, a service s knows whether it will or not be re-used by a service, in a given time limit. For example, a portal is first asked by a user to provide him/her with estimates. The portal also knows that the user will ask for printing (realize an estimate) in the next 2 days (time limit).
- thus, a service which is not currently working for some requests, and not waiting for some new requests (all the time limits for requests are already passed) will vanish by itself after a given idle time t.
- the extra time t, enables s to backtrack, and thus maybe re-use s' in a new solution without having to launch it again (a connection will be enough).

7 Execution on Web Portal Photo

We now illustrate how the above algorithms work on the Web portal photo application. For lack of space, we describe only a part of the execution, showing the main features of our framework: we mainly focus on the backtracking.

Consider a photo lab l that receives the following request from the portal p: $estimateRq(100, 10 \times 15, 48)$, a request for an estimate for printing 100 pictures in the 10×15 format, in 48h (including delivery time).

l can work with p, it requires 16h for printing the pictures for 50 €. The 100 pictures weights 200g. l knows 6 delivery services: d_1 to d_6. Consulting the property constraints of the d_i's, l knows that d_6 and d_5 require 72h for delivery, d_1, d_3, and d_4, require less than 24h, and d_2 does not specify its maximum delay for delivery.

The Cc constraint of l indicates that l must connect to deliveries with a m_outof_n connector with $n = 3$ and $m = 2$. It will thus connect to 3 deliveries and wait for answers of 2 of them.

l solves all the constraints it has at hand, and finds 4 candidate solutions (d_6 and d_5 having too long delays w.r.t. the request, and the order of the services in the m_outof_n connector being irrelevant): $\{(d_1, d_2, d_3), (d_2, d_3, d_4), (d_1, d_3, d_4), (d_1, d_2, d_4)\}$. Thus, l builds a new part of the composition ($build$ function) selecting for example the first solution. It connects to d_1, d_2, and d_3, and sends them the following induced constraint: $estimateDel(200, pics, 32h)$ meaning that the estimate is for sending a parcel of pictures (pics), weighting 200g, in 32h (48h of the original request, minus the 16h l required for printing).

Although the d_i's do not have any Cc constraint (they do not need to build a new part of the composition), they however solve all the constraints they have (constraints in the request and their own property constraints) to verify that they can work with l and achieve the task.

d_1 has the property constraint $non-link(l, d_1, estimateDel)$ meaning that d_1 does not want to work with l for making delivery estimates. The set of solutions of d_1 is empty, and thus d_1 will send a backtrack request to l.

Since d_1 was in a m_outof_n connector, and that 2 delivery services (d_2 and d_3) among the 3 are still connected to l, l does not backtrack. Now, consider that d_2 requires $50h$ for delivery: d_2 will also have an empty set of solutions, and will thus ask l to backtrack. This time, l must backtrack since only one delivery remains connected. l thus call $build$ again to select another solution, say (d_2, d_3, d_4): the corresponding composition is built.

Again d_2 will make a backtrack request to l. Since d_3 and d_4 are still connected, l does not backtrack. Assume that d_3 and d_4 have some solutions: they will thus send back to l the price of an estimate, and the delay: 10 € in 24h for d_3, and 5 € in 30h for d_4. Consider l chooses the cheapest one: l will thus send this estimate to p: 100 pictures in 46h (16+30) for 55 € (50+5).

8 Reusing a Distributed Cooperative Constraint Solving Environment

The proposed framework presents many similarities with the BALI environment [23]. The main components of BALI are the solvers and the collaboration language. The solvers can be viewed as dedicated services and input constraints as solving requests. The BALI collaboration language supports the following connectors: sequence, parallel, and concurrent. In addition, control connectors such as iterator, fixpoint and conditional strengthen the expressivity of the language.

The implementation of BALI transforms an expression written in the collaboration language into a distributed system that behaves like a classical solver: inputs are constraints and outputs are solutions. The backtrack is fired when the set of constraints becomes unsatisfiable. In our framework, the backtrack arises when it is not possible to entirely build the service collaboration/choreography. Therefore, the backtrack plays an important role in the instantiation of the

composition, whilst it is only used to build solutions of constraints in a system like BALI. These two kinds of backtrack are used in quite different ways, but both are based on the same principles to take into account the distributed aspect.

The first implementation of BALI has been realized by using the ECLiPSe constraint programming language [31] augmented with the CHRs [15] for managing constraints and implementing the primitives of the collaboration. In this configuration, solvers and their collaborations are encapsulated into ECLiPSe processes to create homogeneous agents. Then, the agents execute independently in a distributed environment, where some ECLiPSe processes monitor and coordinate the whole system. This implementation turns solver/service collaboration into services to which clients/host languages can connect. More recently, BALI implementation has been revisited to take advantage of the features of a control-oriented coordination language such as MANIFOLD [5].

As numerous similarities exist between the solver-dedicated distributed platform BALI and the proposed framework, we have chosen to develop a prototype that is coded by reusing significant parts of the BALI architecture and a general-purpose coordination language.

9 Conclusion

There exist several works related to the use of constraint solving for Web services composition [2,3,13,17,18,19]. As in [17], we do not consider all the dimensions of the problem, since we assume that a pattern composition is already known, and we restrict us to the problem of instantiating the variables of the pattern, i.e., the different kinds of Web services.

In this paper, we promote the use of constraint reasoning to implement a form of "pattern" instantiation for any kind of "pattern" (possibly a cyclic graph). We have proposed a framework in which constraint solving is performed in a distributive way via sophisticated backtrack mechanisms. An interesting point of our approach is to preserve privacy (as opposed to [17]) by considering only public properties of possible sub-services, by limiting exchange of data, and by keeping locally most of the knowledge. In our framework, each solver is in charge of computing a part of the full composition, and each solver interacts with some others by forwarding an update of its request. Note that the degree of privacy depends on the locality of solvers. If a solver is associated to a block of services instead of a single one, then there is less privacy but we get more efficiency. In the future, we plan to study the relationship between locality of solvers, privacy, and efficiency. More generally, our aim is to improve the applicability of our approach. Hence, it is important to take into account the fact that properties can be dynamically modified (temporal availability or QoS information about services for instance). Moreover, it would be interesting to provide a flexible support for interfaces of services and to introduce variations in expression of properties (e.g. required or optional parameters). Another extension deals with the possibility to monitor a fully instantiated composition, so that a selection of services can be changed on-the-fly depending on their behaviors. Applying

constraints for the monitoring of a composition appears to be an interesting line of research we want to investigate as a continuation of the presented framework.

References

1. Web Services Business Process Execution Language, `http://www.oasis-open.org`
2. Aggarwal, R., Verma, K., Miller, J.A., Milnor, W.: Constraint driven web service composition in meteor-s. In: Proc. of SCC, pp. 23–30. IEEE, Los Alamitos (2004)
3. Albert, P., Henocque, L., Kleiner, M.: Configuration-Based Workflow Composition. In: 2005 IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, USA, July 11-15, 2005, pp. 285–292. IEEE Computer Society, Los Alamitos (2005)
4. Apt, K.R.: Principles of Constraint Programming. Cambridge Univ. Press, Cambridge (2003)
5. Arbab, F., Monfroy, E.: Coordination of heterogeneous distributed cooperative constraint solving. SIGAPP Appl. Comput. Rev. 6(2), 4–17 (1998)
6. Berardi, D.: Automatic Service Composition. Models, techniques and tools. PhD thesis, La Sapienza University, Roma (2005)
7. Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Mecella, M.: Automatic composition of transition-based semantic web services with messaging. In: Proc. of VLDB, pp. 613–624. ACM, New York (2005)
8. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 43–58. Springer, Heidelberg (2003)
9. Berardi, D., Calvanese, D., Giacomo, G.D., Mecella, M.: Composition of services with nondeterministic observable behavior. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 520–526. Springer, Heidelberg (2005)
10. Bhiri, S., Perrin, O., Godart, C.: Ensuring required failure atomicity of composite web services. In: Proc. of WWW, pp. 138–147 (2005)
11. Bienvenu, M., Fritz, C., McIlraith, S.: Planning with qualitative temporal preferences. In: Proc. of KR, pp. 134–144 (2006)
12. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: Proc. of WWW (2003)
13. Channa, N., Li, S., Shaikh, A.W., Fu, X.: Constraint satisfaction in dynamic web service composition. In: Proc. of DEXA, pp. 658–664. IEEE, Los Alamitos (2005)
14. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
15. Frühwirth, T.W.: Theory and practice of constraint handling rules. Journal of Logic Programming 37(1-3), 95–138 (1998)
16. Guermouche, N., Perrin, O., Ringeissen, C.: Timed specification for web services compatibility analysis. Electr. Notes Theor. Comput. Sci. 200(3), 155–170 (2008)
17. Hassine, A.B., Matsubara, S., Ishida, T.: A constraint-based approach to horizontal web service composition. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 130–143. Springer, Heidelberg (2006)
18. Kona, S., Bansal, A., Gupta, G.: Automatic Composition of Semantic Web Services. In: 2007 IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, Utah, USA, July 9-13. IEEE Computer Society, Los Alamitos (2007)

19. Lazovik, A., Aiello, M., Gennari, R.: Encoding requests to web service compositions as constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709. Springer, Heidelberg (2005)
20. Lazovik, A., Arbab, F.: Using Reo for Service Coordination. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 398–403. Springer, Heidelberg (2007)
21. Martin, D.L., Burstein, M.H., McDermott, D.V., McIlraith, S.A., Paolucci, M., Sycara, K.P., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing semantics to web services with owl-s. In: World Wide Web, pp. 243–277 (2007)
22. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: Proc. of KR, pp. 482–496. Morgan Kaufmann, San Francisco (2002)
23. Monfroy, E.: The Constraint Solver Collaboration Language of BALI. In: Proceedings of FroCoS 1998. Studies in Logic and Computation, vol. 7, pp. 211–230. Research Studies Press Ltd. (2000)
24. Monfroy, E., Perrin, O., Ringeissen, C.: Modeling Web Services Composition with Constraints. In: Selected Papers of 3CCC — Colombian Conference on Computer Science. Revista Avances en Sistemas e Informática, vol. 5 (2008)
25. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: Proc. of WWW, pp. 77–88 (2002)
26. Nareyek, A., Freuder, E.C., Fourer, R., Giunchiglia, E., Goldman, R.P., Kautz, H.A., Rintanen, J., Tate, A.: Constraints and AI Planning. IEEE Intelligent Systems 20(2), 62–72 (2005)
27. Peer, J.: Web Service Composition as AI Planning - a Survey. Technical Report Univ. of St. Gallen (2005)
28. Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., Traverso, P.: Planning and Monitoring Web Service Composition. In: Bussler, C.J., Fensel, D. (eds.) AIMSA 2004. LNCS (LNAI), vol. 3192, pp. 106–115. Springer, Heidelberg (2004)
29. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: IJCAI, pp. 1252–1259 (2005)
30. Sirin, E., Parsia, B., Wu, D., Hendler, J.A., Nau, D.S.: HTN planning for web service composition using shop2. J. Web Sem. 1(4), 377–396 (2004)
31. Wallace, M., Novello, S., Schimpf, J.: ECLiPSe: A Platform for Constraint Logic Programming. ICL Systems Journal 12(1), 159–200 (1997); Revised version of Technical Report IC-Parc, Imperial College, London (August 1997)