Processus collaboratifs ubiquitaires: architecture, fiabilite et securite

Frederic Montagut

Thèse

présentée pour obtenir le grade de docteur de

l'Ecole nationale Supérieure des Télécommunications

Spécialité : Informatique et Réseaux

Frédéric MONTAGUT

Processus Collaboratifs Ubiquitaires Architecture, Fiabilité et Sécurité

soutenue le 15 Octobre 2007 devant le jury composé de

Président	**Elie Najm**	**Professeur, Télécom Paris**
Rapporteurs	**Elisa Bertino**	**Professeur, Purdue University**
	Yves Deswarte	**Directeur de Recherche, L.A.A.S. CNRS**
Examinateurs	**Frédéric Cuppens**	**Professeur, ENST-Bretagne**
	Volkmar Lotz	**Directeur de Programme de Recherche, SAP**
Directeur de thèse	**Refik Molva**	**Professeur, Institut Eurecom**

PhD thesis

Ecole Nationale Supérieure des Télécommunications

Computer Science and Networks

Frédéric Montagut

Pervasive Workflows Architecture, Reliability and Security

Defense date: October, 15^{th} 2007

Committee in charge:

Chairman	**Elie Najm**	**Professeur, Télécom Paris**
Reporters	**Elisa Bertino**	**Professor, Purdue University**
	Yves Deswarte	**Research Director, L.A.A.S. CNRS**
Examiners	**Frédéric Cuppens**	**Professor, ENST-Bretagne**
	Volkmar Lotz	**Research Program Manager, SAP**
Advisor	**Refik Molva**	**Professor, Institut Eurecom**

. . . A mes parents.

The secret to creativity is knowing how to hide your sources.
- Albert Einstein -

Acknowledgments

My thesis work having come to an end it is time to look back over the past three years and thank the people without the support of whom conducting my PhD research would not have been possible.

I would like first to express my gratitude to my supervisor Refik Molva. I learned a lot from our discussions and without his support these past three years would not have been successful. I will always be grateful to him.

I would like then to thank the SAP Research team in France for giving me the opportunity to perform my PhD thesis in the best possible conditions. I am especially thankful to the MOSQUITO team: Konrad Wrona, Annett Laube, Jean-Christophe Pazzaglia and Laurent Gomez. Working with you was a pleasure. Collaborating with the other SAP team members was also inspiring. Special thanks to Cedric Ulmer, Cedric Hebert and Laurent Gomez for the daily jokes and the new fancy expressions I got to learn. Many thanks go also to the PhD student crew Alessandro Sorniotti, Paul El Khoury, Azzedine Benameur and Mohammad Ashiqur Rahaman for their continuous support and to Bettina, Julien and Silvan for their help on the implementation.

I would like also to thank the team members of the Eurecom CE department with whom I had the chance to collaborate, in particular Slim Trabelsi and Yves Roudier. Thank you for the valuable feedbacks.

Last but not least, I would like to thank Elie Najm, Elisa Bertino, Yves Deswarte, Frédéric Cuppens and Volkmar Lotz for accepting to be part of the thesis committee.

Funds for this work were provided by SAP Labs France and by the European Commission (FP6 MOSQUITO project).

Frédéric Montagut, October 2007

Résumé en Français

Toute avancée des connaissances génère autant d'interrogations qu'elle apporte de réponses.
- Pierre Joliot -

La notion de processus collaboratif ou *workflow* se définit par l'exécution automatisée d'un ensemble de tâches nécessaire à l'exécution d'une procédure d'entreprise ou processus métier tel que la réservation d'un billet d'avion ou la planification de congés au sein d'une entreprise. Durant l'exécution d'un processus collaboratif des documents et des données sont échangés entre les participants de cette collaboration qui peuvent être des machines ou de simples utilisateurs humains. Un système de gestion *workflow* assure le contrôle et la gestion de l'exécution d'un processus collaboratif d'après la spécification du *workflow* qui définit la séquence des opérations à exécuter pour terminer le processus métier associé au processus collaboratif. Comme toutes applications collaboratives, l'exécution de processus collaboratifs doit satisfaire des contraintes rigoureuses en termes de sécurité informatique et de consistance transactionnelle. La majorité des systèmes de gestion de *workflow* implémente une infrastructure de coordination centralisée pour répondre à ces contraintes. Les performances des applications collaboratives sont cependant limitées par le système de coordination centralisé et ceci d'autant plus que les applications les plus récentes requièrent une flexibilité importante lors de leur exécution. Dans cette thèse de doctorat, nous présentons un système de gestion de *workflow* décentralisé afin d'apporter une solution à ce problème de flexibilité.

La principale contribution de cette thèse est le design et l'implémentation d'un système de gestion de *workflow* distribué ainsi que des protocoles de coordination transactionnelle et de sécurité informatique pour répondre aux contraintes identifiées ci-dessus. L'architecture de gestion de *workflow* que nous avons développée, appelée processus collaboratif ubiquitaire ou *workflow* ubiquitaire, permet d'exécuter des processus collaboratifs de manière distribuée entre acteurs qui peuvent partager leurs ressources par le biais d'un protocole de découverte. Cette infrastructure de gestion de *workflow* permet entre autre la sélection des acteurs qui exécuteront les tâches du processus collaboratif au moment même de son exécution sans assignation préalable.

La suite de ce résumé en Français est organisé comme suit. Nous résumons dans un premier temps les contributions scientifiques de cette thèse. Nous développons ensuite les trois axes

principaux de la thèse : le design de l'architecture de *workflow* ubiquitaire, la fiabilité du *workflow* ubiquitaire et la sécurité du *workflow* ubiquitaire. Ce résumé est conclu par une analyse des perspectives pour la poursuite de ces travaux de recherche.

Introduction

Avec l'émergence de l'Internet, le commerce électronique et les applications exécutées en ligne sont devenus incontournables pour tous types d'échanges commerciaux. Ces applications "e-commerce" intègrent souvent de complexes applications telles que des outils de gestion de stock ou des plateformes de paiement sécurisées et ceci de manière transparente pour l'utilisateur. La technologie du *workflow* permet l'exécution de telles collaborations qui mettent en commun les ressources de plusieurs fournisseurs dans le but de créer des applications à forte valeur ajoutée. A ce titre le concept de processus collaboratif est devenu le standard pour assurer l'orchestration de services fournis par différentes organisations dans le cadre d'applications collaboratives.

Les principaux concepteurs de logiciels tels que Microsoft, IBM ou SAP [MBB+03, KH05, biz05] ont encouragé le développement d'applications utilisant la technologie du *workflow* au cours des dernières années. Les sites de commerce en ligne ou les systèmes de réservation de billets d'avion qui permettent à des utilisateurs de remplir un caddie virtuel et de payer des commandes en lignes sont autant d'exemples d'applications construites autour de la technologie du *workflow*. On distingue deux principaux types d'infrastructures permettant l'exécution de ces applications collaboratives:

- **Centralisées**: une infrastructure dédiée s'occupe des tâches de gestion de l'exécution d'une application collaborative telles que le routage des messages entre les participants ou le stockage des données.

- **Décentralisées**: la gestion de l'exécution d'une application collaborative est partagée entre les participants qui exécutent tour à tour une sous partie de l'application globale. C'est l'infrastructure standard pour assurer la gestion de l'exécution d'applications collaboratives impliquant des acteurs appartenant à différentes organisations.

Ces infrastructures de gestion de *workflow* manquent cependant de flexilité pour permettre l'exécution des processus collaboratifs les plus récents. Par exemple, elles sont statiques en celà qu'elles ne supportent pas l'assignation d'acteurs du *workflow* en cours d'exécution, le processus de sélection a en effet souvent lieu avant l'instanciation du *workflow*. Par conséquent, les systèmes de gestion de *workflow* actuellement disponibles ne semblent pas être adaptés à la gestion de l'exécution des processus collaboratifs les plus récents comme celui présenté en annexe D par le biais duquel un médecin, un pharmacien et un travailleur social collaborent afin de porter assistance à une personne hospitalisée à domicile et dont l'état de santé est surveillé à distance. L'exécution de telles applications collaboratives pose cependant de nouveaux défis technologiques et introduit de nouvelles contraintes en matière de sécurité informatique et de coordination transactionnelle. Ces aspects sont développés dans la suite de cette introduction.

Architecture Orientée Services

Le concept d'Architecture Orientée Services et les technologies "Web services" sur lesquelles reposent son implémentation apparaissent *de facto* comme le standard pour permettre l'exécution des applications collaboratives les plus récentes étant donné qu'elles ont été adoptées par les principaux acteurs du monde informatique comme SAP, IBM ou Microsoft [BS06, KH05, biz05]. Reposant sur un socle de technologies utilisant le langage [XML], la technologie "Web services" permet aux fournisseurs de services de publier et combiner leurs fonctionnalités à faible coût sur l'Internet. Le but de la technologie "Web services" est en effet d'offrir un environnement d'exécution pour les relations "clients - fourniseurs de services" qui soit à la fois sûr, fiable, interopérable et faiblement couplé. En particulier, le paradigme d'Architecture Orientée Services offre un ensemble de principes d'exécution qui sont en adéquation avec les prérequis fonctionnels des infrastructures de gestion de *workflow*.

- **Interactions faiblement couplées**: les échanges entre collaborateurs ne dépendent pas de l'implémentation sur laquelle les applications offertes par ces derniers reposent.
- **Composition de services**: les fonctionnalités offertes par les fournisseurs de services peuvent être combinées pour offrir des services à forte valeur ajoutée aux clients.
- **Protocole de découverte**: les fournisseurs de services peuvent publier les fonctionnalités qu'ils offrent de sorte que des clients potentiels peuvent les contacter très facilement.

Ces trois principes sont en fait les fonctionnalités requises pour supporter les prérequis fonctionnels associés au design d'une architecture de gestion de *workflow* supportant les applications collaboratives les plus récentes. Dans ce cadre, l'orientation service et l'ensemble des technologies "Web services" sur lequel elle repose apparaissent comme la solution la plus appropriée pour implémenter les résultats théoriques présentés dans cette thèse.

Prérequis de sécurité informatique et consistance transactionnelle

Le design d'une infrastructure de gestion de *workflow* ne doit pas seulement respecter des contraintes fonctionnelles mais doit aussi prendre en compte des contraintes en termes de sécurité informatique et consistance transactionnelle afin que les processus collaboratifs qu'elle supporte s'exécutent de manière sécurisée et sans inconsistance au niveau des données échangées. Les applications collaboratives les plus récentes requièrent une certaine flexibilité qui introduit de nouveaux problèmes de recherche en termes de sécurité informatique et coordination transactionnelle.

Fiabilité et consistance transactionnelle

Une des principales propriétés que doit vérifier un système de gestion de *workflow* est d'assurer que le résultat atteint par les processus collaboratifs dont il gère l'exécution produisent des résultats qui sont consistants au niveau des données. Etant donné le manque de fiabilité des systèmes distribués, assurer la consistance transactionnelle des plus récents processus collaboratifs est un problème difficile. Il y a en fait deux propriétés qui doivent être prises en compte par un protocole de coordination transactionnelle afin que celui-ci soit adapté à la coordination des applications collaboratives modernes:

- **Atomicité relachée**: certains résultats intermédiaires produits par l'exécution d'un *workflow* peuvent être conservés intacts bien que l'exécution d'autres tâches du *workflow* n'ait pas réussi.

- **Assignation dynamique des acteurs du *workflow***: les dernières applications collaboratives sont dynamiques dans le sens où ses exécutants peuvent être sélectionnés durant son exécution en fonction par exemple d'informations contextuelles.

Les protocoles de coordination transactionnels existants n'offrent pas les propriétés suffisantes pour satisfaire ces deux exigences. De nouvelles solutions doivent donc être envisagées et dans cette thèse, nous proposons le design d'un protocole de coordination transactionnelle qui satisfait les contraintes que nous avons identifiées.

Sécurité informatique

Le support d'exécution distribué nécessaire à l'exécution des applications collaboratives modernes introduit de nouvelles contraintes en termes de sécurité informatique. Les acteurs de celles-ci ne reposent pas, en effet, sur un acteur de confiance pour assurer la gestion de l'exécution du *workflow*. Par conséquent, certaines propriétés vérifiées par les applications traditionnelles telles que la conformité de l'exécution du *workflow* avec la spécification de celui-ci ne sont plus assurées. Par ailleurs, étant donné l'aspect distribué de l'exécution, l'ensemble des données sont transférées entre les participants d'un *workflow* même si ceux-ci ne doivent accéder qu'à un sous-ensemble de ces données pour exécuter les tâches auxquelles ils sont assignés. De ce fait, il devient difficile d'appliquer les politiques de sécurité définissant les droits d'accès des acteurs potentiels du *workflow* sur chaque donnée. Les protocoles de sécurité définis pour les applications collaboratives traditionnelles ne permettent malheuresement pas de résoudre les problèmes identifiés ci-dessus et dans cette thèse de doctorat, nous proposons le design et l'implémentation de protocoles de sécurité qui satisfont les prérequis de sécurité informatique que nous avons identifiés pour les applications collaboratives récentes.

Organisation et contribution de la thèse

Chaque chapitre de cette thèse introduit un composant de l'infrastructure de gestion de *workflow* que nous avons conçue pour supporter l'exécution des applications collaboratives modernes. Les trois principaux composants de l'architecture de *workflow* ubiquitaire sont représentés par la figure 1. Le composant central est en fait le système de gestion de *workflow* distribué qui doit satisfaire les contraintes fonctionnelles que nous avons identifiées pour supporter l'exécution d'applications collaboratives dans des environnements qui n'offrent pas d'infrastructure dédiée. Afin de garantir une exécution fiable et sécurisée, ce système de gestion de *workflow* est supporté par un protocole de coordination transactionnel dont le rôle est de garantir la consistante des données du *workflow* et par des mécanismes de sécurité. Le reste du rapport est organisé de la manière suivante.

Chapitre 1 Le premier chapitre introduit les concepts technologiques nécessaires à la compréhension de la thèse. La notion de *workflow* et les technologies "Web services" sont en particulier abordées.

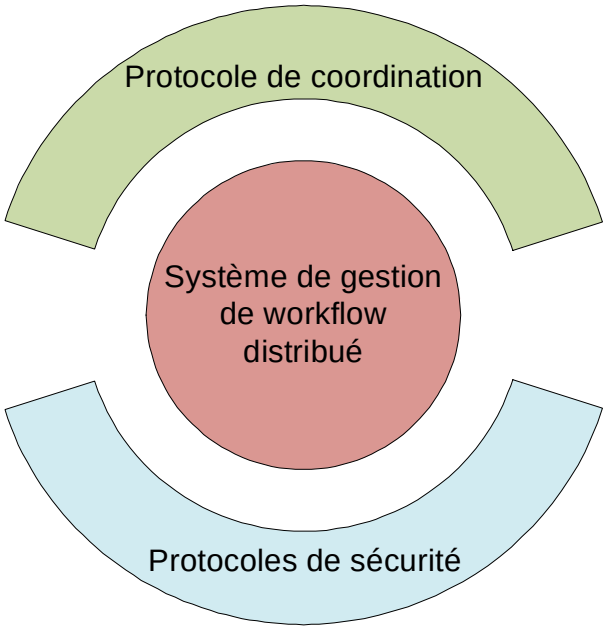

Figure 1: Organisation de la thèse en fonction des composants de l'architecture

Les chapitres 2, 3 et 4 constituent le coeur de cette thèse. Ils spécifient en effet les solutions que nous avons conçues pour remplir les objectifs fixés dans cette partie introductive. Le lecteur est invité à les lire de manière linéaire même s'ils décrivent chacun des solutions indépendantes applicables dans d'autres contextes.

Chapitre 2 A l'inverse des systèmes de gestion de *workflow* traditionnels, l'exécution des plus récentes applications collaboratives exploitant le concept de *workflow* ne reposent pas forcément sur une infrastructure dédiée. Par conséquent, de nouveaux problèmes se posent quant au design d'une infrastructure de gestion de *workflow* capable d'assurer l'exécution de celles-ci. Le chapitre 2 présente l'architecture de *workflow* ubiquitaire que nous avons conçue. Celle-ci répond aux contraintes fonctionnelles suivantes:

- **Exécution décentralisée**: la gestion de l'exécution d'une application collaborative est partagée entre les participants qui exécutent tour à tour une sous partie de l'application globale.
- **Assignation dynamique des acteurs du *workflow***: les acteurs du *workflow* peuvent être sélectionnés durant son exécution.

Par ailleurs, une implémentation de cette architecture basée sur les technologies "Web services" est aussi présentée.

Chapitre 3 Le chapitre 3 traite des problèmes de fiabilité durant l'exécution d'un processus collaboratif ubiquitaire. Nous présentons en particulier un protocole de coordination transactionnel. L'exécution de ce dernier se déroule en deux phases. Au cours de la première phase, un algorithme de sélection d'acteurs transactionnels est exécuté grâce

auquel les acteurs du *workflow* sont sélectionnés en fonction de leurs propriétés transactionnelles telles que la possibilité d'annuler les effets d'une tâche (compensation) ou la capacité de relancer une exécution après un échec (rejeu). Ainsi construite une instance de *workflow* satisfait aux contraintes transactionnelles spécifiées au moment du design du *workflow*. Le *workflow* est ensuite à proprement parlé exécuté et son exécution est supporté par un protocole de coordination transactionnel dont les règles de coordination sont dérivées de l'instance de *workflow* construite grâce à l'algorithme de sélection d'acteurs transactionnels que nous avons développé.

Une implémentation de ce protocole transactionnel est aussi présenté dans ce chapitre.

Chapitre 4 Ce chapitre présente les solutions que nous avons développées pour répondre aux contraintes de sécurité informatique que nous avons identifiées pour l'exécution de *workflow* ubiquitaire. Ces solutions exploitent les concepts d'encryption concétrique (oignons) et les modèles de politique de sécurité pour assurer entre autre la conformité de l'exécution du *workflow* distribué avec sa spécification. Le design de ces protocoles de sécurité est fortement couplé à l'exécution du *workflow* et de ce fait peut-être facilement intégré à d'autres systèmes de gestion de *workflow* distribués. Nous spécifions aussi dans ce chapitre comment les protocoles de sécurité que nous avons développés peuvent être intégrés dans l'exécution du protocole transactionnel discuté dans le chapitre 3.

Une implémentation de ce travail utilisant le concept de cryptographie à base d'identité est enfin présentée.

Appendices Les appendices présentent en détail les travaux d'implémentation réalisés au cours de cette thèse de doctorat.

Les travaux de recherche réalisés dans le cadre cette thèse ont donné lieu à différentes publications [MM05, MM06a, MM06b, MM07a, MM07b, MMG08b, MMG08a, Kha08] où sont détaillées les principales idées développées dans ce rapport.

Architecture du *workflow* ubiquitaire

Les applications collaboratives exploitant les ressources de l'informatique ubiquitaire permettent à des utilisateurs nomades d'accéder à des fonctionnalités offertes par des fournisseurs de services ambiants. La plupart de ces applications n'offrent cependant pas à l'heure actuelle la possibilité à leurs clients de collaborer ensemble et se limitent seulement à l'accès à des applications fournies par une infrastructure dédiée. Afin de rendre possible l'exécution d'applications collaboratives entre clients nomades, nous avons défini une architecture distribuée permettant l'exécution d'un processus collaboratif ubiquitaire. Dans un tel environnement d'exécution, les utilisateurs nomades collaborent grâce à de complexes interactions qui peuvent être vues comme une extension du concept de processus collaboratif centralisé. L'architecture de processus collaboratif ubiquitaire introduit un support d'exécution permettant à des utilisateurs nomades travaillant à proximité les uns des autres de partager leurs ressources et d'accéder à celles des autres selon un plan d'exécution prédéfini ou *workflow*. Ce support d'exécution est caractérisé par les points suivants :

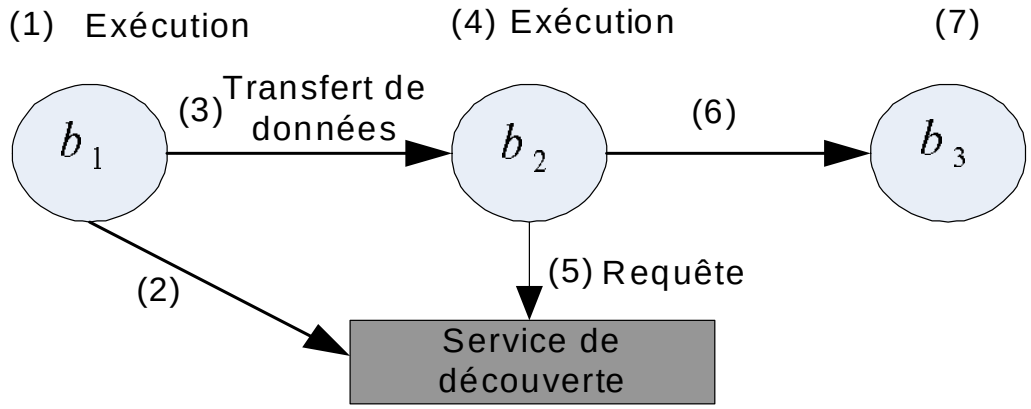

Figure 2: Principes d'exécution du *workflow* ubiquitaire

- **Une architecture distribuée**: la gestion des échanges ayant lieu au cours de l'exécution du processus collaboratif entre clients nomades est assurée par les clients eux-mêmes de telle sorte qu'aucune infrastructure dédiée ne soit nécessaire.

- **Un processus dynamique d'assignation des acteurs aux différentes étapes d'exécution**: les ressources mises à disposition par les clients peuvent être découvertes au moyen d'un service spécialisé suivant les besoins fonctionnels nécessaires à la poursuite de l'exécution d'un processus collaboratif donné.

Les étapes de l'exécution d'un processus collaboratif ubiquitaire sont décrites à la figure 2. Ayant défini une représentation abstraite d'un processus collaboratif, c'est à dire l'ensemble des étapes fonctionnelles nécessaires à l'exécution d'un processus collaboratif, un utilisateur démarre le processus. Il exécute un premier ensemble de tâches (1) avant de rechercher dans son environnement immédiat, via le service de découverte, un collaborateur susceptible d'exécuter le sous ensemble d'opérations suivant dans le plan d'exécution de son processus collaboratif (2). Dès cette phase de découverte achevée, l'ensemble des données nécessaires au processus collaboratif est transmis par l'utilisateur à ce nouveau collaborateur (3) permettant la poursuite de l'exécution du processus (4). L'enchaînement précédent : découverte d'un utilisateur nomade, transfert de données entre utilisateurs et exécution d'un ensemble de tâches se répète itérativement jusqu'à ce que l'ensemble des opérations prévu par le plan du processus collaboratif soit achevé. Afin d'optimiser les ressources des utilisateurs nomades, le mode d'exécution est sans mémoire de telle sorte que les acteurs d'une instance d'un processus collaboratif ne gardent aucune information résiduelle nécessaire à la suite de son exécution et puissent se déconnecter dès qu'ils en ont terminé avec l'exécution des opérations qui leur ont été assignées. Toutes les données nécessaires à l'exécution d'une telle collaboration sont donc transférées entre chaque étape d'exécution d'un utilisateur vers le suivant. Les échanges de données entre acteurs nomades incluent le plan d'exécution du processus collaboratif pour permettre une exécution cohérente de celui-ci.

La section suivante décrit une implémentation de cette architecture basée sur les technologies "Web services".

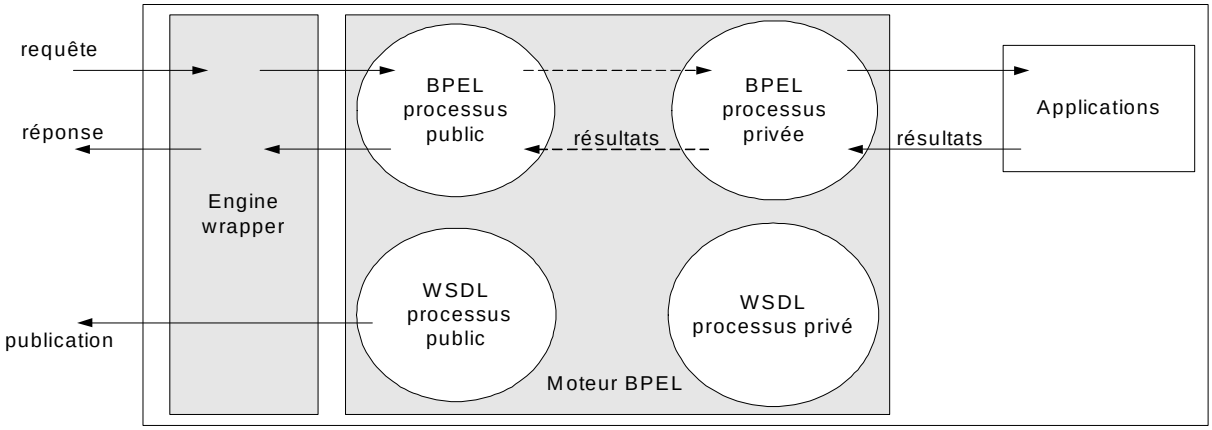

Figure 3: Implémentation de l'architecture de *workflow* ubiquitaire

Implémentation basée sur le paradigme d'Architecture Orientée Services

Les acteurs impliqués dans une instance d'un processus collaboratif ubiquitaire sont en général membres d'une organisation (entreprise, hôpital, etc.) et doivent s'assurer qu'aucune information sensible n'est divulguée par leur participation à un processus collaboratif. En étant acteur de l'exécution d'un processus collaboratif, ils partagent d'une part leurs ressources applicatives et doivent d'autre part assurer la gestion locale de leur participation au processus. Chaque acteur implémente de ce fait deux modules logiciels : le premier regroupe l'ensemble des applications dont il dispose et le second est un moteur de processus collaboratif. Les applications logicielles implémentées par un utilisateur donné doivent rester la propriété de l'organisation à laquelle il appartient et donc demeurer privées. Elles ne peuvent être appelées que par cet utilisateur et le moteur de processus collaboratif local doit donc jouer le rôle d'une interface entre ces applications et les autres acteurs impliqués dans l'exécution d'un processus collaboratif. Notre implémentation décrite par la figure 3 se base sur le langage Business Process Execution Language (BPEL) qui permet de spécifier des processus collaboratifs. Deux processus BPEL, un public et un autre privé, sont déployés sur le moteur de *workflow* afin de remplir ce rôle d'interface. Le processus public est contacté par les autres acteurs nomades tandis que le processus privé ne reçoit de requêtes que du processus public. Le processus public met donc à disposition les ressources d'un utilisateur et assure l'interface avec les acteurs potentiels alors que le processus privé implémente les logiciels fournissant ces ressources.

Par ailleurs, afin de ne pas modifier l'implémentation d'un moteur de *workflow* déjà existant, nous avons choisi de développer un module externe appelé "engine wrapper" afin d'implémenter les primitives d'exécution liées au design de l'architecture de *workflow* ubiquitaire.

Consistance transactionnelle du *workflow* ubiquitaire

L'architecture de *workflow* de par sa nature distribuée et dynamique n'offre aucune garantie en ce qui concerne la fiabilité et la tolérance aux pannes. Nous présentons dans cette partie les solutions que nous avons mises en oeuvre pour répondre à cette limitation. Nous décrivons dans un premier temps les prérequis en termes de consistance transactionnelle associés à l'exécution

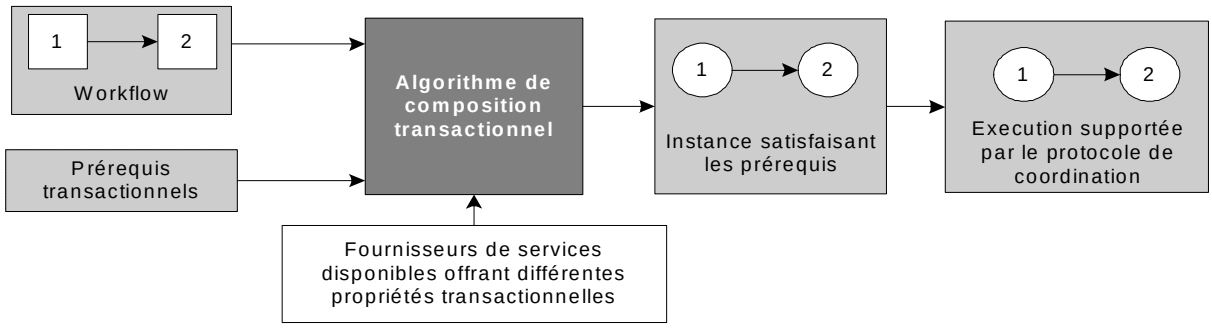

Figure 4: Methodologie

d'un processus collaboratif ubiquitaire avant de présenter les solutions que nous avons conçues pour y répondre.

Prérequis trasanctionnels

L'exécution d'un processus collaboratif ubiquitaire doit principalement satisfaire les deux contraintes transactionnelles suivantes:

- **Atomicité relachée**: certains résultats intermédiaires produits par l'exécution d'un *workflow* peuvent être conservés bien que l'exécution d'autres tâches du *workflow* ait été un échec. Les contraintes d'atomicité strictes des systèmes de base de données traditionnels ne sont en effet pas adaptés à l'exécution d'un *workflow*.
- **Assignation dynamique des acteurs du *workflow***: les exécutants du *workflow* peuvent être sélectionnés durant son exécution en fonction de différents paramètres.

Les solutions existantes en matière de coordination transactionnelle sont inadaptées pour satisfaire, par exemple, la deuxième contrainte. Il n'existe en effet pas de solution pour combiner les fonctionnalités offertes par différents fournisseurs de services et prenant en compte les prérequis transactionnels fixés par les designers d'un *workflow*. Il est en effet seulement possible de valider une instance de *workflow* une fois que celle-ci a été formée mais malheuresement pas au moment de sa construction. Nous proposons dans cette thèse un algorithme de composition transactionnel résolvant ce problème.

Méthodologie

Notre but est d'assurer la coordination transactionnelle de certaines tâches du *workflow* lorsque l'exécution le requiert. Notre approche consiste en la partition du *workflow* en différentes zones appelées zones critiques dont les tâches nécessitent une coordination transactionnelle. Afin d'assurer la coordination transactionnelle de ces tâches, nous choisissons un protocole hiérarchisé dont la gestion est assurée de manière centralisée par l'initiateur d'une zone critique. Un *workflow* ubiquitaire est exécuté par des acteurs qui peuvent être sélectionnés au cours de son exécution. Considérant la variété des propriétés offertes par les acteurs potentiels qui peuvent être assignés à l'exécution du *workflow*, nous supposons que ces acteurs peuvent offrir en plus

de fonctionnalités différentes, des propriétés transactionnelles différentes. Par exemple, un acteur peut offrir la possibilité d'annuler les effets d'une exécution alors que d'autres n'offrent pas cette propriété. Il devient par conséquent nécessaire de choisir les acteurs du *workflow* non seulement en fonction de leurs fonctionnalités mais aussi en tenant compte des propriétés transactionnelles qu'ils offrent. Dans ce but, nous proposons un algorithme de composition transactionnelle dont le but est de créer une instance de *workflow* satisfaisant des prérequis transactionnels fixés par les designers de ce *workflow* afin d'assurer que l'exécution de ce dernier soit consistante. Cette approche suppose donc que tous les acteurs du *workflow* soient découverts avant son instanciation.

Afin de remplir ces objectifs, nous proposons dans le chapitre 3 la méthodologie suivante. Nous présentons dans un premier temps une sémantique permettant de spécifier les propriétés transactionnelles offertes par les fournisseurs de services. Cette sémantique est nécessaire afin de mettre en relation les propriétés transactionnelles requises par l'exécution d'une tâche et celles offertes par les acteurs potentiels pouvant être assignés à celle-ci. Nous définissons ensuite un outil basé sur cette sémantique qui permet aux designers d'un *workflow* de définir les contraintes transactionnelles que l'exécution de ce dernier doit vérifier. Les acteurs sont ensuite assignés aux différentes tâches du *workflow* en fonction des prérequis définis par les designers. Enfin, l'instance du *workflow* résultant de l'algorithme de composition transactionnelle peut être coordonnée en fonction de règles de coordination dérivées du processus de composition. Cette méthodologie est résumée par la figure 4.

Implémentation

Les contributions théoriques décrites ci-dessus ont été implémentées dans le cadre du paradigme d'Architecture Orientée Services en utilisant les technologies suivantes : OWL-S [OWL03], BPEL [BPE], et WS-Coordination [La05c]. En particulier, nous avons développé une infrastructure de coordination transactionnelle composée de deux modules. Le premier module intègre un "matchmaker" OWL-S auquel nous avons rajouté des fonctionnalités transactionnelles implémentant notre algorithme de composition. Le deuxième module est l'implémentation du protocole de coordination qui a été développé en suivant un modèle similaire à celui du standard WS-Coordination.

Sécurité du *workflow* ubiquitaire

Cette section présente le design de protocoles de sécurité permettant d'assurer une exécution sécurisée des processus collaboratifs ubiquitaires. Dans un premier temps, nous présentons les prérequis de sécurité associés à l'exécution de *workflow* ubiquitaire avant de présenter la solution que nous avons développée afin de satisfaire à ces contraintes.

Prérequis de sécurité

Par rapport aux processus collaboratifs centralisés classiques, l'exécution distribuée de processus collaboratifs introduit des contraintes en termes de sécurité informatique dues à l'absence

d'un point de coordination prenant en charge la gestion des échanges entre les acteurs impliqués. Par conséquent, de simples primitives d'exécution vérifiées dans le cas centralisé comme l'adhérence de l'exécution d'un processus avec un plan prédéfini ne sont plus assurées. Nous avons classifié les prérequis que nous avons identifiés vis-à-vis de la sécurité informatique en trois catégories : autorisation, preuves d'exécution et protection des données du processus collaboratif.

Autorisation La principale contrainte pour un processus collaboratif est d'assurer que seuls des acteurs autorisés sont assignés aux différentes étapes d'exécution. Dans le cas décentralisé et ubiquitaire, ces acteurs peuvent être sélectionnés au moment même de l'exécution par d'autres acteurs au moyen d'un service de découverte. La procédure de sélection de ces acteurs potentiels doit faire ici correspondre les contraintes de sécurité spécifiées par le plan d'exécution du processus collaboratif avec les droits que possèdent les acteurs disponibles.

Preuves d'exécution Comme mentionné plus haut, l'adhérence de l'exécution d'un processus avec un plan prédéfini n'est pas assurée dans le cas de processus collaboratifs décentralisés. Sans aucun coordinateur de confiance auquel se référer, les acteurs impliqués doivent avoir la possibilité de vérifier au cours de l'exécution d'un processus que cette dernière satisfait le plan qui a été au préalable défini afin qu'aucun acteur malveillant ne puisse fabriquer un faux processus collaboratif dans le but de nuire à des tiers.

Protection des données Dans le cas décentralisé, toutes les données du processus collaboratif sont transférées entre les acteurs impliqués. Ceci nécessite des mécanismes spécifiques pour en assurer l'intégrité, la confidentialité et en contrôler l'accès dans la mesure où l'intégrité de l'exécution doit être assurée :

- Le contrôle d'accès sur les données du processus collaboratif est basé sur le plan d'exécution qui spécifie pour chaque étape le sous-ensemble de ces données qui sont accessibles en lecture,
- Seul un sous-ensemble de ces données doit être modifié par l'exécution d'une étape donnée.

La solution que nous avons développée pour satisfaire à ces contraintes est décrite dans la section suivante.

La solution

Assurer la protection des données du processus collaboratif est au coeur des mécanismes de sécurité nécessaires pour satisfaire les contraintes que nous avons identifiées. Nous associons à chaque étape de l'exécution d'un processus distribué deux paires de clefs : la première appelée "paire de clefs politique de sécurité" est utilisée dans le processus de sélection des acteurs potentiels au cours d'un processus collaboratif, la seconde appelée "paire de clefs d'étape" protège l'accès aux données du processus collaboratif. Nous proposons un procédé de distribution de clefs au cours duquel les clefs privés d'étape $(SK_i)_{[1,n]}$ sont mises à disposition des acteurs sélectionnés au moment même de l'exécution de l'étape qui leur a été assignée. Il est à noter que

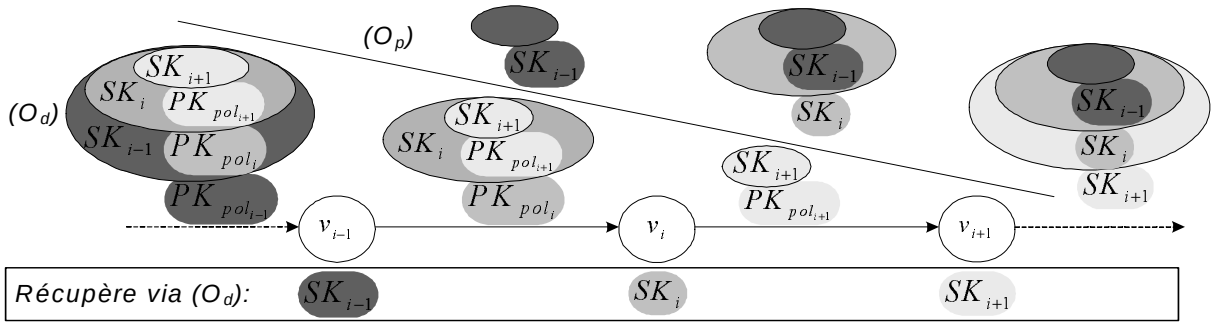

Figure 5: Protocoles de sécurité: principes

l'identité de ces acteurs n'est a priori connue qu'au moment même de l'exécution du processus : une distribution de clefs au préalable n'est donc pas une solution envisageable. La conception de ce mécanisme se base sur les techniques de chiffrement concentriques ou chiffrement en oignon utilisant les clefs publiques de politique PK_{pol_i} associées à chaque étape de l'exécution du processus collaboratif. En outre, des preuves d'exécution doivent être produites au cours de l'exécution d'un processus collaboratif pour assurer la conformité de ce dernier avec le plan d'exécution préalablement défini. Dans ce but, nous utilisons aussi les techniques de chiffrement concentrique pour construire une structure regroupant les signatures des différents acteurs calculées avec les clefs privées d'étape.

La figure 5 présente les principes de ces deux mécanismes. A chaque étape de l'exécution du *workflow* les acteurs du workflow pèlent une couche de l'oignon O_d afin de récupérer la clef privée d'étape qui leur permettra de lire et de modifier les données auxquelles ils ont accès. Dès qu'un acteur termine l'exécution de l'étape à laquelle il a été assigné, il signe un oignon de preuve O_p avec la clef privée d'étape qu'il a récupérée en pelant O_d.

Propriétés de sécurité

L'ensemble des mécanismes de sécurité que nous avons développés vérifient plusieurs propriétés détaillées dans le chapitre 4 de cette thèse. Ces propriétés concernent en particulier la conformité de l'exécution du *workflow* ubiquitaire avec sa spécification ainsi que l'intégrité des données échangées durant une instance de processus collaboratif ubiquitaire. Il est cependant à noter que ces propriétés dépendent du protocole de distribution des clefs privées de politique aux acteurs potentiels d'une instance de *workflow* ubiquitaire.

Conclusion

Dans cette thèse de doctorat nous avons présenté le design et l'implémentation d'une infrastructure de gestion de *workflow* distribuée supportant l'assignation d'acteurs en cours d'instance. Nous avons aussi proposé des protocoles de sécurité informatique et de coordination transactionnelle afin de garantir une exécution sécurisée et fiable pour les instances de *workflow* supportée par cette infrastructure. Cette infrastructure satisfait les contraintes d'exécution posées par les applications collaboratives modernes tout en leur offrant la flexibilité qu'elles requièrent.

Les travaux de recherche présentés dans cette thèse peuvent être poursuivis suivant différents axes. Tout d'abord notre étude se limite aux plus simples modèles d'exécution de *workflow* tels que l'exécution séquentielle ou concurrente. Cet aspect pourrait être étendu à des modèles plus complexes incluant par exemple la synchronisation entre branches parallèles. D'autre part, les aspects de sécurité informatique ayant trait à la composition sécurisée d'acteurs d'un *workflow* ne sont que brièvement traités dans cette thèse et une piste possible pourrait être la prise en compte, par le biais d'un algorithme de composition à la manière de ce qui a été présenté dans ce travail pour les aspects transactionnels, des assertions de sécurité satisfaites par les acteurs potentiels d'un *workflow*. Enfin, notre approche de sécurité pourrait être étendue à des modèles de politiques de sécurité plus complexes pour pallier les problèmes liés par exemple à la concurrence entre deux acteurs participant à une même instance de *workflow*.

Abstract

With the emergence of the Internet, electronic commerce and Web-based applications have become the standard support for Business-to-Business and Business-to-Costumer collaborations. The concept of workflow or business process has been the main enabler concept for such collaborative applications. Workflow technologies indeed make it possible to leverage the functionalities of multiple service providers to build value-added services. Typical business processes however rely on a centralized coordinator that is in charge of assuring the management and the control tasks of the process execution. New trends in collaborative business applications call for flexibility to enable for instance the execution of business collaborations that can be built on the fly without the need of a dedicated coordination infrastructure. As a result, the usual centralized coordination paradigm is no longer suitable to adequately support the execution of most recent business applications. In this dissertation we present a decentralized workflow management system to overcome this limitation.

The main contribution of this thesis is the design and implementation of a full-fledged decentralized workflow management system. The workflow architecture denoted pervasive workflow architecture that we developed supports the execution of business processes in environments whereby computational resources offered by each business partner can potentially be used by any party within the surroundings of that business partner. It also features the runtime assignment of business partners to workflow tasks in order to provide the adequate flexibility to support dynamic collaborations of business partners. This flexibility however comes at the expense of security and reliability and introduces new research challenges as opposed to usual workflow management systems in terms of security and fault management. To cope with the latter, we first propose an adaptive transactional protocol to support the execution of pervasive workflows. This transactional protocol features an algorithm enabling the selection of partners not only according to functional requirements but also to transactional ones. Besides, we introduce new security mechanisms capitalizing on onion encryption techniques and security policy models in order to assure the integrity of the pervasive workflow execution and to prevent workflow instance forging.

Contents

List of Figures

List of Tables

Notations and Accronyms

Mathematical notations

$\mathcal{M}$	Message space
$\mathcal{C}$	Ciphertext space
$\mathcal{K}$	Key space
$(\mathbb{G}_1, +)$	Additive group of order q for a prime q
$(\mathbb{G}_2, .)$	Multiplicative group of order q for a prime q
$\hat{e}$	Bilinear map
$\mathbb{Z}_q$	Integers modulo q: the set of integers $0, 1, ..., n-1$
$\mathbb{Z}_q^*$	The multiplicative group of $\mathbb{Z}_q$
h	Cryptographic hash function
PK	Public key
SK	Private key
$\{m\}_{PK}$	Using public key PK on a message m (e.g. encryption)
$\{m\}_{SK}$	Using private key SK on a message m (e.g. signature)
W	Workflow
v_i	Workflow vertex
b_i	Business partner

Accronyms

UML	Unified Modeling Language
DOM	Document Object Model
JSP	Java Server Pages
SVG	Scalable Vector Graphic
XML	Extensible Markup Language
Ajax	Asynchonous Java script and XML

BPEL	Business Process Execution Language
CDL	Choreography Description Language
WSDL	Web Service Definition Language
UDDI	Universal Description, Discovery and Integration
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
SMTP	Simple Mail Transfer Protocol
RSA	Rivest Shamir Adleman
IBE	Identity-Based Encryption
ECB	Electronic CodeBook
CBC	Cipher Block Chaining
JCE	Java Cryptographic Extension
SOA	Service Oriented Architecture
SOC	Service Oriented Computing
PDA	Personal Digital Assistant
RFID	Radio Frequency IDentification
URI	Uniform Resource Identifier

Publications based on this Thesis

Here is the list of papers that have been written since the beginning of this Ph.D. thesis.

International Conferences

[MM05] "**Enabling pervasive execution of workflows**"[1] [2]
F. Montagut and R. Molva, in the proceedings of CollaborateCom 2005, 1st IEEE International Conference on Collaborative Computing:Networking, Applications and Worksharing, December 19-21, 2005, San Jose, USA.
Pervasive workflow architecture (Chapter 2)

[MM06a] "**Augmenting Web services composition with transactional requirements**"[1] [2]
F. Montagut and R. Molva, in the proceedings of ICWS 2006, IEEE International Conference on Web Services, September 18-22, 2006, Chicago, USA. ***This paper received the conference runners-up award***.
Theoretical grounds of our work on reliability issues (Chapter 3)

[MM06b] "**Towards transactional pervasive workflows**" [1] [2]
F. Montagut and R. Molva, in the proceedings of EDOC 2006, 10th IEEE International EDOC Conference "The Enterprise Computing Conference", 16-20 October 2006, Hong-Kong.
Specification of the coordination protocol we designed (Chapter 3)

[MM07a] "**Enforcing Integrity of Execution in Distributed Workflow Management Systems**" [1] [2]
F. Montagut and R. Molva, in the proceedings of SCC 2007, 2007 IEEE International Conference on Services Computing, 9-13 July 2007, Salt Lake City, USA.
Specification of the security solutions we designed (Chapter 4)

[MM07b] "**Traceability and Integrity of Execution in Distributed Workflow Management Systems**" [1] [2]
F. Montagut and R. Molva, in the proceedings of ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007.
Specification of the security solutions we designed (Chapter 4)

Journal Papers

[MMG08b] "**The Pervasive Workflow: A Decentralized Workflow System Supporting Long Running Transactions**" [1]
F. Montagut, R. Molva and S. Golega, to appear in IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews. Special Issue on Enterprise Service Computing and Industrial Applications.
Extended version of [MM06b]

[MMG08a] "**Automating the composition of transactional Web services**" [1]
F. Montagut, R. Molva and S. Golega, To appear in International Journal on Web Services Research, Idea Publishing, 2008.
Extended version of [MM06a]

Book chapters

[PWL+07] "**Utilisation des informations contextuelles pour assurer la sécurité d'un processus collaboratif distribué : un exemple dans l'e-Santé**"
J.-C. Pazzaglia, K. Wrona, A. Laube, F. Montagut, L. Gomez, Y. Roudier, and S. Trabelsi. OFTA, Paris, France, 2007.
Book chapter in French including a presentation of the pervasive workflow

[Kha08] "**Automating the composition of transactional Web services**" (book chapter) [1]
F. Montagut, R. Molva and S. Golega, "Managing Web Services Quality: Measuring Outcomes and Effectiveness", IGI Global, to appear in 2008
Invited book chapter based on the results presented in Chapter 3

[1] The paper was presented in a conference by the author of this dissertation
[2] The main author of the paper is the author of this dissertation

Introduction

The skill of writing is to create a context in which other people can think.
- Edwin Schlossberg -

The notion of workflow or business process refers to the automation of business procedures towards reaching a business goal such as for instance, “booking an airline ticket", and executed based on a pre-defined plan specifying the overall sequence of operations required to be performed in order to achieve the defined goal [Hol95]. Within the execution of a workflow, documents or data are transferred between participants that can be simple human users or machines and in order to support these business interactions, a workflow management system is usually in charge of routing messages between participants based on the defined sequence of operations. As for any business applications, workflow-based applications require stringent approaches to provide appropriate security and transactional mechanisms in order to assure their sound execution. Typical business processes rely on a centralized coordinator that is in charge of assuring the management and the control tasks of the process execution. This coordinator is however considered a performance bottleneck to enable the execution of business collaborations that can be built on the fly without the need of a dedicated coordination infrastructure. In this dissertation we present a decentralized workflow management system to overcome this limitation.

The main contribution of this thesis is the design and implementation of a decentralized workflow management system and that of the underlying security and transactional protocols. The workflow architecture denoted pervasive workflow architecture that we developed supports the execution of business processes in environments whereby computational resources offered by each business partner can potentially be used by any party within the surroundings of that business partner. It also features the runtime assignment of business partners to workflow tasks in order to provide the adequate flexibility to support dynamic collaborations of business partners.

The remainder of this introductory chapter is organized as follows. We first provide an overview the concept of workflow on which rely modern business applications. We then motivate the need of decentralized workflow infrastructure to offer the adequate flexibility required by state of the art business applications. Our choice of relying on the Service Oriented Computing paradigm whose underlying technologies are today's de facto standard to support collaborative applications is later on motivated. We finally outline the new requirements in terms of reliability and security introduced by the execution of collaborative business applications in the decentralized setting, before highlighting the structure and contributions of this thesis.

Workflow-based collaborative business applications

With the emergence of the Internet, electronic commerce and Web-based applications have become the standard support for Business-to-Business and Business-to-Costumer collaborations. Relying on inexpensive communication means and dynamic graphical interfaces, these collaborative business applications integrate complex services ranging from stock management components to secure payment platforms in a way that is perfectly seamless for end-users. The concept of workflow or business process has been the main enabler concept for such collaborative applications. Workflow technologies indeed make it possible to leverage the functionalities of multiple service providers to build value-added services so that workflows are today's standard for the orchestration of both inter and cross-organizational collaborative applications.

Major software vendors including IBM, Microsoft and SAP have promoted the development of workflow-based applications [MBB+03, KH05, biz05] over the past years. In fact, one of the key feature offered by workflow technologies lies within the automation of the sequence of operations necessary to reach a business goal. As a result, multiple instances of a single business process can be launched concurrently easing the management and control of business applications. Examples of workflow-based applications range from quite simple applications such as airline booking systems, electronic shopping on a merchant website, etc. that enable users to select goods remotely, manage their orders with their electronic cart, pay electronically and track the shipment to complex cross-organizational collaborations involving many different parties. The execution support of these collaborative applications usually relies on dedicated infrastructure whose deployment can take place based on two main types of architecture:

- **Centralized**: a dedicated coordinator is in charge of routing messages between the business partners or services involved in a workflow instance based on the workflow specification that states the sequence of operations necessary to complete the workflow. This architecture is the most common one.
- **Distributed**: each business partner involved in the workflow instance implements a part of the complete workflow corresponding to the tasks he is assigned to within the workflow. This is the standard architecture for complex collaborations that span over organizational boundaries.

These typical workflow architectures have however major drawbacks that limit the high number of applications that could potentially leverage on the concept of workflow. First, these workflow systems are quite static in that the business partners or organizations involved in the execution of a workflow instance are often assigned to workflow tasks prior to the process instantiation thus restricting the flexibility at runtime. In the centralized setting, the centralized point of coordination may be a performance bottleneck for some collaborative applications and may raise scalability issues when used concurrently by several clients. In the decentralized setting, the deployment of the business processes on peers' side is often done prior to the process instantiation and requires a kind of agreement between business partners beforehand. Finally, no solution combining both runtime business partner assignment and runtime deployment of business processes has yet been designed. As a result, current architectures available in the field of workflow management systems do not seem to offer the adequate flexibility to meet the requirements introduced by modern collaborative applications such as the ones outlined in the next section.

New paradigms in collaborative business applications

Mark Weiser described sixteen years ago [Wei91] a world wherein computers would be seamlessly integrated into everyday life: the pervasive or ubiquitous computing paradigm. What was foreseen by Weiser has nowadays become a reality [Den01] with the increasing computing power available on devices such as personal assistants or sensors and RFID whose size gets smaller and smaller. Software vendors have identified quite early the new business opportunities associated with the ubiquitous computing trend and developed new products wherein artifacts (i.e. devices with embedded computational and communication functionalities) play central role in the business logic of collaborative applications. These artifacts can be integrated at different levels within the business logic of applications:

- **Graphical User Interface**: handheld devices such as PDA can be integrated to serve as the interface between users and ambient services provided for example by an intelligent home or a shop [RM04]. At this level no complex interactions are taking place between artifacts and other ambient systems, the main goal of these applications is just to enhance user experience.

- **Information provider**: sensors can for instance deliver contextual information at runtime so that the application business logic is modified dynamically based on the changes occurring within its execution environment. Such applications are called context-aware applications. At this level, artifacts have a direct impact on application business logic yet the interactions between applications and artifacts are quite limited.

- **Service provider**: the ever increasing capacity of artifacts allows them play the role of application provider themselves. Many pieces of work have recently been presented to study the feasibility of deploying a Web service on different devices such as for instance

a watch or a PDA [BMNR03, WTK02, LW05, CYT04, LCY+04]. At this level, complex interactions take place between mobile users who can share their applications available on their devices including phones, MP3 readers and the like, anywhere at any moment. These interactions may not in fact require any dedicated server infrastructure and can use transmission mediums such as bluetooth or wireless LAN.

Interactions between artifacts within these examples do not extend beyond data and resource sharing. In this thesis, our objective is to reach a further stage of interaction by enabling the collaboration amongst users so that the latter can actively participate in the execution of complex collaborative applications in order to build value-added services. There are many different use cases for such flexible collaborative applications, as illustrated by the following examples:

- **Emergency response team management**: in this example, workflow technologies can be used to coordinate the efforts between some actors assigned to manage an accident without the need of a dedicated infrastructure deployed on site. In this case, team members can be dynamically assigned to tasks depending on contextual information including their availability or proximity to the accident location. A concrete example of such a scenario is depicted in appendix D whereby a physician, a pharmacist and a social worker collaborate to treat a patient whose health condition is remotely monitored.
- **Dynamic collaboration within an airport**: in this example, workflow technologies can be used to initiate the dynamic collaboration of mobile workers that meet in an airport and decide to do business together without the need of knowing each other beforehand or relying on any infrastructure implemented within the airport.
- **Peer-to-peer collaboration over the Internet**: flexible collaborations are in fact not only limited to pervasive environments but can also take place over the Internet, in this case as for the previous example, business partners can initiate dynamic cross-organizational workflows over the Internet.

We present in this dissertation the design of a distributed workflow management system that meets the requirements associated with the execution of dynamic collaborations between business partners without the need of a dedicated infrastructure or that of knowing each others' identities beforehand. In the next two sections we introduce the set of technologies most appropriate for the deployment of such dynamic collaborations before examining the challenges in terms of security and reliability raised by this new collaboration paradigm.

Service Orient or Be Doomed! [BS06]

The title "Service Orient or Be Doomed!" of a recent book on the state of the art in software technologies [BS06] perfectly describes the current trend in the Information Technology industry. The Service Oriented Computing paradigm [AHMS06, MBB+03] and the underlying Web

services technologies are indeed the current industry standards to implement collaborative business applications. Promoted by leading industry vendors including SAP [KH05] or Microsoft [biz05], service orientation has become the de facto standard for service providers to implement and advertise the services they offer. Relying on simple standards such as XML [XML] or HTTP protocol, Web services technologies offer basic primitives that allow service providers to leverage their business functionalities by providing them with the means to expose, combine, integrate and coordinate their services at low cost over the Web. Web services technologies encompass a lot of specifications aiming at providing reliable and secure message-based exchanges between software components in a loosely-coupled way. The ultimate goal of Web services indeed is to offer an interoperable framework that does not depend on any specific implementation for Business-to-Business and Business-to-Consumer interactions. In fact, one major advantage of Web services technologies lies within their capability to enable the cross-organizational cooperation of distant Web-based services without the need to adapt component interfaces to any specific implementation paradigm.

The Service Oriented Architecture paradigm and the design of a workflow management system supporting dynamic collaborations of business partners in particular are governed by a set of principles as follows.

- **Loosely coupled interactions**: interoperability of business platforms is a major requirement to enable seamless interactions between business partners.
- **Service composition**: services can be combined or composed in order to build value-added services offering complex functionalities to clients.
- **Service discovery**: service providers can be easily contacted by clients based on a match-making process between the functionalities they expose and what is requested in terms of functionalities by the latter.

These three principles are the basic functionalities required to meet the architectural requirements identified above for the design of a workflow system supporting dynamic collaborations. To that respect, service orientation and the underlying Web services technologies appear to be the most appropriate solution to implement the theoretical results that are presented in this thesis.

New requirements for security and reliability

The design of a workflow management system should not only take into account functional requirements derived from the workflow applications whose execution it is meant to support but it should also meet security and reliability requirements raised by the distributed workflow system. Based on the architectural requirements we identified above, the flexibility required

by modern collaborative applications comes at the expense of security and reliability and introduces new research challenges as opposed to usual workflow management systems in terms of security and fault management. These new requirements are presented in this section.

Reliability and transactional consistency

One key requirement for a workflow management system is to ensure that the outcome reached by business processes whose execution it supports are consistent. Considering the lack of reliability akin to distributed or pervasive environments, assuring data and transactional consistency of the outcome reached by workflow instances supported by the proposed workflow management system is a challenging issue. There are in fact mainly two requirements that are brought up by the design of a suitable coordination protocol assuring the consistency of modern collaborative applications:

- **Relaxed atomicity**: some intermediate results produced within the execution of a collaborative application may be kept despite the failure to execute some other operation.
- **Dynamic assignment of business partners**: modern collaborative applications are dynamic in that the peers who can be assigned to the execution of some tasks can be selected at runtime depending on, for instance, contextual information.

Transactional protocols designed so far to support the execution of traditional collaborative applications do not offer the adequate flexibility to cope for example with the runtime assignment of computational tasks to business partners. As a result, new solutions have to be designed and we propose in this thesis a transactional coordination protocol that offer adequate features to support the execution of flexible collaborative applications.

Security

Modern workflow-based applications raise new security requirements compared to traditional collaborative applications because of their dynamicity and in some cases unusual execution environments. Being dynamically composed without the need of a dedicated infrastructure, they may not indeed rely on a trusted centralized coordination mechanism to manage their execution. As a result, basic security features including the assurance that the collaboration is executed according to a sequence of operations specified by the collaboration initiator or agreed upon by the participants may no longer be enforced within the execution. In addition, enforcing access control policies on the documents processed by business partners or keeping track of the identity of the different actors involved in the execution of a collaborative application become critical issues when relying on a business partner selection process that can be performed at runtime based on some contextual information. Existing security mechanisms designed for the

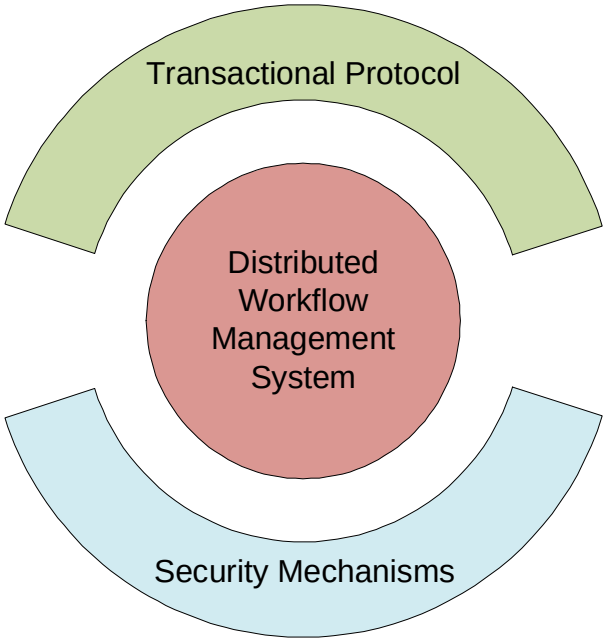

Figure 6: Thesis organization in terms of architectural building blocks

execution of traditional business applications do not offer adequate solutions to solve these new security issues introduced by state-of-the-art collaborative applications. In this thesis, we thus suggest the design of appropriate security mechanisms in order to meet the security requirements associated with the execution of flexible workflow-based applications.

Structure and contributions

Each chapter of this report introduces a building block in the design of a complete solution to support the execution of modern workflow-based collaborative applications. The three main architectural components we mentioned in this section are depicted in figure 6. The central component is in fact the workflow management system that should meet the functional requirements we identified in order to offer adequate support for the execution of collaborative applications in environments that may not offer a dedicated infrastructure. In order to provide adequate guarantees in terms of security and reliability for the execution of collaborative applications, the workflow system is supported on the one hand by a transactional protocol that is in charge of managing faults that may occur during business process executions and on the other hand security mechanisms that enforce security requirements. The remainder of this manuscript is outlined as follows.

Chapter 1 In the first chapter, we give some introductory information on the technical background material that will be used throughout this thesis. We especially introduce some

basic definitions associated with the concept of workflow and give a broad overview of the Service Oriented Architecture paradigm and the underlying Web services technologies.

Chapters 2, 3 and 4 are the core of this thesis and specify the solutions we designed towards reaching the objectives we set in this introductory chapter. The reader is invited to read them linearly, even though they describe solutions that can be used independently in other context. There are in each of these chapters some cross references that require a basic understanding of the concepts specified in previous chapters.

Chapter 2 As opposed to traditional workflow management systems, the execution of modern workflow-based applications may not rely on a dedicated infrastructure to assure the workflow coordination task. As a result, there are many challenging issues and requirements relevant to the execution of business processes in the pervasive setting that will be developed in the course of this thesis. In chapter 2, we focus on the workflow coordination and management tasks and suggest the design of a workflow management system, denoted pervasive workflow, supporting distributed execution of workflows in pervasive environments. The pervasive workflow architecture features a fully decentralized control and supports dynamic assignment of workflow tasks to business partners so that it meets the following target requirements raised by the execution of flexible collaborative applications:

- **Fully decentralized**: the management of the workflow execution is distributed amongst the business partners taking part in a workflow instance in order to cope with the lack of dedicated infrastructure,
- **Dynamic assignment of business partners to workflow tasks**: the business partners in charge of executing the workflow can be discovered at runtime based on available resources or contextual information.

In addition, we also present in this chapter the implementation of the pervasive workflow model based on Web services technologies.

Chapter 3 In chapter 3, we tackle the issue of reliability within pervasive workflow instances and propose an adaptive transactional protocol to assure their coordination from the transactional point of view. The execution of this protocol takes place in two phases. First, business partners are assigned to tasks using an algorithm whereby workflow partners are selected based on functional and transactional requirements. Given an abstract representation of a process wherein business partners are not yet assigned to workflow tasks, this algorithm enables the selection of partners not only according to functional requirements but also based on transactional ones. The resulting workflow instance is compliant with the defined consistency requirements and its execution can be easily coordinated as our algorithm also provides coordination rules. The workflow execution further proceeds through a hierarchical coordination protocol managed by the workflow initiator and controlled using the coordination rules computed as an outcome of the partner assignment procedure.

We also describe an implementation of the theoretical results presented in this chapter based on Web services technologies.

Chapter 4 In this chapter, we present security solutions in order to assure the secure execution of the pervasive workflow instances. These mechanisms capitalize on onion encryption techniques [SGR97] and security policy models in order to assure the integrity of the distributed execution of workflows, to prevent business partners from being involved in a workflow instance forged by a malicious peer and to provide business partners' identity traceability for sensitive workflow instances. The design of the suggested mechanisms is strongly coupled with the runtime specification of decentralized workflow management systems in order to ease their integration into existing distributed workflow management solutions. We also specify how the security mechanisms presented in this chapter can be integrated into the transactional coordination protocol outlined in chapter 3. An implementation of the security mechanisms presented in this chapter is also specified using identity based encryption techniques and Web services technologies.

Appendices Appendix D presents an example of an actual collaborative application whose execution is supported by the pervasive workflow architecture. The other appendices provide details on the implementation of the architectural building blocks presented in this thesis. The reader is also invited to read appendices in a linear fashion.

The research work performed by the author in the scope of this thesis resulted in a number of scientific publications that contain the main ideas presented in this manuscript [MM05, MM06a, MM06b, MM07a, MM07b, MMG08b, MMG08a, Kha08].

Chapter 1

Preliminaries and Technical Background

The path to our destination is not always a straight one. We go down the wrong road, we get lost, we turn back. Maybe it doesn't matter which road we embark on. Maybe what matters is that we embark.
- Barbara Hall -

In this chapter we introduce some preliminary definitions related to the concepts underpinning the results of this thesis. The main architectural concepts associated with workflow technologies are first presented. We then review some technical background related to the Service Oriented Architecture (SOA) paradigm and Web services technologies on which is based the implementation work we pursued in this thesis.

1.1 Workflows

The concept of workflow is defined as follows by the Workflow Management Coalition [Hol95]. "Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal". In this section a formal definition of the notion of workflow is first presented, we then give an overview of the existing architectures to support the execution of workflows namely centralized and decentralized workflow architectures.

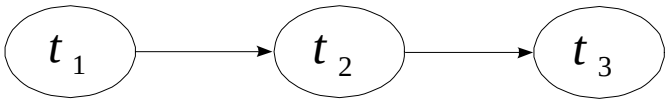

Figure 1.1: SEQUENCE execution pattern

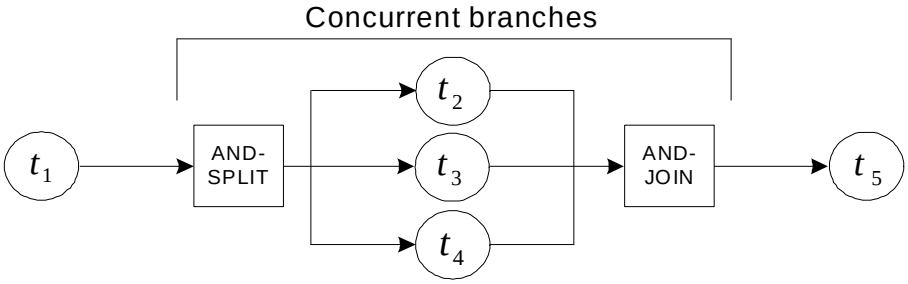

Figure 1.2: AND-SPLIT/AND-JOIN execution patterns

1.1.1 Definition and basic principles

Definition 1.1 (**Workflow**). In its common form a workflow W_f is a finite set defined by:

$$W_f = \{(t_i)_{i\in[1,n]}, \delta\} \text{ where:}$$

- t_i denotes a task executed by a business partner within the workflow,
- δ is the set of execution dependencies between these tasks.

The workflow therefore specifies the sequence of operations that should be performed by a set of business partners in order to achieve the business goal associated with the workflow execution. The set δ specifies the dependencies between workflow tasks in terms of execution patterns. The concepts developed in this thesis rely on most common execution patterns namely:

- **SEQUENCE**: a set of tasks is executed sequentially as depicted in figure 1.1.
- **AND-SPLIT/AND-JOIN**: this pattern corresponds to the execution of concurrent branches as depicted in figure 1.2. The branches split at the AND-SPLIT and merge at the AND-JOIN. The execution right after the AND-JOIN is carried out if only if the execution of all merging branches has been performed.
- **OR-SPLIT/OR-JOIN**: this pattern corresponds to an exclusive choice, a single branch is executed based on a condition associated with the OR-SPLIT as depicted in figure 1.3.

In the course of this thesis, the definition of workflow will be however extended so that it better suits the requirements associated with our architectural design.

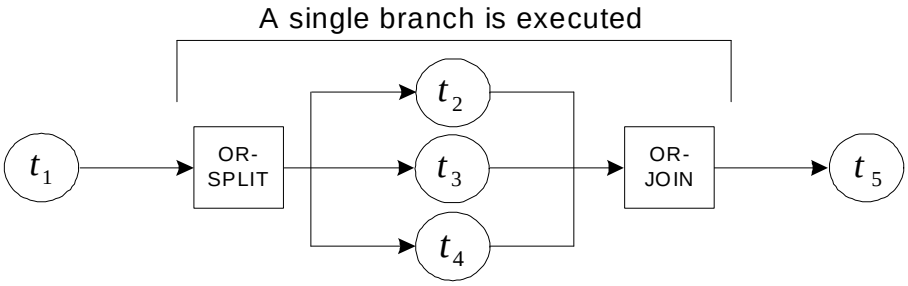

Figure 1.3: OR-SPLIT/OR-JOIN execution patterns

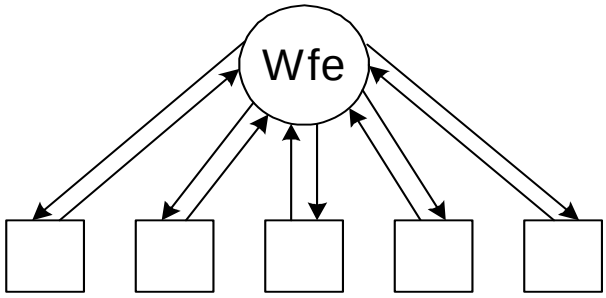

Figure 1.4: Centralized setting

1.1.2 Deployment architectures

There are mainly two types of architectural settings to assure the coordination and management tasks within the execution of workflows: the centralized and the decentralized ones. Both specifications however rely on the same basic component that implements workflow coordination primitives, the workflow engine.

Definition 1.2 (**Workflow engine**)**.** A workflow engine is a piece of software in charge of interpreting the representation of a workflow using a computational form called workflow description language.

Centralized workflow coordination

In the centralized setting, a dedicated coordinator implementing a single workflow engine (Wfe) is in charge of routing messages between business partners based on the execution patterns specified within the workflow, as depicted in figure 1.4.

Decentralized workflow coordination

In the decentralized setting, the management and control tasks are distributed amongst business partners that directly communicate between each others based on the execution patterns speci-

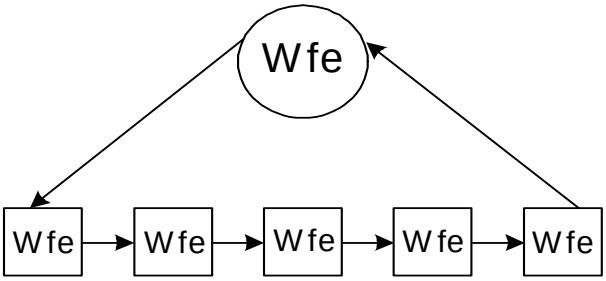

Figure 1.5: Decentralized setting

fied within the workflow as depicted in figure 1.5. In this case, each business partner implements a workflow engine that manages the workflow execution locally.

1.2 Service Oriented Architecture and Web services

Web services are becoming the de facto industry standard in Web enabled middleware architectures that support the execution of collaborative applications. They indeed offer crucial features including: data representation and transport, service description and discovery, service composition and orchestration. In addition, they only rely on simple standards and protocols including XML, SOAP or HTTP. In this section we first give an overview of the Service Oriented Architecture concepts. We then analyse how Web services technologies implement these architectural concepts.

1.2.1 Service Oriented Architecture

This section describes the Service Oriented Architecture concepts on which the Web services technologies rely. The main goal of the Service Oriented Architecture paradigm is to ease the "client-service provider" relationship providing service providers with means to advertise their business functionalities and providing clients with means to search for service providers matching their business needs. The basic concepts of the SOA paradigm are depicted in figure 1.6. SOA features three main actors:

- a **service provider** offers access to some business functionalities.
- a **client** would like to access some business functionalities.
- the **service repository** makes the glue between clients and service providers. Service providers can register their business functionalities while clients can browse the repository listings.

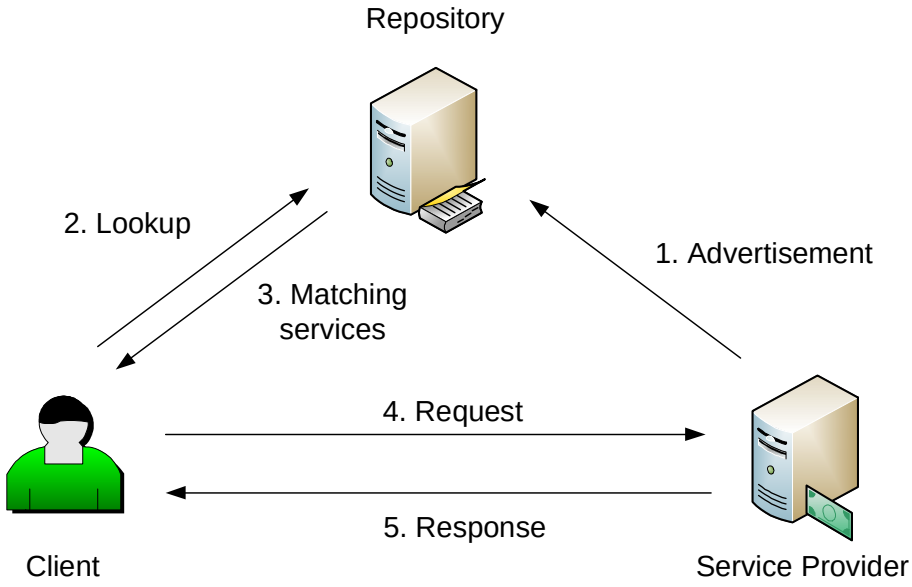

Figure 1.6: Service Oriented Architecture concepts

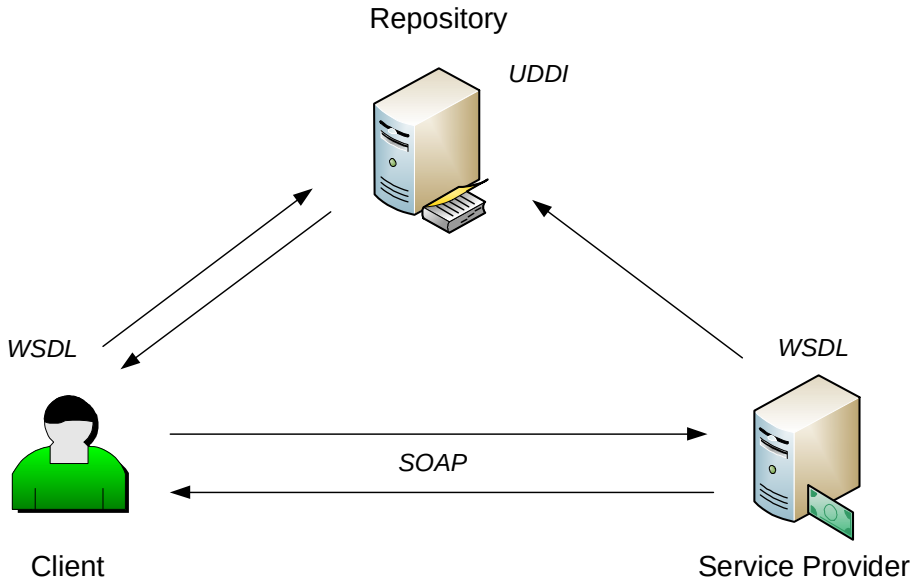

Figure 1.7: Service Oriented Architecture implementation based on Web services technologies

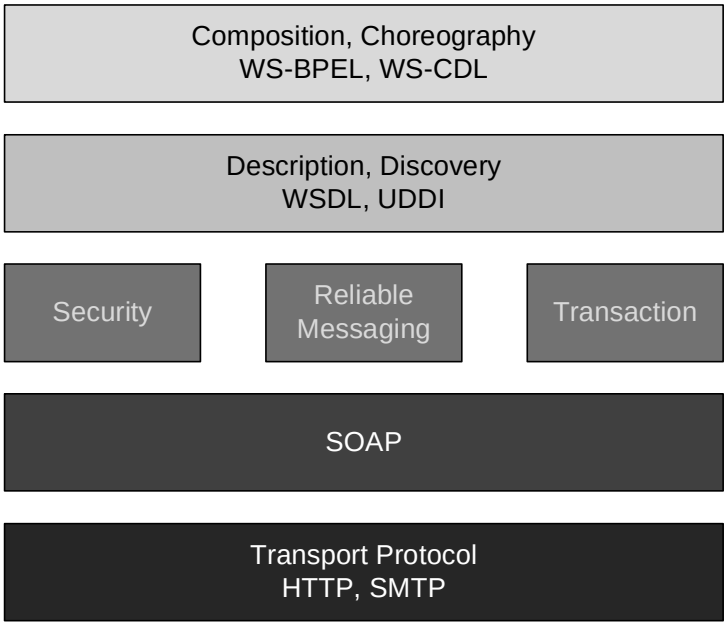

Figure 1.8: Web services stack

Once a client has found a service provider that matches his functional needs, he can directly invoke the operations exposed by the service provider.

Now that we have a clear idea of the SOA basic principles, we present how the latter are implemented using Web services technologies.

1.2.2 Implementation based on Web services technologies

The Web services enabling technologies towards implementing the SOA paradigm are depicted in figure 1.7. Service providers expose their service interface using the WSDL [CCMW01] standard that is an XML-based language [XML] specifying the communication protocols and URIs required to interact with a service provider. WSDL interfaces are stored into service repositories that are generally implemented using the UDDI standard [Ca04]. Clients can browse UDDI repositories in order to retrieve the WDSL interfaces of the service providers that match their business needs. Once clients have retrieved relevant WSDL interfaces, they are able to contact service providers relying on the SOAP protocol [SOA] that is the standard protocol for Web services communications.

1.2.3 Web services stack overview

The Web services stack [WSa] and the associated technologies are depicted in figure 1.8. The Web services stack consists of the following layers.

- **Transport layer**: Web services communications can be supported by various transport protocols including HTTP or SMTP.
- **Messaging**: Web services communications are implemented by means of the SOAP protocol [SOA] that is an XML-based message exchange protocol.
- **Message level specifications**: various Web services specifications have been proposed to enhance the SOAP protocol in order to provide SOAP message exchanges with adequate security mechanisms [WSS] or reliability features [WSR05].
- **Description, Discovery**: the interface of a Web service is specified based on the WSDL specification [CCMW01] that is an XML-based language specifying the syntax to describe the communication protocols supported by a Web service. Web service WSDL interfaces are often published in UDDI [Ca04] registries. The UDDI specification is an XML-based registry wherein service providers can register their available Web services and whose content can be browsed by clients.
- **Composition, Choreography**: Web services can be combined in order to build value-added services. The business logic of so-called composite applications can be specified using the BPEL specification [BPE] that is an XML-based workflow description language. BPEL constructs specify the sequence of operations in terms of Web service invocations required to complete the execution of composite applications. Web services choreography [WSC] is a possible alternative to the BPEL language when it comes to specifying distributed business collaborations as it offers adequate means to describe possible interactions between business participants in a peer-to-peer manner whereas BPEL only offers a centralized view of Web service interactions.

More details on the technologies that will be used later on in this thesis are provided in the next sections.

1.2.4 Web service description

The WSDL specification is used to describe how to interact with a given Web service. Basically it includes the definition of the following parameters:

- **Data structure**: the data types required to interact with a Web service are specified using XML schema [XML04].
- **Operations**: these are the operations offered by a Web service. They are specified in terms of input and output data.
- **Service endpoint reference**: this is the URI of a Web service, i.e. for instance its location on the Internet.

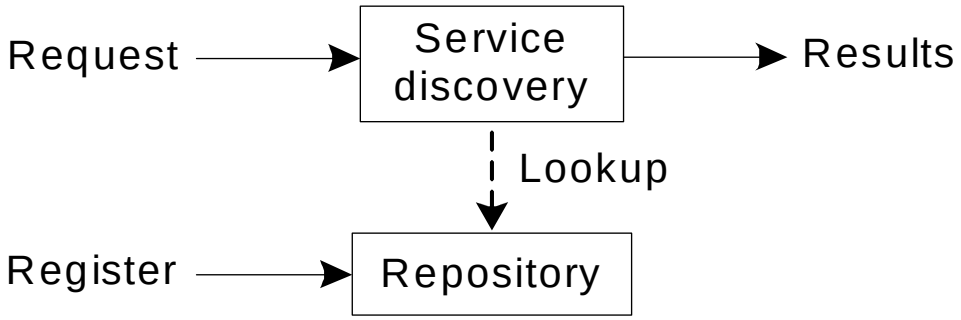

Figure 1.9: Service discovery mechanism

- **Bindings**: message bindings specify the communication protocols that should be used by a client to interact with a Web service.

1.2.5 Web services discovery

The principles of a typical service discovery mechanism are depicted in figure 1.9. A basic service discovery mechanism offers the two following primitives to its clients:

- **Register**: service providers register themselves and advertise their functionalities that are stored in a repository.
- **Request**: clients can send requests to the service discovery mechanism in order to get a list of service providers that match their requirements. In this case, the service discovery mechanism performs a lookup on available repositories by matching incoming requests against the functionalities that business partners advertise and outputs the resulting list of business partners.

There are various types of deployment architectures available to implement a service discovery mechanism ranging from centralized to peer-to-peer solutions.

1.2.6 Web services based workflow applications

The BPEL language has become the industry standard to assure the coordination of Web services composite applications. As we specified in section 1.1 there are two main types of workflow architectures and BPEL processes can be as well deployed based on these two architecture settings. In the centralized setting that is depicted in figure 1.10, the BPEL engine acts as the centralized point of coordination to manage the collaboration of component services. The BPEL engine presents itself as a Web service whose WSDL specification is derived from the BPEL

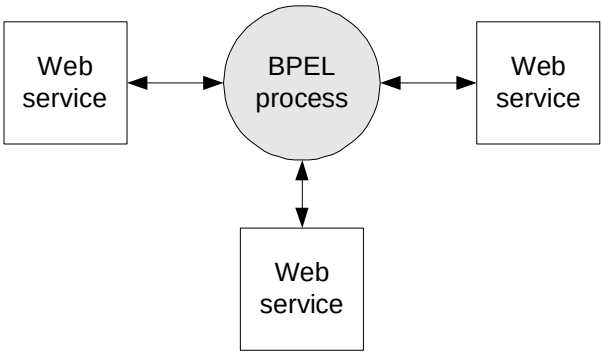

Figure 1.10: BPEL centralized paradigm

process specification. As a result, the BPEL engine is in charge of routing SOAP messages between component services based on the BPEL process specification.

In the decentralized setting, process communications take place between BPEL engines as depicted in figure 1.11. In this case, BPEL engines are also in charge of coordinating the local execution of each business partner involved in the workflow. BPEL processes are locally executed on BPEL engines which expose themselves as Web services. Thus the different BPEL engines involved in the distributed workflow execution see each others as simple Web services and thus communicate by means of SOAP message exchanges.

BPEL processes are strongly linked to the WSDL standard used to specify Web service interfaces. We thus present the role of WSDL within the deployment and execution of BPEL processes before giving details on the BPEL language.

Role of WSDL within BPEL processes

The deployment procedure of a BPEL process includes that of the WSDL interfaces of the Web services involved in the process execution. The latter are indeed used by the underlying BPEL engine to generate the Web service clients to interact with component Web services. In the context of BPEL, WSDL is also used to define communication channels between the BPEL engine and component Web services by means of the `<partnerlinkType>` construct. The latter associates a `<portType>` (a WSDL `<portType>` defines a set of operations) of each service to form a channel composed of two unidirectional links. Each service part of this channel is assigned to a unique role and can send information to the other using the latter's specified `<portType>`.

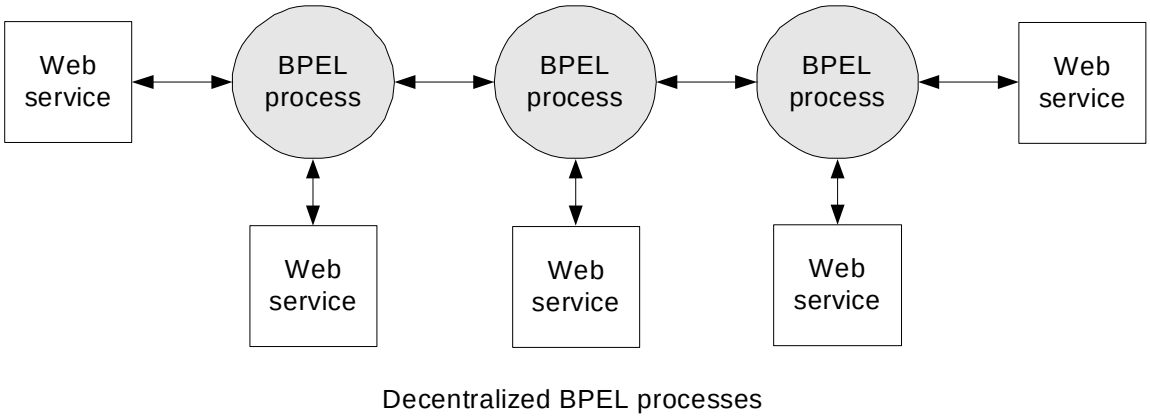

Figure 1.11: BPEL decentralized paradigm

BPEL process specification

BPEL is an XML-based workflow description language that specifies the execution of Web services composite applications in a centralized perspective. As depicted in figures 1.10 and 1.11, Web services part of a BPEL workflow indeed only have interactions with the BPEL engine in charge of managing and interpreting the whole process execution and that even in the decentralized setting whereby BPEL engines consider each others simple Web services. The interactions within the process are described in terms of `<partnerLink>` which associates two Web services in a WSDL `<partnerlinkType>` by assigning the `<partnerlinkType>` roles to them. The BPEL language uses a set of activities to represent the workflow amongst which are the following:

- `<Invoke>` **Web service invocation**: the BPEL engine issues a request (SOAP message) to a Web service. The Web service invocation can either be synchronous (the Web service that has been invoked should reply within a few minutes) or asynchronous (the execution of the operation invoked on the targeted Web service can span over days).
- `<Receive>` **Waiting state**: the BPEL engine waits for a message from a distant Web service. This construct is especially used to instantiate a BPEL process.
- `<Sequence>` **Sequential execution**: this corresponds to the SEQUENCE workflow execution pattern.
- `<Flow>` **Concurrent execution**: this corresponds to the AND-SPLIT / AND-JOIN workflow execution patterns.
- `<Switch>` **Conditional scheme**: this corresponds to the OR-SPLIT / OR-JOIN workflow execution patterns.
- `<Pick>` **Conditional scheme based on external events**: a decision is made based on the content of an incoming message.

- `<While>` **Iterative process**: this corresponds to a loop.
- `<faultHandlers>` **Fault-handling mechanism**: if an error occurs during the execution of an operation, the content specified within the `<faultHandlers>` construct is executed.
- `<eventHandlers>` **Event-handling mechanism**: if the BPEL engine catches a message specified within the `<eventHandlers>` construct, the content specified within the `<eventHandlers>` construct is executed.

1.3 Conclusion

We introduced in this chapter most of the background information used in the remainder of this thesis. We especially introduced the concept of workflow that is the underpinning concept to the work presented in the manuscript and outlined the Service Oriented Architecture paradigm that is the implementation framework of the theoretical results detailed later on.

Chapter 2

Pervasive Workflow Architecture

To create architecture is to put in order. Put what in order? Function and objects.
- Le Corbusier -

2.1 Introduction

Pervasive environments feature complex computer applications that allow users to access ambient services. These services range from residential temperature control to customer assistance in a shopping centre [RM04]. In most existing examples of pervasive services, the end-users only act as clients of complex business processes executed by the ambient infrastructure and none of the existing pervasive service examples allows end-users to share their resources to become themselves service providers. In this thesis, we propose to go beyond this simple client - ambient infrastructure relationship and suggest a distributed computation paradigm wherein users do not act merely as clients but can also share their resources and actively participate in the execution of complex collaborations aiming at building value-added services. Using such a distributed execution environment, complex interoperations between mobile business partners can therefore be envisioned as an extension of the classical workflow concept. The execution of a workflow in a fully decentralized fashion raises of course new issues in terms of workflow control and management, security or reliability. For instance, the dynamic nature of the pervasive environments does not allow permanent assignment of workflow tasks to business partners. Business partners can in fact be assigned to tasks dynamically based on the resources available at runtime. In addition, the compliance of the overall sequence of operations with the pre-defined workflow execution plan is no longer assured without a centralized point of coordination to manage the workflow execution.

As opposed to existing workflow management systems, the distributed execution of workflows in pervasive environments can not rely on a dedicated infrastructure to assure the workflow coordination. There are many issues and requirements relevant to the execution of business processes in the pervasive setting that will be developed in the course of this report. In this chapter however, we focus on the workflow coordination and management tasks and suggest an architecture, denoted pervasive workflow, supporting distributed execution of workflows in pervasive environments. The pervasive workflow architecture features a fully decentralized control and supports dynamic assignment of workflow tasks to business partners so that it meets the following requirements brought up by the pervasive execution environment:

- **Fully decentralized**: the management of the workflow execution is distributed amongst the business partners taking part in a workflow instance in order to cope with the lack of dedicated infrastructure in the pervasive setting,

- **Dynamic assignment of business partners to workflow tasks**: the business partners in charge of executing the workflow can be discovered at runtime based on available resources.

It should be noted that the architecture design proposed in this thesis is not only suited for the execution of workflows in the pervasive setting but also offers adequate support for the execution of cross-organizational workflows over traditional mediums including the Internet.

The remainder of this chapter is organized as follows. In section 2.2 the architecture is defined in terms of data structures associated with the workflow execution plan and dynamic information, a protocol for the exchange of the workflow data amongst business partners and a new function called role discovery dealing with the dynamic assignment of workflow tasks to business partners. Section 2.3 depicts a detailed application of the conceptual model that we implemented based on the Web services technologies and an extension of the workflow description language BPEL. Section 2.4 discusses the related work and section 2.5 gives some concluding remarks.

2.2 Pervasive workflow architecture

The challenges raised by pervasive execution of workflows mainly result from the lack of a dedicated infrastructure to perform workflow management tasks. The main objective of our architecture is to cope with these challenges through distributed mechanisms. We first introduce some preliminary definitions and present the requirements underpinning the design of the pervasive workflow architecture. The architecture is then outlined in terms of the runtime and cross-organizational specifications. The architecture lifecycle is finally described.

2.2.1 Problem statement and definitions

We first introduce the requirements underpinning the pervasive workflow architecture design before defining the pervasive workflow model.

Requirements

The pervasive nature of the execution environment is characterized by the two following assumptions mostly derived from the lack of a dedicated infrastructure to manage the workflow execution:

- **Distributed control**: no single business partner is in charge of managing the workflow execution and the overall control is assured through the collaboration of the business partners involved in the workflow execution,
- **Dynamic task assignment**: the business partners that will be taking part in the instance of a given workflow are not known in advance, they can be selected at runtime.

These two assumptions basically define the requirements our architecture design will have to meet.

Our second assumption raises the need of a mechanism enabling the discovery of business partners executing a workflow instance. In this chapter, we focus on the workflow execution support, the main features of this service discovery mechanism are therefore detailed but its specification is out of the scope of this work. The solution specified in [TPR06] featuring a distributed service discovery mechanism can be used to implement this functionality. We define two modes within the business partner selection process:

- **Source discovery**: all the business partners are assigned to tasks before the workflow instantiation. This is a requirement whenever the execution of a workflow has specific requirements such as reliability requirements as detailed later on in chapter 3.
- **Runtime discovery**: business partners are selected along with the workflow execution, this is the default mode.

Workflow model

A pervasive workflow W is represented using a directed graph. In the graphical workflow model, each vertex represents all workflow actions contiguously performed by a business partner, whereas the edges represent the sequence of workflow steps among business partners. We distinguish the following elements associated with the graph definition:

- t_{ni}: task activities consisting of actions performed by the business partners involved in the workflow. Each task activity is locally executed by a given business partner and a vertex v_i represents the set of task activities locally executed by the same business partner. As a result, each vertex is linked to a single business partner and two neighbor vertices in the graph are linked to two different business partners. Each task activity is identified by an index n and the index i of the vertex where it is executed.

- $M_{i \to j}$: workflow messages used to transfer control and data between business partners. These are the graph edges and represent the message exchanged between the business partners linked to two consecutive vertices during the execution of a workflow instance. i and j are the indices of the vertices linked by this edge. We note $(M_{i \to j_p})_{p \in [1,z_i]}$ the set of workflow messages issued by the business partner assigned to v_i to the z_i partners assigned to the vertices $(v_{j_p})_{p \in [1,z_i]}$ executed right after v_i.

- d: process control activities specifying execution patterns or dependencies between workflow tasks. The set of dependencies between tasks within a workflow is denoted δ. In the scope of this thesis, we consider three types of dependencies or workflow patterns as they are defined in [Hol95]:

  - Sequential execution: abstract interpretation of the directed graph,
  - AND-SPLIT/AND-JOIN: branches executed concurrently,
  - OR-SPLIT/OR-JOIN: branches based on a conditional choice.

W is therefore a finite set of v_i, $M_{i \to j}$ and d. In addition, we also consider the notion of role derived from usual workflows. A role R_i is defined as a set of attributes required to execute the task activities associated with a vertex v_i. As we stated before, we assume that the execution of a workflow instance is not *a priori* assigned to any business partners, the latter can be discovered at runtime. Our workflow model is therefore an abstract one. We consider that a role R_i consists of two categories of attributes:

- **Functional requirements**: these are the functionalities a candidate business partner needs to implement in order to be able to execute the vertex v_i.

- **Non functional requirements**: these are the other requirements specified for the execution of the vertex v_i including for instance, security credentials or transactional characteristics as developed later on in this thesis.

In fact, the notion of role is associated with the service discovery mechanism that is indeed in charge of assigning business partners to vertices through a match-making procedure in such a way that the functional and non functional properties offered by selected business partners meet the requirements defined for vertices.

We can for instance model a simple process with two branches of sequential vertices using the graph depicted in Figure 2.1. The business partner assigned to the vertex v_1 sends the

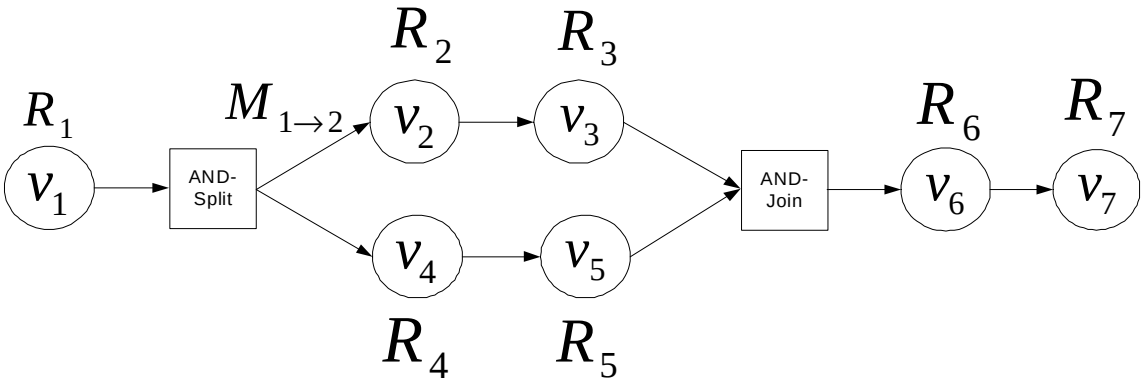

Figure 2.1: Process with two branches

workflow messages $M_{1\to 2}$ and $M_{1\to 4}$ to the business partners assigned to the vertices v_2 and v_4, respectively. Two branches are then executed concurrently and the business partner assigned to the vertex v_6 gathers the data resulting from the execution of the concurrent branches. The business partners involved in that workflow instance of course satisfy the set of roles specified for the workflow. In this thesis, we only consider simple workflow scenarios with no synchronization between concurrent branches.

The notations we introduced in this section are summarized in the following definition.

Definition 2.1 (**Pervasive Workflow**). A pervasive workflow W is a finite set defined by:

$$W = \{(v_i)_{i\in[1,n]}, (R_i)_{i\in J}, (M_{i\to j})_{i,j\in K}, \delta\} \text{ where:}$$

- v_i denotes a vertex which is a set of workflow tasks that are performed by a business partner from the receipt of workflow data till the transfer of data to the next partner,
- δ is the set of execution dependencies between those vertices,
- $(R_i)_{i\in J}$ is the set of roles defining for each workflow vertex the attributes required for the vertex execution. J is a subset of $[1, n]$,
- $(M_{i\to j})_{(i,j)\in K}$ is the set of workflow messages linking two consecutive vertices in the workflow. K is a subset of $[1, n]^2$.

The instance of the workflow W wherein a set of $k < n$ business partners $(b_i)_{i\in[1,k]}$ are assigned to the vertices $(v_i)_{i\in[1,n]}$ is denoted W_d.

Definition 2.2 (**Adjacency matrix of a pervasive workflow**). The adjacency matrix of a pervasive workflow denotes the $n \times n$ matrix $A := (a_{i,j})_{(i,j)\in[1,n]^2}$ defined by:

$$a_{i,j} : \begin{cases} 1 \text{ if } M_{i\to j} \in (M_{i\to j})_{(i,j)\in K} \\ 0 \text{ otherwise} \end{cases}$$

2.2.2 Runtime specifications

The runtime specifications of our architecture are detailed in this section. We first specify the components required to support the distributed execution of pervasive workflows. The appropriate messaging protocols ensuring the proper deployment and execution of the workflow within the pervasive workflow management system are then examined.

A distributed workflow management system

The design of a distributed workflow management system that does not rely on any dedicated workflow management system requires that the workflow management task is distributed amongst all involved business partners. As a result, as we mentioned in chapter 1, a part of the workflow management infrastructure should be deployed on each business partner. A workflow engine is therefore pre-installed on business partners that wish to take part in the execution of a pervasive workflow. The role of local workflow engines consists in assuring the management of the workflow execution on business partners' devices. With respect to the workflow model introduced in the previous section, it basically consists in the following sequence of operations:

- Receipt of all incoming workflow messages,
- Execution of the required task activities associated with the vertex to which the business partner has been assigned,
- Assignment of business partners to the next vertices to be executed within the workflow,
- Sending workflow messages to the next business partners.

Overall, the complete runtime specification of the pervasive workflow architecture is depicted in figure 2.2. Having designed an abstract representation of the workflow whereby business partners are not yet assigned to functional tasks, the workflow initiator launches the execution. He executes a first vertex (1) before discovering in its surrounding environment a business partner able to perform the next set of workflow tasks (2). Once the discovery phase is complete, workflow data are transferred from the business partner that performed the discovery to the discovered one (3) and the workflow execution further proceeds with the processing of a set of tasks (4). The sequence composed of the discovery procedure, the transfer of data and the execution of a set of tasks is iterated till the final set of tasks.

Due to availability issues introduced within the pervasive setting, we choose by default a stateless execution model. A single vertex can only be linked to a business partner at a time, meaning that after the execution of a given vertex, no residual information are stored on a business partner's device, they are all transferred to the next one. In the previous section, we defined the pervasive workflow W which specifies the complete workflow execution pattern. At

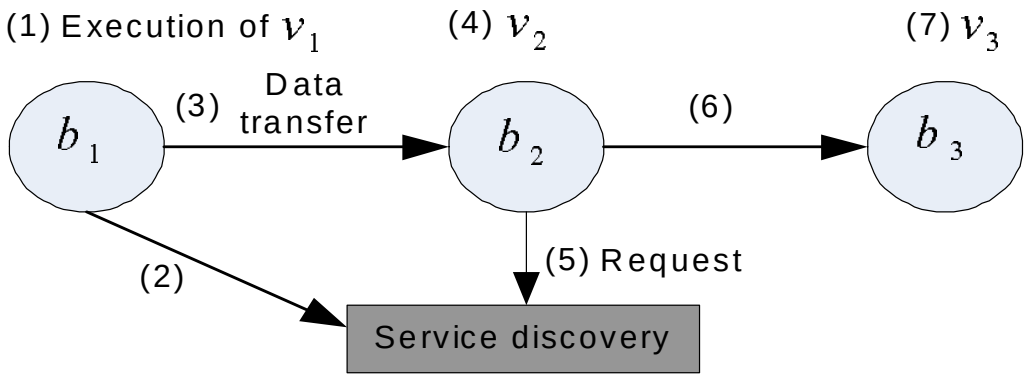

Figure 2.2: Pervasive workflow runtime specification

the local workflow engine level, we need to specify the subset W_i of W locally executed by the business partner b_i assigned to the vertex v_i. The local workflow W_i of a vertex v_i consists of the following set of operations:

- Receipt of the workflow messages that the business partner should wait for before starting the vertex execution,
- The task activities t_{ni} associated with the execution of v_i,
- Sending outgoing workflow messages upon completion of the vertex execution,
- The set of process control activities d describing the dependencies between the operations required to complete the vertex execution.

The complete execution of W thus corresponds to the execution of the different local workflows $(W_i)_{[1,n]}$ by the local workflow engines deployed on the devices of the business partners involved in the workflow instance. We need now to specify the messaging protocols used in the communications between the different workflow engines to allow a coherent execution of the set of local workflows $(W_i)_{[1,n]}$ with respect to the complete workflow W.

Messaging protocols within the architecture

The communications between the business partners are managed at the workflow engine level based on the complete workflow while the management task of the local workflows is performed by workflow engines as depicted in figure 2.4. The message exchanges between the engines refers to the workflow messages introduced in the previous section. Each workflow message should include the data required to enable a coherent workflow execution. The main objective of the message exchanges within the architecture is first to allow business partners which may not have any a priori knowledge about a running instance of a workflow to be part of it and

Figure 2.3: Workflow message format

execute what it is requested upon their assignment to a vertex. Business partners should then be able to forward operation requests to the appropriate recipients which are next in the workflow execution. The data carried by a workflow message should therefore enable the following set of operations:

- Identify the vertex v_i of the workflow instance that the business partner is assigned to,
- Retrieve the local workflow W_i associated with the considered vertex to actually know what to execute,
- Identify the workflow instance to which the received message belongs,
- Retrieve the identity of the business partners already assigned to a vertex when source discovery is used,
- Retrieve workflow data to process them during the execution of v_i.

In order to meet these functional requirements, the workflow message format is defined as depicted in figure 2.3 and includes the following information:

- W: Complete workflow definition, which is a representation of the directed graph defined in 2.2.1,
- W_{iid}: Workflow instance identifier, unique string to identify an instance of a workflow. The instance identifier can be built for instance using some random numbers and the workflow initiator identity,
- i: Workflow vertex number i that the message recipient has to execute,
- Map: Mapping table between vertices and business partners' URIs. This table is updated with respect to the discovery mode used; it can either be complete at the workflow initialization phase with source discovery or updated at each workflow vertex with runtime discovery,
- $Data$: Workflow data to be processed by business partners.

The workflow message format defined this way meets the functional requirements we identified:

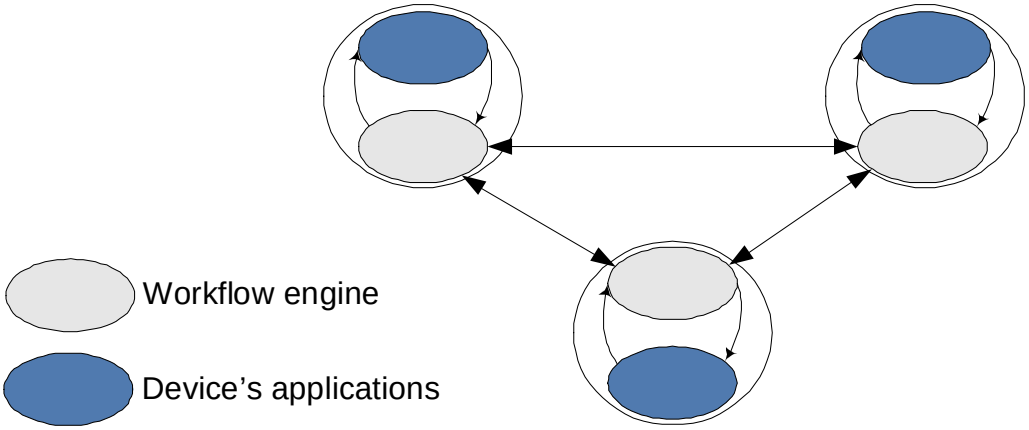

Figure 2.4: Distributed workflow management system

- The vertex index i is included in the message, the vertex v_i can thus be retrieved,
- W_i can be derived from W with the knowledge of i,
- The workflow instance is identified with W_{iid},
- The association vertices to business partners is stored in Map,
- Workflow data are included in the message and can be processed.

2.2.3 Cross-organizational aspects

The business partners involved in a workflow instance are an extension of the organization they belong to. They take part in the workflow execution with respect to their available applications and are also in charge of managing their participation to the workflow instance. We thus consider that the business partners' devices consist of two parts:

- A set of mobile applications available on the device,
- An embedded workflow engine.

The mobile applications of each device may be required to be kept private, they should therefore only be invoked by the device itself. The workflow engine thus acts as a proxy between these applications and the other devices of the collaboration. In this section, we detail how the access to mobile applications through the workflow engine is performed.

Device representation

To cope with the requirements introduced by cross-organizational collaborative workflows, we adopt the representation of "private-public" processes introduced in [SO01]. As a result, the embedded workflow engine mainly assumes the two following roles:

- **Management of the device internal applications**: the local workflow engine invokes appropriate applications based on functional requirements defined for the vertex to which the business partner is assigned.
- **Coordinating the execution of** W_i: W_i is deployed on the local workflow engine.

Thus, two processes corresponding to these roles are deployed on local workflow engines:

- **A public process**: this is the subset W_i of W specified in section 2.2.2. This process definition includes the activities the business partner is asked to perform to complete the execution of the vertex v_i to which he is assigned.
- **A private process**: this is an internal one developed by the business partner or his organization based on the applications available on his device. The internal process of the business partner b_i assigned to the vertex v_i is noted IP_i and it is not considered to be distributed but coordinated in a centralized manner by the local workflow engine.

We consider in fact a hierarchical model with two levels to represent the task activities the business partners will have to perform during a pervasive workflow instance. The first level corresponds to the public definition of the workflow provided by the global process definition W and the subsets W_i derived from it. At this level, the specification of task activities are known by all involved business partners.

These public task activities can themselves be complex processes involving many execution steps. The atomic definitions of these public tasks are specified within the private processes implemented by business partners; this constitutes the second level. The atomic definition of a given task activity is thus only known by the business partner in charge of executing this specific task within the workflow. The link between these two levels is done by means of message exchanges between the public and private processes deployed on a business partner's local workflow engine.

Private-public processes link specification

The internal process IP_i is itself a workflow and thus consists of a set of the following elements:

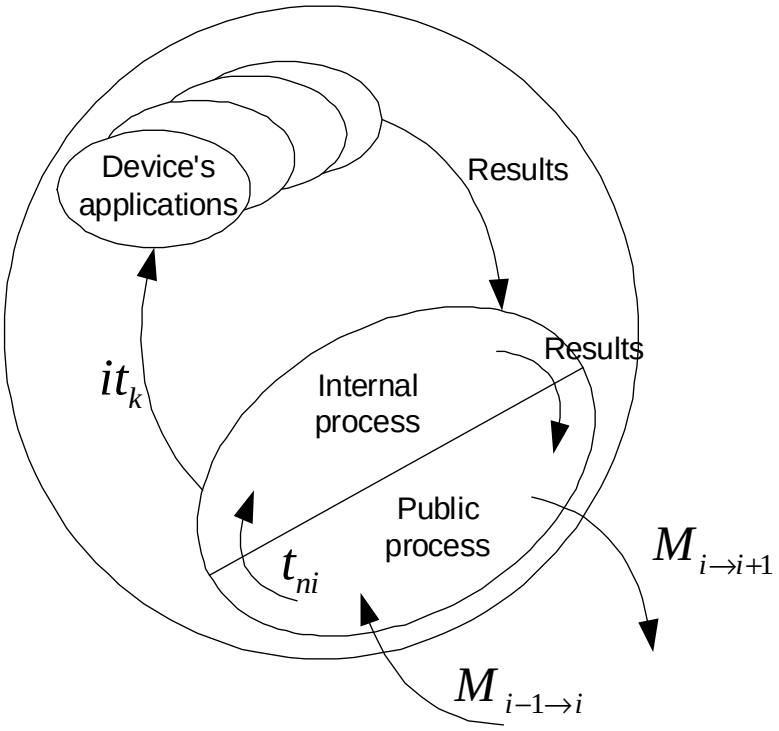

Figure 2.5: Device representation

- Internal task activities it_k specifying the atomic tasks performed by the device,

- Process control activities d specifying the execution schemes of the internal task activities.

This process is centralized, and thus workflow messages are in this case not defined. The internal process of a business partner implements all the task activities that his device is able to perform and a public task activity t_{ni} can be associated with a subset T_n of the internal process composed of internal task activities and dependencies. The internal process implemented by a business partner thus consists of different subsets T_n corresponding to the public task activities t_{ni} that can be performed by the business partner's device.

The internal communications resulting from this description are depicted in figure 2.5. Whenever a business partner's device is asked to perform a task activity t_{ni} upon receipt of a set of workflow messages, the public process forwards the request to the internal process which executes the subset T_n corresponding to t_{ni}. The local applications required to achieve that subset are thus invoked. The results are transferred to the other business partners by the public process which issues a set of workflow messages.

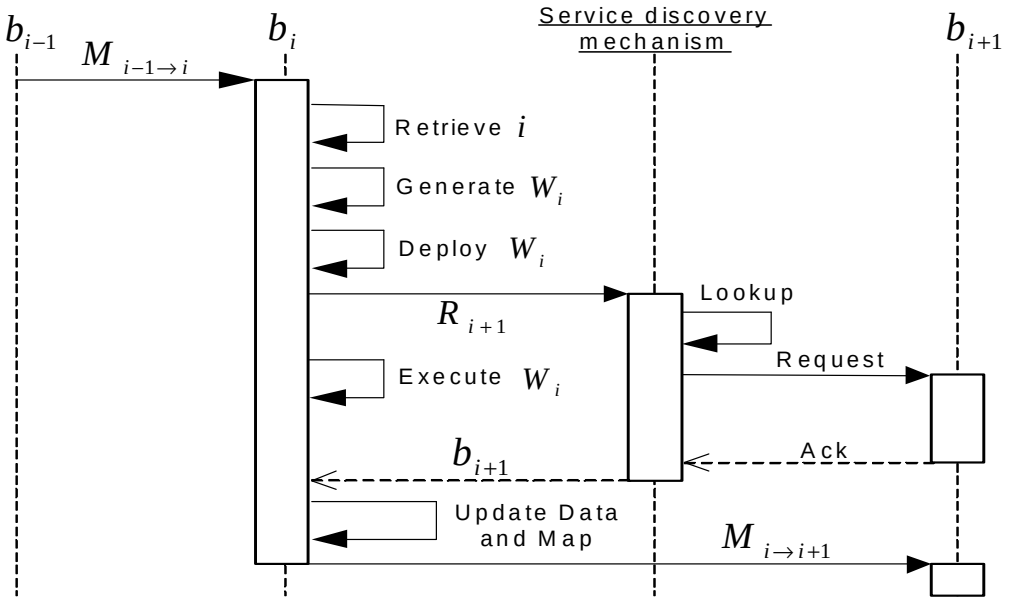

Figure 2.6: Architecture sequence diagram

2.2.4 Complete architecture mechanisms

We detailed so far the behavioral characteristics of the elements part of our pervasive workflow architecture. In this section, we present how these elements work together and the basic mechanisms of this architecture. Figure 2.6 presents a sequence diagram depicting the interactions between three business partners b_{i-1}, b_i and b_{i+1} in charge of executing the vertices v_{i-1}, v_i and v_{i+1} and the ambient service discovery mechanism. We assume a runtime discovery scenario. Upon receipt of a workflow message $M_{i-1 \to i}$, the business partner b_i first retrieves the index i and generates the workflow W_i associated with the vertex to which he is assigned. The process W_i is then dynamically deployed on the business partner's local workflow engine and instantiated. In the meantime, the business partner makes a request to the service discovery mechanism in order to select a business partner that matches the requirements set for the vertex v_{i+1}. Once the execution of W_i is complete and the business partner b_{i+1} is assigned to v_{i+1}, Data and Map fields are updated, and b_i sends the workflow message $M_{i \to i+1}$ to b_{i+1}.

There are two specific issues relevant to the AND-SPLIT, AND-JOIN patterns that we did not mention so far. First, the business partner assigned to a vertex into which concurrent branches (AND-JOIN) merge is discovered by the business partner assigned to the vertex during the execution of which these concurrent branches split (AND-SPLIT). Second, we consider that when concurrent branches merge the update of data that have been concurrently modified during the execution of these branches is an issue that is handled by the applications implemented by business partners.

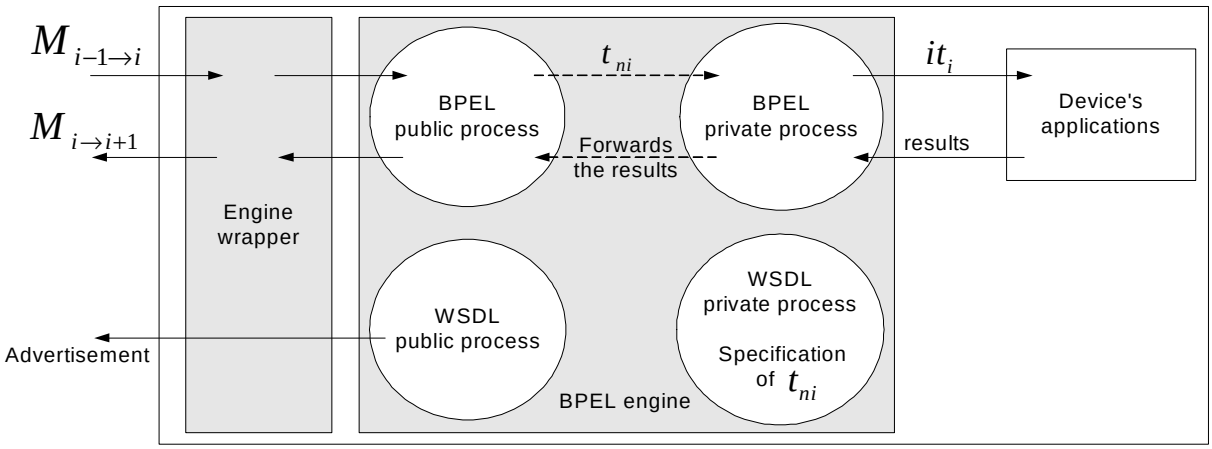

Figure 2.7: Web services infrastructure deployed on business partners' devices

2.3 Web services application

Section 2.2 presented the characteristics of the pervasive workflow architecture we designed. We now present its implementation based on the Web services technologies whose composition properties are appropriate for distributed workflow applications. Web services technologies can make use of BPEL [BPE] as workflow description language and WSDL [CCMW01] standard to enable the composition of services within workflows. We first give an overview of the infrastructure components we implemented to enable the execution of pervasive workflows in a Web services environment. We then analyze the deployment of the architectural mechanisms which include the specification of the pervasive workflow W, cross-organizational aspects and data management. Finally, some performance aspects are discussed.

2.3.1 Infrastructure components

In the architecture definition we analyzed the requirements on the workflow management system to enable the execution of our pervasive workflow model. Our conclusion was the deployment on each device of workflow engines. When using Web services technologies, these workflow engines are BPEL enabled and communicate via Web service interfaces. Each device executing a vertex of a workflow instance will have to support this infrastructure. Figure 2.7 presents the infrastructure components that business partners willing to be involved in pervasive workflow instances have to implement. The implementation is composed of the following modules:

- **Engine wrapper**: this module implements the pervasive workflow intelligence and is in charge of the following execution primitives:
  - Workflow message processing,

  - Public process generation and deployment,
  - Business partner discovery,
  - BPEL engine invocation.

  The engine wrapper is contacted by other business partners involved in a workflow with workflow messages and acts as the business partner's device interface. As such it advertises the business partner's available task activities.

- **BPEL workflow engine**: as specified in the architecture design section, two BPEL processes are deployed on the BPEL workflow engine. The BPEL public process W_i is deployed at runtime and specifies the task activities that a business partner has to execute to complete the execution of a vertex. The BPEL public process is instantiated by the engine wrapper. The BPEL private process IP_i is implemented by the business partner and specifies the sequence of invocations required to perform the task activities implemented by a business partner. The BPEL private process is instantiated by the BPEL public process.

- **Device's applications**: the set of applications available on a business partner's device. They are contacted by the BPEL private process only.

The communications within this infrastructure including workflow messages, process instantiations and application invocations are asynchronous so that workflows that span over hours or days can be supported. The workflow instance identifier is in this case used to identify the workflow instance to which asynchronous messages belong. W_{iid} is especially used within BPEL processes to initiate correlation sets.

In addition to the infrastructure that business partners should support, the pervasive workflow model relies on a service discovery mechanism that enables the assignment of business partners to workflow vertices. There are many service discovery systems implemented using Web services technologies [Ca04, DGA04, LH03, Ba05, SLM04] that can be either centralized or distributed. In our implementation, we worked with two of them, WS-Discovery [Ba05, TPR06] featuring a distributed solution based on multicast communications and a centralized one based on the Web service ontology OWL-S [OWL03] (section 3.9 provides details on this work). The WS-Discovery specification implements a simple string based match-making procedure between functionalities advertised by service providers and requests issued by clients. In the OWL-S solution the match-making procedure is more complex and details about semantic match-making can be found in [DGA04, LH03].

The integration of either solutions into the pervasive workflow infrastructure, consists in the specification of roles associated with workflow vertices and that of the advertisement of task activities. In the case of WS-Discovery, this is simply done by means of strings whereas in the case of OWL-S, OWL-S profiles are used. Both solutions however requires that business partners that may be involved in a pervasive workflow instance share a common semantics or ontology to express functional requirements and advertise their functionalities. If the service discovery mechanism finds a set of business partners that matches the requirements specified

by clients, it outputs the WSDL interfaces exposed by the selected partners so that they can be later on contacted.

Now that we have a clear idea of the components part of the Web services infrastructure, we need to specify the deployment of the runtime mechanisms presented in the architecture design. This includes first the computational representation of W to enable a business partner to retrieve W_i expressed in BPEL so that he can execute a given vertex v_i.

2.3.2 Specification of *W*

The specification of W is a crucial point in the infrastructure implementation since it is used by all involved business partners to determine what to execute. We want to specify pervasive workflows using the characteristics of the BPEL language so that BPEL private processes W_i can be easily derived from W. The solution we designed features a monolithic workflow file specifying pervasive workflows based on an extension BPEL denoted distBPEL. When a business partner is assigned to a vertex v_i, the BPEL public process W_i is extracted from this monolithic specification and the process is dynamically deployed on the local workflow engine of the business partner. In this section we thus describe how a pervasive workflow W is specified before detailing the deployment process of a BPEL public process W_i derived from W.

distBPEL pervasive workflow representation

The main concepts underpinning the extension of BPEL we define are first presented; the distBPEL language is then specified. Our goal is to use BPEL constructs to represent a distributed workflow W specified using a directed graph. We first translate the task activities, process control activities and workflow messages that are defined in section 2.2 into standard BPEL activities:

- t_{ni}: task activities are associated with `<invoke>` activities that correspond to the invocation of the BPEL private process implemented by a business partner.
- $M_{i \rightarrow j}$: workflow messages are also `<invoke>` activities, the device of the business partner assigned to the vertex v_i invokes the device of the business partner assigned to the next vertex. This is a Web service call.
- δ: the set of dependencies between operations. These are `<flow>`, `<sequence>`, `<pick>` and `<switch>` BPEL constructs.

We use the basic concepts of the BPEL language and map them to the directed graph approach presented in section 2.2 in order to define a workflow description language capable to

```
<sequence>
  <t11>
  <flow>
    <sequence>
      <M12>
      <t12>
      <M23>
      <t13>
      <M36>
    </sequence>
    <sequence>
      <M14>
      <t14>
      <M45>
      <t15>
      <M56>
    </sequence>
  </flow>
  <t16>
  <M67>
  <t17>
</sequence>
```

Table 2.1: Listing distBPEL associated with the workflow depicted in figure 2.1

```
<invoke type="WorkflowMessage" source="i" recipient="j"/>
```

Table 2.2: Listing distBPEL associated with a workflow message

completely specify pervasive workflows W. We call this representation distBPEL standing for distributed BPEL. The goal of this extension is therefore to represent distributed processes from a global perspective whereas BPEL is currently limited to depict processes from a centralized perspective. BPEL control activities are used to represent graph dependencies. For instance, the `<flow>` construct represents AND-SPLIT, AND-JOIN workflow patterns, `<sequence>` construct represents sequential execution and `<switch>` construct represents OR-SPLIT, OR-JOIN workflow patterns. Figure 2.1 can therefore be represented by the XML listing depicted in table 2.1. We assume that each vertex of this process specifies a single task activity.

The task activity of v_1 is first executed; a concurrent execution of two sequence branches is then performed.

In order to provide a full XML representation of a distBPEL workflow, we now specify the representation of workflow messages and task activities in distBPEL. Workflow messages and task activities are standard BPEL `<invoke>` activities as they correspond to Web service calls. A workflow message $M_{i \to j}$ has the form depicted in table 2.2. We simply specify the source and recipient vertices of the workflow message.

```
<invoke partnerlink="internal" portType="internalportType"
        operation="signature" inputVariable="DataIn"
        outputVariable="DataOut" vertex="i"/>
```

Table 2.3: Listing distBPEL associated with a task activity

We define a specific XML schema for the workflow message invocation as it is only relevant to the engine wrapper component of our infrastructure and will not be part of the BPEL public process dynamically deployed on the BPEL engine. As we indeed explained above, the processing of workflow messages is done by the engine wrapper. The specification of a task activity t_{ni} is an abstract one based on the common semantics shared between business partners and we adopt the representation depicted in table 2.3 in case, for example, of a signature operation. The vertex attribute defines the vertex to which the task activity belongs. This representation mainly translates the idea of a task executed internally since the `partnerLink` and `portType` associated with the operation signature are not known by other business partners.

Example of process representation in distBPEL

We provide a complex example of a distBPEL workflow specification based on the workflow depicted in figure 2.8.

Example 2.1. We consider a process with concurrent execution of two branches between activities and dependencies between the branches. The workflow proposes two parallel branches of activities executed in sequence, the process control activities `<sequence>` and `<flow>` are thus used. This example is particular due to the workflow message $M_{2\rightarrow 5}$ which links two vertices from different branches. To have a coherent and consistent process definition, this workflow message is specified twice in the distBPEL process definition. The computational representation of the directed graph has indeed to take into account all dependencies between workflow messages and task activities. In that specific case, $M_{2\rightarrow 5}$ and $M_{2\rightarrow 3}$ are sent concurrently by the business partner assigned to the vertex v_2 while it also has to be received by the business partner assigned to the vertex v_5 before executing the task activity t_{15}. This workflow message appears as a result twice in the process representation. The distBPEL listing of the workflow is depicted in table 2.4.

Deployment of the public process W_i

As we stated in our architecture, the execution of a vertex v_i described in W is linked to its local description W_i. Based on the distBPEL workflow representation we depicted in the last section, we now present an extraction procedure enabling the retrieval of the public business process W_i associated with a vertex v_i with the only knowledge of the vertex index i. In our infrastructure, the deployment of W_i includes the retrieval of information that are relevant to the engine wrapper and the deployment of the BPEL public process. The deployment of the

```
<sequence>
  <invoke partnerlink="internal" portType="internalportType"
          operation="11" inputVariable="DataIn1"
          outputVariable="DataOut1" vertex="1"/>
<flow>
<sequence>
  <invoke type="WorkflowMessage" source="1" recipient="2"/>
  <invoke partnerlink="internal" portType="internalportType"
          operation="12" inputVariable="DataOut1"
          outputVariable="DataOut2" vertex="2"/>
<flow>
  <invoke type="WorkflowMessage" source="2" recipient="3"/>
  <invoke type="WorkflowMessage" source="2" recipient="5"/>
</flow>
  <invoke partnerlink="internal" portType="internalportType"
          operation="13" inputVariable="DataOut2"
          outputVariable="DataOut3" vertex="3"/>
  <invoke type="WorkflowMessage" source="3" recipient="6"/>
</sequence>
<sequence>
  <invoke type="WorkflowMessage" source="1" recipient="4"/>
  <invoke partnerlink="internal" portType="internalportType"
          operation="14" inputVariable="DataOut1"
          outputVariable="DataOut4" vertex="4"/>
<flow>
  <invoke type="WorkflowMessage" source="4" recipient="5"/>
  <invoke type="WorkflowMessage" source="2" recipient="5"/>
</flow>
  <invoke partnerlink="internal" portType="internalportType"
          operation="15" inputVariable="DataOut4"
          outputVariable="DataOut5" vertex="5"/>
  <invoke type="WorkflowMessage" source="5" recipient="6"/>
</sequence>
</flow>
  <invoke partnerlink="internal" portType="internalportType"
          operation="16" inputVariable="DataOut5"
          outputVariable="DataOut6" vertex="6"/>
</sequence>
```

Table 2.4: Listing distBPEL associated with the workflow depicted in figure 2.8

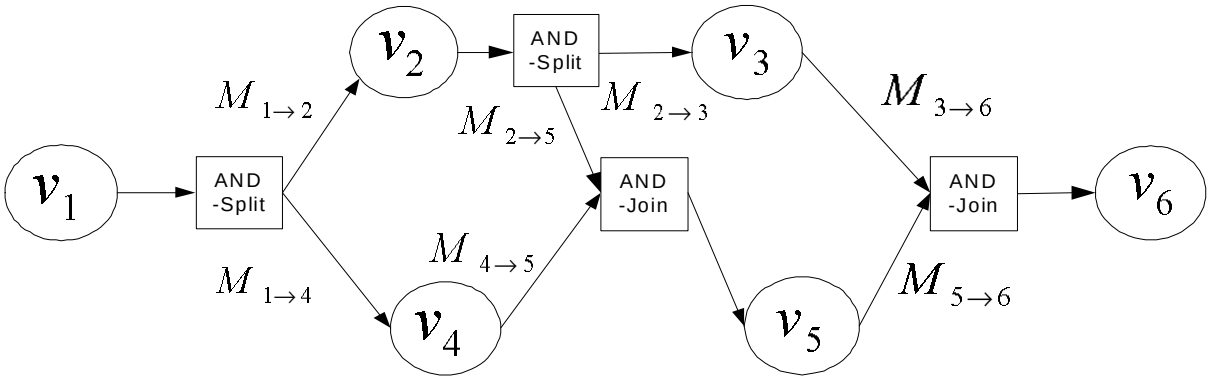

Figure 2.8: Complex business process

BPEL public process requires both WSDL and BPEL specifications. The WSDL definitions to be deployed are the ones from the three following modules: the BPEL public process, the BPEL private processes and the engine wrapper. These WSDL definitions can be easily retrieved and we thus focus first on the BPEL public process extraction procedure. The procedure towards the extraction of W_i consists of three steps and is illustrated considering the vertex v_5 in the workflow depicted in Figure 2.8.

Step 1: Focus on the considered vertex The input of this procedure is the global process description specified in section 2.3.2. We want to derive from this input the local vision of vertex v_5 and only consider the incoming and outgoing workflow messages as well as the task activities that have to be performed for vertex v_5. The first step of our procedure is therefore to focus on those specific activities and delete the activities in which vertex v_5 is not involved. The other aspect is also to keep the dependencies specified by the process control activities. According to our procedure, $M_{2\to 5}$, $M_{4\to 5}$, t_{15}, $M_{5\to 6}$ are only kept as depicted in table 2.5.

Step 2: Simplification of redundant workflow messages. As we stated in the description of distBPEL, all dependencies between tasks need to be specified in a distBPEL process description. As a result, in case of dependencies between concurrent branches, tasks creating these dependencies are specified twice. When focusing on a single vertex, we only keep the instance of a redundant workflow message that creates dependencies on the task activities or workflow messages concerning the vertex. In our example, the second instance of $M_{2\to 5}$ introduces dependencies (sequential dependency) with the other tasks of vertex v_5. The first instance is thus erased as depicted in table 2.6.

Step 3: Simplification of process control activities. So far, we restricted the global process description to the tasks involving a given vertex. This created inconsistencies with the process control activities. A process control activity indeed specifies an execution scheme between at least two workflow messages or task activities. Process control activities which do not verify this property are deleted from the specification. To do so, we perform an iterative procedure till all process control activities properly specify an execution scheme of at least two activities and we get the listing depicted in figure 2.7. There is

```
<sequence>
<flow>
<sequence>
<flow>
  <invoke type="WorkflowMessage" source="2" recipient="5"/>
</flow>
</sequence>
<sequence>
<flow>
  <invoke type="WorkflowMessage" source="4" recipient="5"/>
  <invoke type="WorkflowMessage" source="2" recipient="5"/>
</flow>
  <invoke partnerlink="internal" portType="internalportType"
          operation="15" inputVariable="DataOut4"
          outputVariable="DataOut5" vertex="5"/>
  <invoke type="WorkflowMessage" source="5" recipient="6"/>
</sequence>
</flow>
</sequence>
```

Table 2.5: Listing distBPEL after the first step of the extraction procedure

```
<sequence>
<flow>
<sequence>
<flow>
</flow>
</sequence>
<sequence>
<flow>
  <invoke type="WorkflowMessage" source="4" recipient="5"/>
  <invoke type="WorkflowMessage" source="2" recipient="5"/>
</flow>
  <invoke partnerlink="internal" portType="internalportType"
          operation="15" inputVariable="DataOut4"
          outputVariable="DataOut5" vertex="5"/>
  <invoke type="WorkflowMessage" source="5" recipient="6"/>
</sequence>
</flow>
</sequence>
```

Table 2.6: Listing distBPEL after the second step of the extraction procedure

```
<sequence>
<flow>
  <invoke type="WorkflowMessage" source="4" recipient="5"/>
  <invoke type="WorkflowMessage" source="2" recipient="5"/>
</flow>
  <invoke partnerlink="internal" portType="internalportType"
          operation="15" inputVariable="DataOut4"
          outputVariable="DataOut5" vertex="5"/>
  <invoke type="WorkflowMessage" source="5" recipient="6"/>
</flow>
</sequence>
```

Table 2.7: Listing distBPEL after the third step of the extraction procedure

no example depicting the procedure associated with the OR-SPLIT pattern but of course in this case once the condition is evaluated there is no need to keep the branch of the workflow that won't be executed in the public process, it is however kept in the pervasive workflow W in order to keep track of the process execution.

The three-step procedure we just presented outputs a distBPEL workflow including incoming and outgoing workflow messages and task activities. In our implementation we have chosen not to modify an existing BPEL engine to make it compliant with our distBPEL representation but rather to add an engine wrapper to implement the pervasive workflow intelligence. The distBPEL representation that is obtained can not be directly deployed on the BPEL engine and some of the information that it contains are used by the engine wrapper while some others are used to derive the BPEL public process that will be deployed on the BPEL engine. In the listing provided in table 2.7, the process W_i would be interpreted as follows.

1. Wait for two workflow messages sent by the business partners assigned to v_2 and v_4,
2. Execute the operation "15",
3. Send a workflow message to the business partner assigned to v_6.

Step 1 and 3 are only relevant to the engine wrapper that is in charge of workflow message processing while step 2 is used in order to build the BPEL public process deployed on the BPEL engine. The generation of the BPEL public process based on the set of tasks that the business partner is required to perform is only an implementation issue, we thus only describe the global structure of a standard BPEL public process based on the example depicted in figure 2.9. The BPEL public process is composed of five parts:

- **Process instantiation**: data that should be processed during the execution of the vertex are transferred by the engine wrapper to the public process. This is an asynchronous Web service call.

- **Variable assignments**: data are prepared to match the input formatting of the task activities that will process them.

- **Task activity execution**: the task activities implemented by the BPEL private process are invoked based on the specification defined in W_i. These are asynchronous Web service calls.

- **Result gathering**: modified data are sent back by the BPEL private process and variable assignments take place to match the data formatting expected by the engine wrapper.

- **Engine wrapper callback**: modified data are transferred to the engine wrapper for processing.

The complete deployment of the BPEL public process includes that of WSDL definitions. In order to ease the deployment process, generic WSDL definitions can be defined using arrays for the input and output data of the BPEL public process, the WSDL can otherwise be generated based on the generated BPEL public process. The WSDL definitions of the engine wrapper and the BPEL private process can be retrieved since these modules are not dynamically deployed.

Actual implementation

The actual implementation work we pursued does not implement the complete distBPEL specification we presented. In order to ease the workflow specification process since we do not have a distBPEL workflow modeler available, the pervasive workflow specification we implemented consists of two pieces of data:

- **Control flow**: This is the adjacency matrix associated with the workflow graph. This basically specifies the execution patterns between workflow vertices.

- **Vertex public workflows**: BPEL code snippets are associated with workflow vertices. These pieces of BPEL code are almost the complete specifications of vertex public processes.

This workflow is processed as follows. The business partner analyses the matrix to determine on the one hand the workflow messages he should wait for before starting the vertex execution and on the other hand the workflow messages he will have to issue once the vertex execution is complete. The business partner then retrieves the BPEL code snippet associated with the vertex he is assigned to.

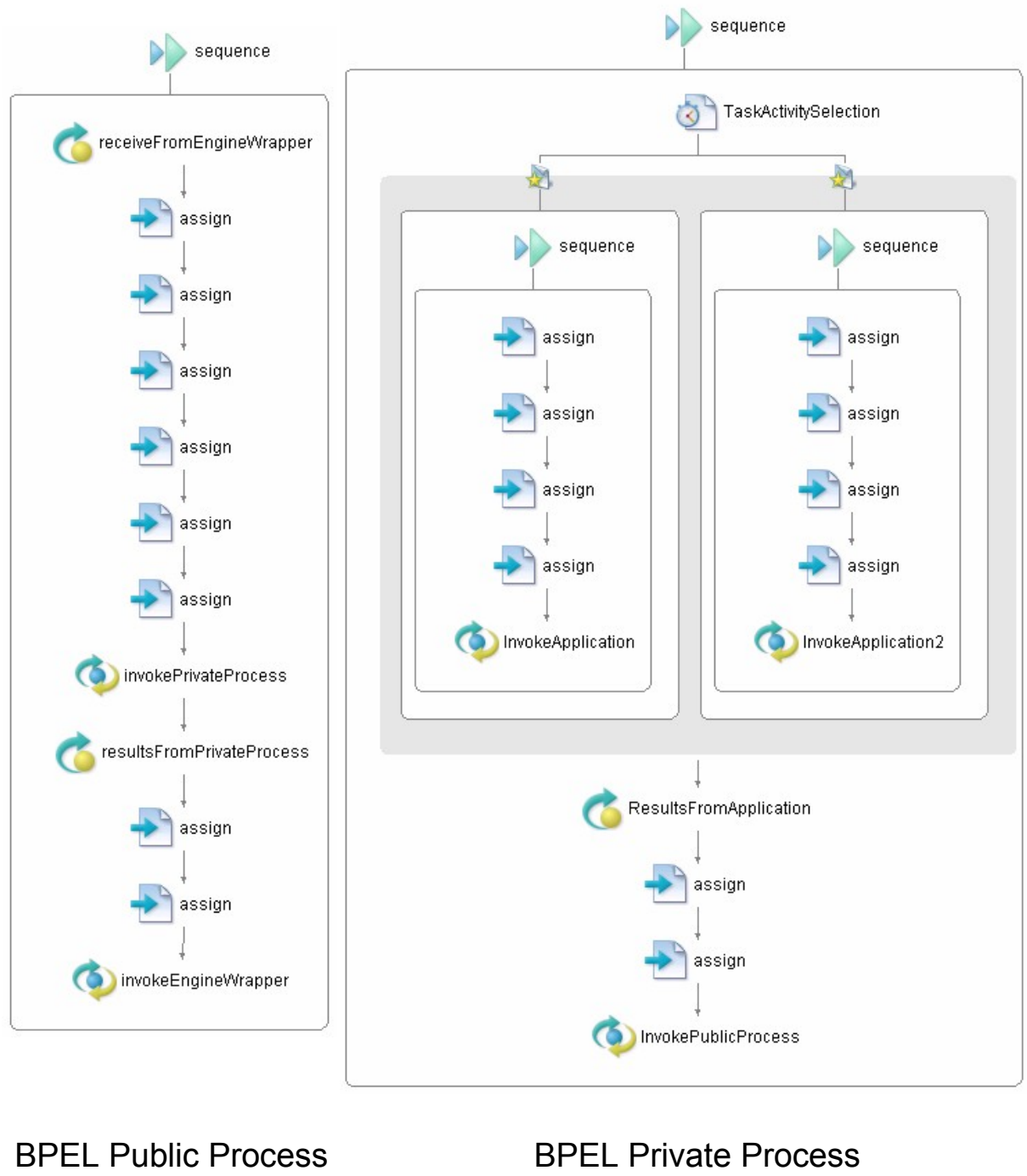

Figure 2.9: Private and public processes (Process graphs from ActiveBPEL engine [AEwe07])

```
<pick createInstance="yes" name="TaskActivitySelection">
 <onMessage operation="signature192" partnerLink="privatepublicPLT"
            portType="privatePT" variable="Data">
  <sequence>
   <invoke inputVariable="document" operation="signature192"
           partnerLink="privatePLT" portType="signPT">
    <correlations>
     <correlation initiate="yes" pattern="out" set="CS1"/>
    </correlations>
   </invoke>
   ...
  </sequence>
 </onMessage>
 <onMessage operation="encryption192" partnerLink="privatepublicPLT"
            portType="privatePT" variable="Data">
  <sequence>
   <invoke inputVariable="data" operation="encryption192"
           partnerLink="privatePLT" portType="encryptPT">
    <correlations>
     <correlation initiate="yes" pattern="out" set="CS1"/>
    </correlations>
   </invoke>
   ...
  </sequence>
 </onMessage>
</pick>
```

Table 2.8: Listing BPEL associated with the Internal Process specification

2.3.3 Internal process specification in BPEL

We present in this section a template for the internal process specification. As part of our architecture each business partner has an internal process deployed on his device that manages the invocation of its embedded applications. This process is defined by the business partner himself and describes the operations his device is able to perform. It is only invoked inside the device by the public process. Hence, the internal process definition is organized based on this role and divided into subsets T_n as defined in section 2.2.3. The process control function `<pick>` of BPEL that implements an IF statement based on external events is used to implement this division into subsets. In that case the external events are indeed the incoming invocations associated with task activities that are performed by the public process in order to execute specific operations. The BPEL code is thus organized in different sections delimited by `<onMessage>` tags. Each section corresponds to a specific subset T_n and is activated based on incoming requests as depicted in table 2.8. Besides, the BPEL code associated with the gathering process of the results that are sent back by the device's applications can be common to the different subsets as depicted in figure 2.9.

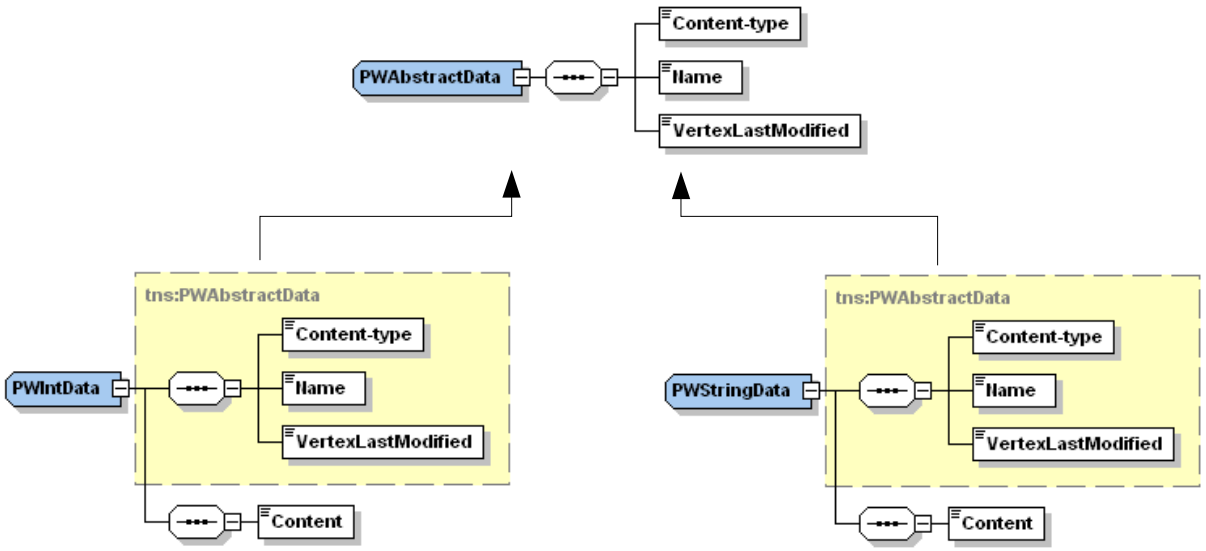

Figure 2.10: Data structure

2.3.4 Data management

The management of data is a crucial aspect of a pervasive workflow instance. There are indeed many different data types ranging from pictures to text documents that can be exchanged and modified during the execution of a workflow and that should be transparent for the underlying workflow management system. The data structure defined to meet this requirement is first specified, we then precise the workflow data lifecycle within the pervasive workflow infrastructure.

Data structure

The data structure we defined is presented in figure 2.10. An abstract data type is defined so that the workflow infrastructure does not have to implement specific operations to handle different data types. In the fashion of MIME attachments [NWG96], data types are identified by means of the "content-type" field associated with the file extension of the data content.

Data lifecycle

Figure 2.11 depicts how workflow data are handled locally by business partners. The engine wrapper receives workflow data in a serialized form [ASF06] and forwards the ones that should be processed by the device's applications to the BPEL engine in the same serialized form so that the process execution does not depend on any data type. Upon receipt of data, device's applications deserialize data based on the the type specified in the "content-type" field, update data contents and serialize them again before sending back these serialized data to the private

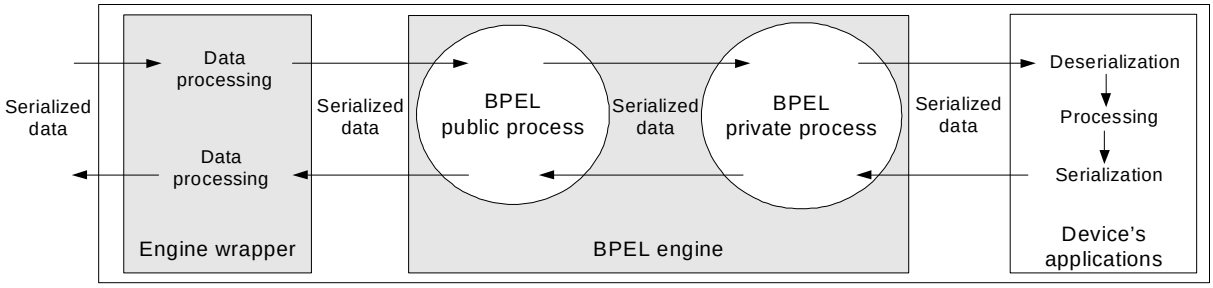

Figure 2.11: Workflow data lifecycle

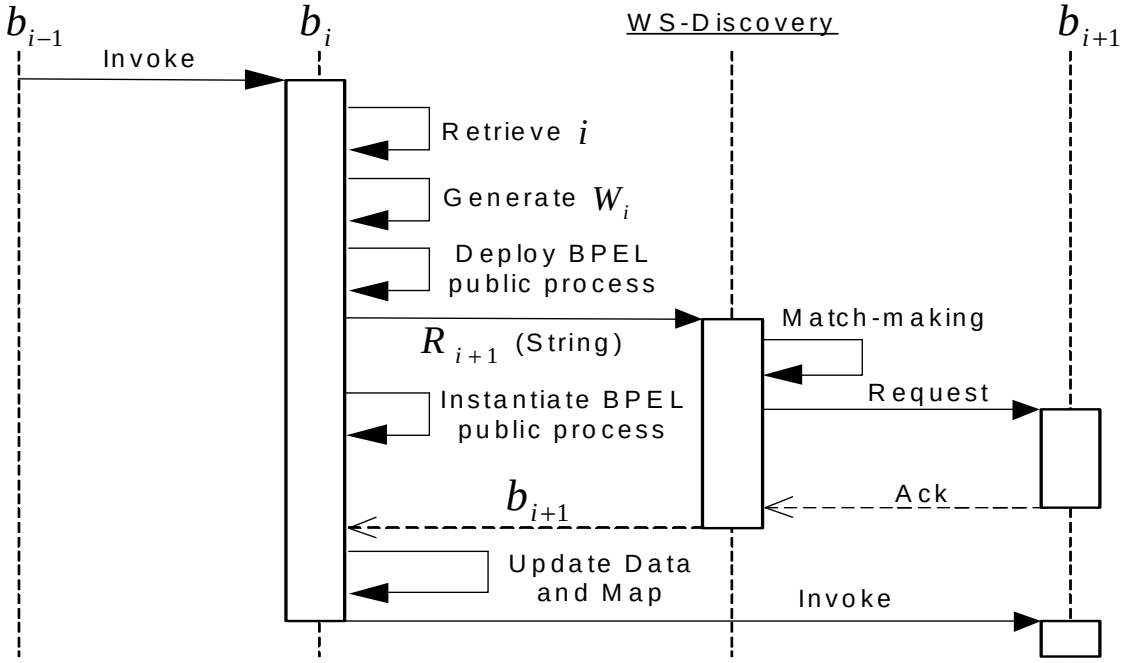

Figure 2.12: Infrastructure sequence diagram

business process.

2.3.5 Execution scheme of a distributed workflow in the infrastructure

In section 2.2.4 we have provided a sequence diagram of the conceptual architecture, we detail in this section the mapping of the latter with the concepts we have just presented. Figure 2.12 depicts the sequence diagram associated with the infrastructure we implemented:

- Workflow messages correspond to Web service asynchronous calls,
- The generation of the public process W_i is performed with the procedure of section 2.3.2,

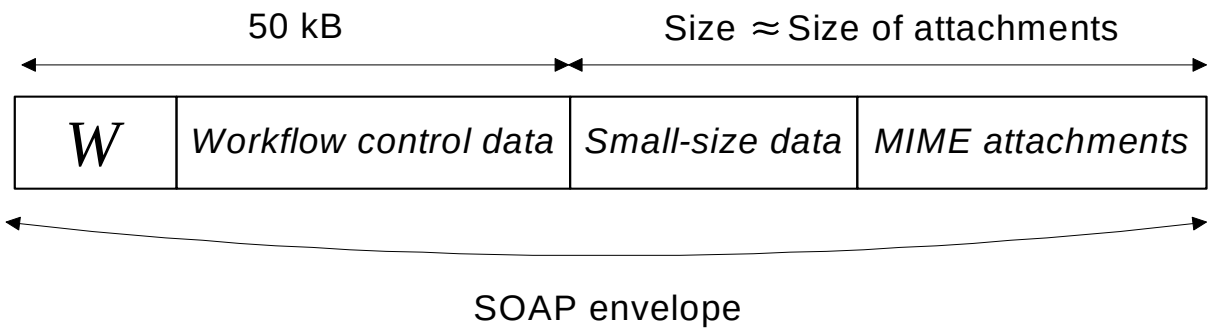

Figure 2.13: Workflow message size

- The discovery procedure is based on a match-making procedure between the OWL-S profile or a string specifying the functional and non functional requirements associated with a vertex and the characteristics advertised by candidate business partners.

Additional details on the engine wrapper design and implementation can be found in appendix A.

2.3.6 Performance considerations

We detail in this section some elements that are relevant to the performances of the pervasive workflow architecture. The size of workflow messages is first evaluated, the pros and cons of a decentralized workflow model vis a vis a centralized one are then discussed.

Workflow message size

Workflow message size is in fact equivalent to that of an email when the SOAP with Attachments [Ba00] specification is implemented to transfer large pieces of data such as pictures or text documents. Large-size data are indeed handled as MIME attachment [NWG96] within the SOAP envelope of workflow messages and it is not required in this case to serialize them using UTF-8 encoding [UTF03] which would lead to a message size explosion. The workflow message implementation is depicted in figure 2.13.

Centralized vs decentralized

Figure 2.14 depicts the execution of a workflow in both centralized and decentralized settings. Less messages are of course required to complete the workflow execution in the decentralized setting than in the centralized one without a centralized point of coordination. The amount of

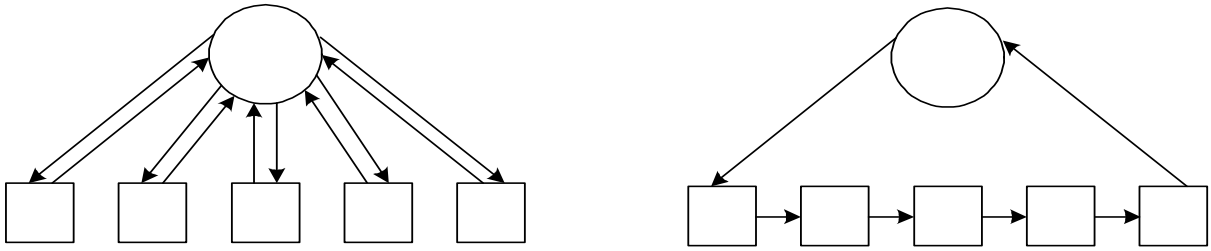

Figure 2.14: Centralized vs decentralized workflow systems

data transferred between partners depends on the workflow specification and no general conclusion can be drawn about this parameter.

The centralized coordinator is in fact a performance bottleneck especially if this dedicated infrastructure is in charge of managing concurrent workflow instances. Therefore this execution setting does not seem to offer sufficient flexibility to support the execution of state of the art business processes. The lack of a dedicated infrastructure comes however at the expense of security and reliability, topics that are developed in the next chapters.

2.4 Related work

The distributed execution of workflows has been long ago foreseen [AM97, AAAM97] as the enabler of complex collaborations crossing organizational boundaries in order to deliver value-added applications to business partners. With the pervasive workflow architecture, we propose to go one step beyond by making it possible to execute cross-organizational workflow-based applications in environments whose ambient infrastructure may not offer the appropriate support of execution. The areas of research tackled by the design work presented in this chapter include:

- Execution of workflows in the pervasive setting and more generally in environments that do not offer any dedicated infrastructure,
- Web services composition,
- Distributed execution of workflows.

This section is organized based on these three topics.

2.4.1 Decentralized workflow architectures

Several pieces of work provide an architecture for distributed execution of workflows. These architectures are almost all based on the design of a communication protocol that is used to transfer workflow data between business partners. They however all suffer from a static design and a lack of flexibility in the business partner selection process which is a critical issue towards meeting the requirements of pervasive environments.

In [BMR96, BMR94] the concept of information carrier (INCA) is introduced which is somehow equivalent to the notion of workflow message we introduced and makes the design of INCA close to that of the pervasive workflow architecture. The INCA system is however not integrated with any service discovery mechanism which makes it not suited for dynamic collaborative applications. The distributed architectures specified in [AAA+95, GAC+97] focus as well on the distributed execution of workflows between identified peers rather than enabling the execution of dynamic collaborations.

In [NCS03] a decentralized orchestration scheme for Web services is proposed. Starting from a centralized BPEL process specification, a distributed one is produced and deployed on already designated Web services. The solution presented in [MWW+98] proposes a similar approach using state chart based workflow specifications. Self-Serv [BDS05] implements a distributed orchestration platform for composite Web services. These solutions propose distributed composition of Web services chosen at the process designing phase. Our solution on the contrary introduces an extension of the BPEL language interpreting distributed processes to allow a runtime discovery of involved business partners.

Agent based solutions [CR04, VBS04, Wan00] are also well adapted to support the execution of distributed workflows. Mobile agents travel between sites implementing the tasks required to complete workflow instances. As opposed to the pervasive workflow architecture, they however rely on pieces of mobile code transferred between business partners and thus require stringent security solutions to protect both the workflow management system and the business partners against malicious software.

Finally, the primary goal of the decentralized workflow architecture described in [SWSS03] is to ensure scalability of workflow executions by means of load balancing mechanisms. The concepts developed in this work could be well applied to the pervasive workflow architecture to improve the business partner selection process when concurrent instances of a same workflow are executed.

2.4.2 Execution of workflows in the pervasive setting

Most of the research in the area of collaborative applications in pervasive environments concerns the interaction between end-users and an ambient infrastructure. In [RM04] a workflow-based

architecture to help mobile users of unfamiliar pervasive environments is proposed. Human interactions within BPEL processes are studied in [CL04] wherein an architecture adapting to workflow users' mobility is introduced. These architectures still consider workflow end-users with a low level of implication within the workflow execution managed by the ambient infrastructure. In comparison, we propose a self-managing architecture enabling a completely distributed workflow execution wherein business partners that can of course be mobile workers can be dynamically selected.

More complex collaborative applications have been recently studied [SGS+04, DSBW+06] to offer the possibility to dedicated workflow systems to dispatch subsets of workflow instances to end-users in a distributed fashion. These pieces of work however implement a stateful execution model that is less flexible than our pervasive workflow architecture and thus still requires the support of a dedicated infrastructure.

2.4.3 Web services composition

Despite quite extensive work [RS04, MM04] the semantic composition of Web services falls in one of two approaches: assignment of Web services instances at process design time [ADK+05] or at runtime [MM03].

Our architecture leverages the runtime approach and the results specified in [MM03] can be used to increase the flexibility of the pervasive workflow model in the business partner selection process. The integration of this work in our workflow management system would have to be pursued within a centralized service discovery mechanism offering semantic functionalities.

2.5 Conclusion

We presented in this chapter an architecture and its implementation based on Web services technologies to support the execution of workflows in pervasive environments. Our solution meets the requirements raised in the pervasive setting namely distributed control to cope with the lack of dedicated workflow management infrastructure and dynamic task assignment to take into account the dynamic nature of pervasive environments wherein available computational resources constantly change. The implementation of the pervasive workflow architecture we have pursued within the Service Oriented Computing paradigm identifies the required adaptations so that existing Web services technologies such as the BPEL workflow description language can be easily integrated to meet the constraints of our architectural design.

We believe that thanks to the fully decentralized control that is the underpinnings of the execution model, the pervasive workflow architecture will foster the development of new business cases suited to pervasive environments. An actual pervasive workflow example in the eHealth-

care field is in fact presented in appendix D as a proof of concept. This prototype that we implemented features a collaborative business application between doctors, pharmacists, social workers and hospitals towards providing medical care to a patient whose health condition is remotely monitored.

Lack of centralized control to perform workflow control and management tasks on the other hand raises serious concerns about security and fault recovery. These specific issues are studied in the next chapters in order to build a secure and reliable workflow management system.

Chapter 3

Consistency of Pervasive Workflows

The transaction concept has emerged as the key structuring technique for distributed data and distributed computations
- Jim Gray -

3.1 Introduction

In chapter 2 we presented the pervasive workflow architecture that we designed in order to support the execution of worfklows in environments that do not offer any dedicated infrastructure. Featuring a dynamic assignment of tasks to workflow partners, the pervasive workflow architecture allows users to initiate workflows in any environment where surrounding users' resources can be advertised by various means including a service discovery mechanism. Yet, this architecture does not provide any guarantee on the consistency of the outcome reached by the process execution. Considering the lack of reliability akin to distributed environments, assuring data and transaction consistency of the outcome of workflow instances supported by the pervasive workflow infrastructure is necessary. The requirements that are relevant to assuring consistency of the execution of processes on top of the pervasive workflow infrastructure are mainly twofold:

- **Relaxed atomicity**: atomicity of the workflow execution can be relaxed as intermediate results produced by the workflow may be kept despite the failure of one partner. The specification process of transactional requirements associated with workflows has to be flexible enough to support coordination scenarios more complex than the coordination

rule "all or nothing" specified for the two phase commit protocol [2PC].

- **Dynamic assignment of business partners**: the workflow execution is dynamic in that the workflow partners offering different characteristics can be assigned to tasks depending on the resources available at runtime. Business partners' characteristics have thus to be combined or composed in a way such that the transactional requirements specified for the workflow are met.

Existing transactional protocols [Elm92, GFJK03] are not adapted to meet these two requirements as they do not offer enough flexibility to cope for instance with the runtime assignment of computational tasks. In addition, existing solutions to combine or compose business partners based on the characteristics they offer appear to be limited when it comes to integrating at the composition phase the consistency requirements defined by workflow designers. These solutions indeed only offer means to validate transactional requirements once the workflow business partners have been selected but no solution to integrate these requirements as part of the workflow partner selection process.

In this chapter, we propose an adaptive transactional protocol for the pervasive workflow architecture. The execution of this protocol takes place in two phases. First, business partners are assigned to tasks using an algorithm whereby workflow partners are selected based on functional and transactional requirements. Given an abstract representation of a process wherein business partners are not yet assigned to workflow tasks, this algorithm enables the selection of partners not only according to functional requirements but also to transactional ones. In our approach, these transactional requirements are defined at the workflow design stage using the Acceptable Termination States (*ATS*) model. The resulting workflow instance is compliant with the defined consistency requirements and its execution can be easily coordinated as our algorithm also provides coordination rules. The workflow execution further proceeds through a hierarchical coordination protocol managed by the workflow initiator and controlled using the coordination rules computed as an outcome of the partner assignment procedure. Besides, it should be noted that the practical solutions that are presented in this chapter do not only answer specific requirements introduced by the pervasive workflow model but are sufficiently generic to be applied to other workflow architectures supporting long-running transactions.

The remainder of the chapter is organized as follows. Section 3.2 introduces preliminary definitions and the methodology underpinning our approach. We present an example of pervasive workflow execution in section 3.3 for the purpose of illustrating our results throughout the chapter. Section 3.4 introduces a detailed description of the transactional model used to represent the characteristics offered by business partners. In section 3.5, we provide details on the termination states of a workflow then section 3.6 describes how transactional requirements expressed by means of the *ATS* model are derived from the inherent properties of termination states. Section 3.7 and section 3.8 present the transaction-aware business partner assignment procedure and the associated coordination protocol, respectively. An implementation of our theoretical results based on Web services technologies including OWL-S [OWL03] and BPEL [BPE] is presented in section 3.9. Section 3.10 discusses related work while section 3.11 presents concluding remarks.

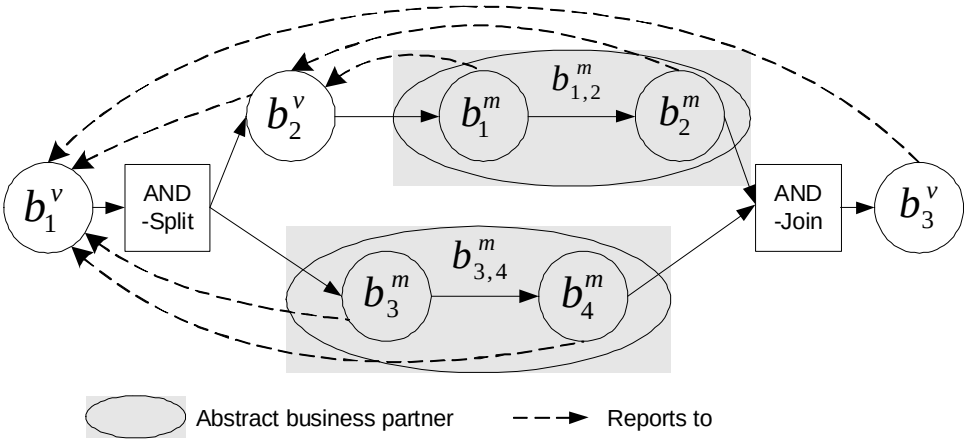

Figure 3.1: Protocol actors

3.2 Definitions and Problem statement

Defining a transactional protocol for pervasive workflows raises challenges that are mainly due to the flexibility of their execution and their lack of dedicated infrastructure in charge of management and control tasks. In this section, we specify the set of requirements in terms of transactional consistency that must be met by the execution of pervasive workflows before outlining our methodology towards meeting these requirements.

3.2.1 Assuring consistency of pervasive workflows

As a first step towards assuring workflow consistency one has to be able to express transactional requirements as part of the workflow model. We therefore want to offer the possibility to coordinate some tasks of a pervasive workflow instance in order to assure the consistency of termination states. Our approach consists in partitioning the specification of a pervasive workflow into subsets or zones and identifying some zones called critical zones wherein transactional requirements defined by designers have to be fulfilled.

Definition 3.1 (Critical zone). We define a critical zone C of a workflow W as a subset of W composed of contiguous vertices which require to meet some transactional requirements. We distinguish within C:

- $(m_k)_{k\in[1,i]}$ the i vertices whose tasks only modify mobile or volatile data,
- $(v_k)_{k\in[1,j]}$ the j vertices whose tasks modify data other than mobile ones, v_1 being the first vertex of C.

The business partner assigned to the vertex v_k (resp. m_k) is noted b_k^v (resp. b_k^m) and the instance of C is noted C_d.

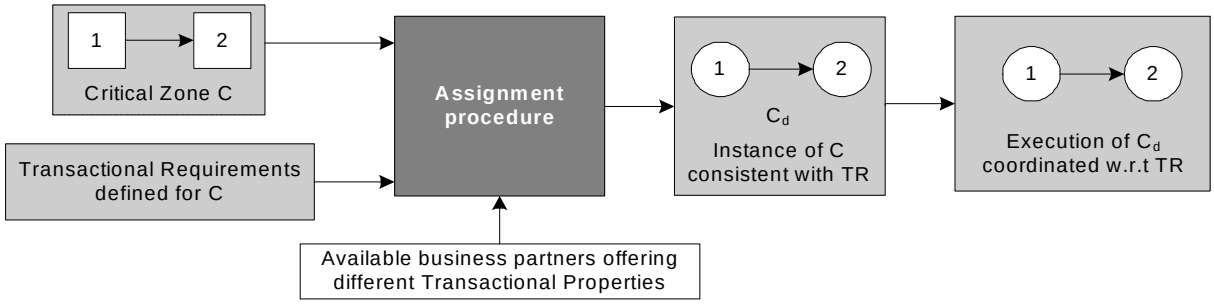

Figure 3.2: Methodology

We adopt a simple transactional protocol wherein the coordination is managed in a centralized manner by b_1^v assigned to v_1. The role of the coordinator consists in making decisions based on the transactional requirements defined for the critical zone given the overall state of workflow execution so that the critical zone execution can reach a consistent state of termination. The coordination is assured in a hierarchical way and the business partners $(b_k^v)_{k\in[1,j]}$ which are subcoordinators report directly to b_1^v whereas the partners b_k^m report to the business partner b_x^v most recently executed [1]. For the sake of simplicity, we consider that the set of business partners $\{b_l^m, b_{l+1}^m, ...b_p^m\}$ reporting to the business partner b_x^v form an abstract partner named $b_{l,p}^m$ that is assigned to the abstract vertex $m_{l,p}$. C therefore denotes a set of n vertices (abstract or not) $C = (c_a)_{a\in[1,n]}$. This reporting strategy based on the type of business partners is depicted in figure 3.1.

Within the pervasive workflow model, the workflow execution is performed by business partners which are assigned to vertices at runtime. Considering the diversity of business partners encountered in the pervasive setting, we assume that these partners might offer various transactional properties, in addition to different functional capabilities. For instance, a business partner can have the capability to compensate the effects of a given operation or to re-execute the operation after failure as possible transactional properties whereas some other business partner does not have any of these capabilities. It thus becomes necessary to select the business partners executing a critical zone of a pervasive workflow not only based on functional requirements but also according to transactional ones. The business partner assignment procedure through which business partners are assigned to vertices based on functional requirements has to be augmented to integrate transactional ones. The assignment procedure based on transactional requirements follows the same strategy as the one based on functional requirements. It is a match-making procedure between the transactional properties offered by business partners and the transactional requirements associated to each task. The purpose of the business partner assignment procedure consists in building an instance of C consistent with the transactional requirements imposed by designers. It is thus required to discover first all the business partners that will be involved in the execution of a given critical zone prior to the execution in order to verify the existence of a set of business partners that can be assigned to C i.e. use the source discovery mode during a critical zone execution. Once the instance of C has been created, the execution supported by the coordination protocol can start. The coordination process is based on rules

[1] business partner of type b_k^v that is located on the same branch of the workflow as these b_k^m business partners and that has most recently completed its execution.

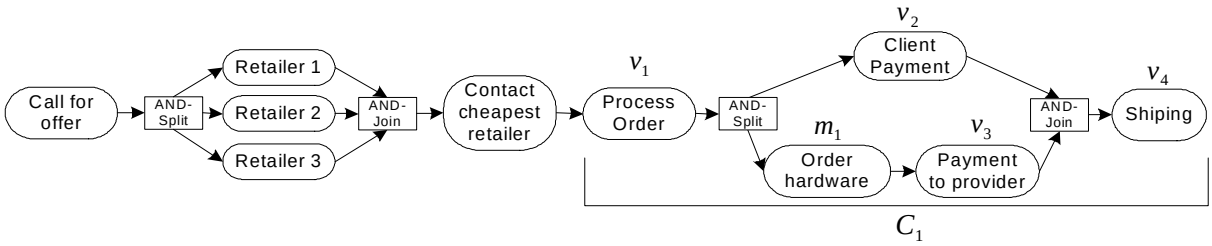

Figure 3.3: Workflow example: Deal at a fair

deduced from the transactional requirements and that specify the final states of execution or termination states each business partner has to reach so that the overall process reaches a consistent termination state. The execution of the coordination protocol therefore consists of two phases: the first phase that includes the discovery and assignment of business partners to vertices and the second one with the actual execution.

3.2.2 Methodology

As described in section 3.2.1, a coordination protocol designed to support the execution of pervasive workflows has to meet two basic requirements. First, business partners have to be assigned based on a transaction-aware process. Second, a runtime mechanism should process and assure the coordination of the execution in the face of failure scenarios. In our approach, the partners part of a critical zone instance C_d are selected according to their transactional properties by means of a matchmaking procedure. We therefore need to specify first the semantic associated with the transactional properties offered by business partners. The matchmaking procedure is indeed based on this semantic. This semantic is also used in order to define a tool allowing workflow designers to specify their transactional requirements for a given critical zone. Based on these transactional requirements, business partners can be assigned to workflow vertices. Finally, once C_d is formed we can proceed towards the second goal by expressing the coordination rules inherent to C_d and designing the actual coordination protocol in charge of processing those rules. This methodology basically follows the steps of the transactional pervasive workflow lifecyle from the instantiation to the execution as depicted in figure 3.2.

3.3 Motivating example

In this section we describe a motivating example that will be used throughout the chapter to illustrate the design methodology. We consider a workflow executed during a computer fair where clients, retailers and hardware providers can exchange electronically orders and invoices. The workflow used in this example is depicted in figure 3.3. Alice would like to buy a new computer and makes a call for offer to three available retailers. After having received some

offers, she decides to go for the cheapest one and therefore contacts the corresponding retailer Bob. Bob initiates the critical zone C_1 by sending an invoice to Alice and contacting his hardware provider Jack (vertex v_1). Alice pays using Bob's trusted payment platform (vertex v_2). In the meantime Jack receives the order from Bob and sends him an invoice (vertex m_1) which he pays (vertex v_3) using Jack's trusted payment platform. Afterwards, Bob starts to build the computer and ships it to Alice (vertex v_4).

Of course in this example, we need to define transactional requirements as for instance Bob would like to have the opportunity to cancel his payment to Jack if Alice's payment is not done. Likewise, Alice would like to be refunded if Bob does not manage to assemble and ship the computer. These different scenarios refer to characteristics offered by the business partners or services assigned to the workflow tasks. For example, the payment platform should be able to compensate Alice's payment and Jack's payment platform should offer the possibility to cancel an order. Yet, it is no longer necessary for Jack to provide the cancellation option if the payment platform claims that it is reliable and not prone to transaction errors. In this example we do not focus on the trust relationship between the different entities and therefore assume the trustworthiness of each of them yet we are rather interested in the transactional characteristics offered by each participant.

3.4 Transactional model

In this section, we provide the semantic specifying the transactional properties offered by business partners before specifying the consistency evaluation tool associated with this semantic. The semantic model is based on the "transactional Web service description" defined in [BPG05].

3.4.1 Transactional properties of business partners

In [BPG05] a model specifying semantically the transactional properties of Web services is presented. This model is based on the classification of computational tasks made in [MRSK92, SAS99] which considers three different types of transactional properties. A task and by extension a business partner executing this task can be of type:

- **Compensatable**: the data modified by the task can be rolled back,
- **Retriable**: the task is sure to complete successfully after a finite number of tries,
- **Pivot**: the task is neither compensatable nor retriable.

In the definition of a critical zone, we distinguish two sets of business partners: $(b_k^m)_{k\in[1,i]}$ which only modify mobile or volatile data and $(b_k^v)_{k\in[1,j]}$ which only modify data other than

$TS(C_1)$	v_1	v_2	m_1	v_3	v_4
ts_1	completed	completed	completed	completed	completed
ts_2	completed	completed	completed	completed	failed
ts_3	completed	completed	completed	compensated	failed
ts_4	completed	compensated	completed	completed	failed
ts_5	completed	compensated	completed	compensated	failed
ts_6	compensated	completed	completed	completed	failed
ts_7	compensated	completed	completed	compensated	failed
ts_8	compensated	compensated	completed	completed	failed
ts_9	compensated	compensated	completed	compensated	failed
ts_{10}	completed	completed	completed	failed	aborted
ts_{11}	completed	compensated	completed	failed	aborted
ts_{12}	completed	canceled	completed	failed	aborted
ts_{13}	compensated	completed	completed	failed	aborted
ts_{14}	compensated	compensated	completed	failed	aborted
ts_{15}	compensated	canceled	completed	failed	aborted
ts_{16}	completed	completed	hfailed	aborted	aborted
ts_{17}	completed	compensated	hfailed	aborted	aborted
ts_{18}	completed	canceled	hfailed	aborted	aborted
ts_{19}	compensated	completed	hfailed	aborted	aborted
ts_{20}	compensated	compensated	hfailed	aborted	aborted
ts_{21}	compensated	canceled	hfailed	aborted	aborted
ts_{22}	completed	failed	completed	aborted	aborted
ts_{23}	completed	failed	canceled	aborted	aborted
ts_{24}	compensated	failed	completed	aborted	aborted
ts_{25}	compensated	failed	canceled	aborted	aborted
ts_{26}	completed	failed	completed	completed	aborted
ts_{27}	completed	failed	completed	compensated	aborted
ts_{28}	completed	failed	completed	canceled	aborted
ts_{29}	compensated	failed	completed	completed	aborted
ts_{30}	compensated	failed	completed	compensated	aborted
ts_{31}	compensated	failed	completed	canceled	aborted
ts_{32}	failed	aborted	aborted	aborted	aborted

Figure 3.4: Termination states of C_1

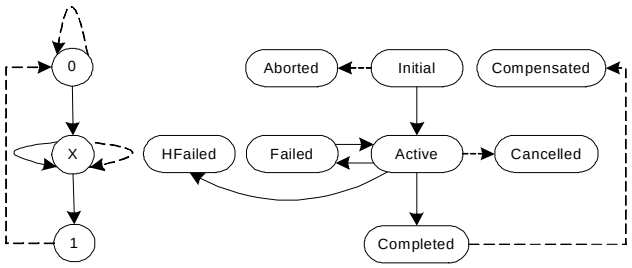

Figure 3.5: State model

mobile ones, e.g. remote database, production of an item, etc. Based on this distinction, the above mentioned transactional model has to be extended. This model describes the modification of permanent data and is thus only relevant to database systems whereas the pervasive setting introduces in addition transactional properties representing business partners' hardware characteristics such as battery level, reliability, connectivity, etc. A new transactional property representing the reliability of a business partner is therefore introduced.

- A business partner is **reliable** (resp. **unreliable**) if it is highly unlikely (resp. likely) that the business partner will fail due to hardware failures (battery level, communication medium access, etc.).

To properly detail this model, we can map the transactional properties with the state of data modified by the business partners during the execution of computational tasks. This mapping is depicted in figure 3.5. Basically, data can be in three different states: initial (0), unknown (x), completed (1):

- In the state (0), it means either that the vertex execution has not yet started $initial$, the execution has been $aborted$ before starting, or the data modified have been $compensated$ after completion.
- In state (1), it means that the vertex has been properly $completed$.
- In state (x), it means either that the execution is $active$, the execution has been stopped, $canceled$ before completion, the execution has $failed$ or an hardware failure, $Hfailed$ happened.

These transactional properties allow to define eight types of business partners:

- (Reliable,Retriable): (r_l,r_t),
- (Reliable,Compensatable): (r_l,c),
- (Reliable,Retriable and Compensatable): (r_l,r_tc),
- (Reliable,Pivot): (r_l,p),
- the four others Unreliable: (ur_l).

Besides, we must distinguish within this model:

- **Inherent termination states**: $failed$, $completed$ and $Hfailed$ which result from the normal course of the task execution,

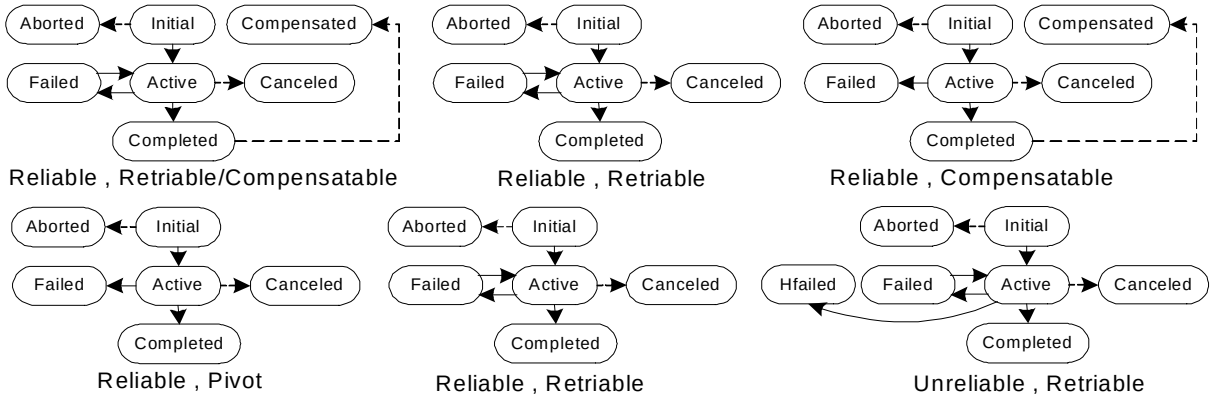

Figure 3.6: State diagrams of business partners b_k^v and b_k^m

- **Forced termination states**: $compensated$, $aborted$ and $canceled$ which result from a coordination message received during a coordination protocol instance and forcing a task execution to either stop or rollback.

In the state diagrams of figures 3.5 and 3.6 plain and dashed lines represent the inherent transitions leading to inherent states and the forced transitions leading to forced states, respectively.

The transactional properties of the business partners are only differentiated by the states $failed$, $compensated$ and $Hfailed$ which indeed respectively specify the retriability, compesatability and reliability aspects.

Definition 3.2 (**Transactional properties**)**.** We have for a given partner b:

- $failed$ is not a termination state of $b \Leftrightarrow b$ is retriable
- $compensated$ is a termination state of $b \Leftrightarrow b$ is compensatable
- $Hfailed$ is not a termination state of $b \Leftrightarrow b$ is reliable

From the state transition diagram, we can also derive some simple rules:

$(R1)$ The states $failed$, $completed$, $Hfailed$ and $canceled$ can only be reached if the business partner is in the state $active$

$(R2)$ The state $compensated$ can only be reached if the partner is in the state $completed$

$(R3)$ The state $aborted$ can only be reached if the partner is in the state $initial$

Regarding the distinction made on the nature of vertices within a critical zone, we specify some requirements for the business partners selected for a critical zone execution.

On the one hand, as the partners $(b_k^v)_{k\in[1,j]}$ modify sensitive and permanent data, we consider that they are required to be reliable. There are therefore four types of b_k^v partners: (r_l,r_t), (r_l,c), (r_l,$r_t c$) and (r_l,p).

On the other hand, as the business partners of type b_k^m only modify mobile and volatile data, we consider first that they are retriable besides compensatability is not required for volatile data. Second, we assume that these tasks can be executed by unreliable partners and there are as a result only two types of b_k^m partners: (r_l,r_t) and (ur_l,r_t). If one of the b_k^m partners part of the abstraction $b_{l,p}^m$ is unreliable then $b_{l,p}^m$ is unreliable, otherwise $b_{l,p}^m$ is reliable. Figure 3.6 depicts the transition diagram for the six types of transactional partners that can be encountered.

3.4.2 Termination states

The crucial point of the transactional model specifying the transactional properties of business partners is the analysis of their possible termination states. The ultimate goal is indeed to be able to define consistent termination states for a critical zone i.e. determining for each partner executing a critical zone vertex which termination states it is allowed to reach.

Definition 3.3 (Termination state operator). We define the operator termination state $ts(x)$ which specifies the possible termination states of the element x. This element x can be:

- a partner b and $ts(b) \in \{aborted, canceled, failed, Hfailed, completed, compensated\}$
- a vertex c and $ts(c) \in \{aborted, canceled, failed, Hfailed, completed, compensated\}$
- a critical zone composed of n vertices $C = (c_a)_{a\in[1,n]}$ and $ts(C) = (ts(c_1), ts(c_2), ..., ts(c_n))$
- an instance C_d of C composed of n partners $C_d = (b_a)_{a\in[1,n]}$ and $ts(C_d) = (ts(b_1), ts(b_2), ..., ts(b_n))$

The operator $TS(x)$ represents the finite set of all possible termination states of the element x, $TS(x) = (ts_k(x))_{k\in[1,j]}$. We have especially, $TS(C_d) \subseteq TS(C)$ since the set $TS(C_d)$ represents the actual termination states that can be reached by C_d according to the transactional properties of the partners assigned to C. We also define for x critical zone or critical zone instance and $a \in [1, n]$:

- $ts(x, c_a)$: the value of $ts(c_a)$ in $ts(x)$
- $tscomp(x)$: the termination state of x such that $\forall a \in [1, n]\ ts(x, c_a) = completed$.

For the remaining of the chapter, $C = (c_a)_{a\in[1,n]}$ denotes a critical zone of n vertices and $C_d = (b_a)_{a\in[1,n]}$ an instance of C.

3.4.3 Transactional consistency tool

We use the Acceptable Termination States (ATS) [RS95] model as the consistency evaluation tool for the critical zone. ATS defines the termination states a critical zone is allowed to reach so that its execution is deemed consistent.

Definition 3.4 (Acceptable Termination States). An $ATS(C)$ is a subset of $TS(C)$ whose elements are considered consistent by workflow designers for a specific execution of C. A consistent termination state of C is called an acceptable termination state $ats_k(C)$, thus $ATS(C) = (ats_k(C))_{k\in[1,i]}$. A set $ATS(C)$ specifies the transactional requirements defined by designers associated with a specific execution of C.

$ATS(C)$ and $TS(C)$ can be represented by a table which defines for each termination state the tuple of termination states reached by each vertex as depicted in figures 3.4 and 3.7. Depending on the application different ATS tables can of course be specified by designers for the same critical zone C, and for the sake of readability we do not introduce in this chapter an index (as in $ATS_i(C)$) in the notation $ATS(C)$. As mentioned in the definition, the specification of the set $ATS(C)$ is done at the workflow design phase. $ATS(C)$ is mainly used as a decision table for a coordination protocol so that C_d can reach an acceptable termination state knowing the termination state of at least one vertex. The coordination decision, i.e. the termination state that has to be reached, made given a state of the critical zone execution has to be unique, this is the main characteristic of a coordination protocol. In order to cope with this requirement, $ATS(C)$ which is used as input for the coordination decision-making process has thus to verify some properties that are specified later on.

3.5 Analysis of *TS(C)*

Since $ATS(C) \subseteq TS(C)$, $ATS(C)$ inherits the characteristics of $TS(C)$ and we logically need to analyze first $TS(C)$. In this section, we first precise some basic properties of $TS(C)$ derived from inherent execution rules of a workflow C before examining $TS(C)$ from a coordination perspective.

3.5.1 Inherent properties of *TS(C)*

We state here some basic properties relevant to the elements of $TS(C)$ and derived from the transactional model presented above. $TS(C)$ is the set of all possible termination states

of C based on the termination states model we chose for business partners. Yet, within a workflow execution, it is not possible to reach all the combinations represented by a n-tuple $(ts(c_1), ts(c_2), ..., ts(c_n))$ assuming $\forall a \in [1, n]\, ts(c_a) \in \{aborted, canceled, failed, completed, compensated\}$. The first restriction is introduced by the sequential aspect of a workflow:

(P_1) A vertex becomes $activated \Leftrightarrow$ all the vertices executed beforehand according to the execution plan of C have reached the state $completed$.

(P_1) simply means that to start the execution of a workflow task, it is required to have properly completed all the workflow vertices required to be executed beforehand.

Second, for the sake of clarity in what follows we consider that only one single vertex can fail at a time. Our approach can indeed be easily extended to concurrent failure scenarios as discussed later on in section 3.7.4. The states $aborted$, $compensated$ and $canceled$ can only be reached by a vertex in a given $ts_k(C)$ if one of the partners assigned to a vertex of C has failed. This means that the coordination protocol is allowed to force the abortion, the compensation or the cancellation only in case of failure of a business partner. We get (P_2):

(P_2) if $\exists a, k$ such that $ts_k(C, c_a) \in \{compensated,\ aborted,\ canceled\} \Rightarrow \exists!\ l$ such that $ts_k(C, c_l) = failed$.

3.5.2 Classification within *TS(C)*

As we explained above the unicity of the coordination decision during the execution of a coordination protocol is a major requirement. We try here to identify the elements of $TS(C)$ that correspond to different coordination decisions given the same state of a workflow execution. The goal is to use this classification to determine $ATS(C)$. Using the properties P_1 and P_2, a simple analysis of the state transition model reveals that there are two situations whereby a protocol coordination has different possibilities of coordination given the state of a workflow task. Assume that the vertex c_b has failed:

- the task c_a is in the state $completed$ and either it remains in this state or it is $compensated$
- the task c_a is in the state $active$ and either it is $canceled$ or the coordinator lets it reach the state $completed$

From these two statements, we define the **incompatibility from a coordination perspective** and the **flexibility**.

Definition 3.5 (Incompatibility from a coordination perspective). Two termination states $ts_k(C)$ and $ts_l(C)$ are said incompatible from a coordination perspective $\Leftrightarrow \exists$ two vertices c_a, c_b such that:

$$\begin{cases} ts_k(C, c_a) = completed \\ ts_k(C, c_b) = ts_l(C, c_b) = failed \\ ts_l(C, c_a) = compensated \end{cases}$$

Otherwise, $ts_l(C)$ and $ts_k(C)$ are said compatible from a coordination perspective.

The value in $\{compensated, completed\}$ reached by a vertex c_a in a termination state $ts_k(C)$ whereby $ts_k(C, c_b) = failed$ is called recovery strategy of c_a against c_b in $ts_k(C)$.

If two termination states are compatible, they correspond to the same recovery strategy against a given vertex. In fact, we have two cases for the compatibility of two termination states $ts_k(C)$ and $ts_l(C)$. Given two vertices c_a and c_b such that $ts_k(C, c_b) = ts_l(C, c_b) = failed$:

- $ts_k(C, c_a) = ts_l(C, c_a)$
- $ts_k(C, c_a) \in \{compensated, completed\}$ and $ts_l(C, c_a) \in \{aborted, canceled\}$

The second case is only possible to reach if c_a and c_b are executed concurrently. Intuitively, the failure of the business partner assigned to c_b occurs at different instants in $ts_k(C)$ and $ts_l(C)$.

Definition 3.6 (Flexibility from a coordination perspective). A vertex c_a is flexible against $c_b \Leftrightarrow \exists ts_k(C)$ such that:

$$\begin{cases} ts_k(C, c_b) = failed \\ ts_k(C, c_a) = canceled \end{cases}$$

Such a termination state is said to be flexible to c_a against c_b. The set of termination states of C flexible to c_a against c_b is denoted $FTS(c_a, c_b)$.

From these definitions, we now study the termination states of C according to the compatibility and flexibility criteria in order to identify the termination states that follow a common strategy of coordination.

Definition 3.7 (Termination state generator). A termination state of C $ts_k(C)$ is called generator of $c_a \Leftrightarrow ts_k(C, c_a) = failed$ and for all vertices c_b executed before or in parallel with c_a, $ts_k(C, c_b) \in \{completed, compensated\}$. The set of termination states of C compatible with $ts_k(C)$ generator of c_a is denoted $CTS(ts_k(C), c_a)$.

The set $CTS(ts_k(C), c_a)$ specifies all the termination states of C that follow the same recovery strategy as $ts_k(C)$ against c_a.

Definition 3.8 (Coordination strategy). Let $ts_k(C) \in TS(C)$ be a generator of c_a. Coordinating an instance C_d of C in case of the failure of c_a consists in choosing the recovery strategy of each task of C against c_a and the $z_a < n$ vertices $(c_{a_i})_{i\in[1,z_a]}$ flexible to c_a whose execution is not $canceled$ when c_a fails. We call coordination strategy of C_d against c_a the set $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a)$ defined as follows.

$$CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = CTS(ts_k(C), c_a) - \bigcup_{i=1}^{z_a} FTS(c_{a_i}, c_a)$$

If the business partner c_a assigned to c_a is retriable then $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset$.

The instance C_d is said to be coordinated according to $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a)$ if in case of the failure of c_a, C_d reaches a termination state in $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a)$. Of course, it assumes that the transactional properties of C_d are sufficient to reach $ts_k(C)$.

From these definitions, we can deduce a set of properties:

Theorem 3.1. *C_d can only be coordinated according to a unique coordination strategy at a time.*

Proof. Two termination states $ts_k(C)$ and $ts_l(C)$ generator of a vertex c_a are incompatible. ■

Theorem 3.2. *Let $ts_k(C)$ such that $ts_k(C, c_a) = failed$ but not generator of c_a. If $ts_k(C) \in TS(C_d) \Rightarrow \exists ts_l(C)$ such that:*

$$\begin{cases} ts_l(C) \in TS(C_d) \\ ts_l(C) \text{ is a generator of } c_a \text{ compatible with } ts_k(C) \end{cases}$$

Proof. We can define $ts_l(C)$ by:

$$\begin{cases} ts_l(C, c_a) = failed \\ ts_l(C, c_i) = ts_k(C, c_i) \text{ if } ts_k(C, c_i) \in \{completed, compensated, aborted\} \\ ts_l(C, c_i) = completed \text{ otherwise} \end{cases}$$

which satisfies the required properties. ■

Given a vertex c_a the idea is to classify the elements of $TS(C)$ using the sets of termination states compatible with the generators of c_a. Using this approach, we can identify the different recovery strategies and the coordination strategies associated with the failure of c_a as we decide which vertices can be $canceled$.

Set 1					Set 2				
Available Business partners		Retriable	Compensatable	Reliable	Available Business partners		Retriable	Compensatable	Reliable
v_1	b_{11}	yes	no	yes	v_1	b_{11}	yes	no	yes
v_2	b_{21}	no	yes	yes	v_2	b_{21}	no	yes	yes
	b_{22}	yes	no	yes		b_{22}	yes	no	yes
m_1	b_{31}	yes	no	no	m_1	b_{31}	yes	no	no
v_3	b_{41}	no	yes	yes		b_{32}	yes	no	yes
v_4	b_{51}	yes	no	yes	v_3	b_{41}	yes	no	yes
	b_{52}	yes	yes	yes	v_4	b_{51}	yes	no	yes

ATS(C_1)		v_1	v_2	m_1	v_3	v_4
ats_1	ts_1	completed	completed	completed	completed	completed
ats_2	ts_4	completed	compensated	completed	completed	failed
ats_3	ts_{11}	completed	compensated	completed	failed	aborted
ats_4	ts_{12}	completed	canceled	completed	failed	aborted
ats_5	ts_{17}	completed	compensated	hfailed	aborted	aborted
ats_6	ts_{18}	completed	canceled	hfailed	aborted	aborted
ats_7	ts_{22}	completed	failed	completed	aborted	aborted
ats_8	ts_{23}	completed	failed	canceled	aborted	aborted
ats_9	ts_{27}	completed	failed	completed	compensated	aborted
ats_{10}	ts_{28}	completed	failed	completed	canceled	aborted
ats_{11}	ts_{32}	failed	aborted	aborted	aborted	aborted

Figure 3.7: $ATS(C_1)$ and available business partners

3.6 Forming *ATS(C)*

Defining $ATS(C)$ is deciding at design time the termination states of C that are consistent. $ATS(C)$ is to be inputted to a coordination protocol in order to provide it with a set of rules which leads to a unique coordination decision in any cases. According to the definitions and properties we have introduced above, we can now explicit some rules on $ATS(C)$ so that the unicity requirement of coordination decisions is respected.

Definition 3.9 (Validity of *ATS*). Let c_a and $ts_k(C)$ such that $ts_k(C, c_a) = failed$ and $ts_k(C)$ belongs to $ATS(C)$. $ATS(C)$ is valid $\Leftrightarrow \exists!$ $ts_l(C)$ generator of c_a compatible with $ts_k(C)$ such that:

$$CTS(ts_l(C), c_a) - \bigcup_{i=1}^{z_a} FTS(c_{a_i}, c_a) \subset ATS(C)$$

for a set of vertices $(c_{a_i})_{i\in[1,z_a]}$ flexible to c_a.

The unicity of the termination state generator of a given vertex comes from the incompatibility definition and the unicity of the coordination strategy. A valid $ATS(C)$ therefore contains for all $ts_k(C)$ in which a vertex fails a unique coordination strategy associated with this failure and the termination states contained in this coordination strategy are compatible with $ts_k(C)$. In figure 3.7, an example of possible ATS is presented for the critical zone C_1. It just consists in selecting the termination states of the table $TS(C_1)$ that we consider consistent and respect the validity rule for the created $ATS(C_1)$. For example here the payment of Alice has to be compensated if Bob fails to deliver the computer as specified in $ats_2 = ts_4$.

3.7 Assigning business partners using *ATS*

In this section, we introduce a new type of business partner assignment procedure: the transaction-aware partner assignment procedure which aims at assigning n business partners to the n vertices c_a in order to create an instance of C *acceptable* with respect to a valid $ATS(C)$. The goal of this procedure is to integrate within the instantiation process of workflows a systematic method ensuring the transactional consistency of the obtained workflow instance. We first define a validity criteria for the instance C_d of C with respect to $ATS(C)$, the business partner assignment algorithm is then detailed. In a third time, we specify the coordination strategy associated with the instance created from our assignment scheme and finally discuss and illustrate our results.

3.7.1 Acceptability of C_d with respect to *ATS(C)*

Definition 3.10 (Acceptability of an instance). C_d is an acceptable instance of C with respect to $ATS(C)$ iff $TS(C_d) \subseteq ATS(C)$.

Now we express the condition $TS(C_d) \subseteq ATS(C)$ in terms of coordination strategies. The termination state generator of c_a present in $ATS(C)$ is noted $ts_{k_a}(C)$. The set of vertices whose execution is not $canceled$ when c_a fails is noted $(v_{a_i})_{i\in[1,z_a]}$.

Theorem 3.3. $TS(C_d) \subseteq ATS(C) \Leftrightarrow \forall a \in [1,n]\ CS(C_d, ts_{k_a}(C), (v_{a_i})_{i\in[1,z_a]}, c_a) \subset ATS(C)$

Proof. straightforward derivation from the theorem 3.2 and the definition 3.9. ■

An instance C_d of C is therefore an acceptable one $\Leftrightarrow$ it is coordinated according to a set of n coordination strategies contained in $ATS(C)$. It should be noted that if $failed \notin ATS(C, c_a)$ where $ATS(C, c_a)$ represents the acceptable termination states of the task c_a in $ATS(C)$ then $CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset$. From the theorem 3.2 and the definition 3.10, we can derive the existence condition of an acceptable instance of C with respect to a valid $ATS(C)$.

Theorem 3.4. *Let $ts_k(C)$ and c_a such that $ts_k(C, c_a) = failed$ and $ts_k(C) \in ATS(C)$. $\exists C_d$ acceptable instance of C with respect to $ATS(C)$ such that $ts_k(C) \in TS(C_d) \Leftrightarrow \exists!\ ts_l(C)$ such that:*

$$\begin{cases} ts_l(C) \in TS(C_d) \\ ts_l(C) \textit{ is a generator of } c_a \textit{ compatible with } ts_k(C) \textit{ in } ATS(C) \end{cases}$$

This theorem only states that an $ATS(C)$ allowing the failure of a given vertex can be used to coordinate a workflow instance also allowing the failure of the same vertex $\Leftrightarrow$ $ATS(C)$ contains a complete coordination strategy associated to this vertex, i.e. it is valid.

3.7.2 Transaction-aware assignment procedure

In this section, we present the procedure that is used to assign business partners to vertices based on transactional requirements. The business partner assignment algorithm uses $ATS(C)$ as a set of requirements during the partner assignment procedure and thus identifies from a pool of available partners those whose transactional properties match the transactional requirements associated with vertices defined in $ATS(C)$. The assignment procedure is an iterative process, partners are assigned to vertices sequentially. At each step i, the assignment procedure therefore generates a partial instance of C noted C_d^i. $TS(C_d^i)$ refers to the termination states of C that can be reached based on the transactional properties of the i partners that are already assigned. Intuitively the acceptable termination states refer to the degree of flexibility offered when choosing the partners with respect to the different coordination strategies complying with $ATS(C)$. This degree of flexibility is influenced by two parameters:

- The list of acceptable termination states for each workflow vertex. This list can be determined based on $ATS(C)$. Using this list, the requirements on the transactional properties of a candidate partner can be derived since this partner can only reach the states defined in $ATS(C)$ for the considered vertex.
- The assignment process is iterative and therefore, as new partners are assigned to vertices, both $TS(C_s^i)$ and the transactional properties required for the assignment of further partners are updated. For instance, we are sure to no longer reach the termination states $CTS(ts_k(C), c_a)$ allowing the failure of the vertex c_a in $ATS(C)$ when we assign a partner of type retriable and reliable to c_a. In this specific case, we no longer care about the states reached by other vertices in $CTS(ts_k(C), c_a)$ and therefore there is no transactional requirements introduced for the vertices to which business partners have not already been assigned.

We therefore need to define first the transactional requirements for the assignment of a partner after i steps in the assignment procedure.

Extraction of transactional requirements

From the two requirements above, we define for a vertex c_a :

- $ATS(C, c_a)$: Set of acceptable termination states of c_a which is derived from $ATS(C)$
- $DIS(c_a, C_d^i)$: Set of transactional requirements that the partner assigned to c_a must meet based on previous assignments. This set is determined based on the following reasoning:

  (DIS_1): the partner must be compensatable iff $compensated \in DIS(c_a, C_d^i)$

  (DIS_2): the partner must be retriable iff $failed \notin DIS(c_a, C_d^i)$

(DIS_3): the partner must be reliable iff $Hfailed \notin DIS(c_a, C_d^i)$

Using these two sets, we are able to compute the set $Min_{TP}(b_a, c_a, C_d^i)$ defined by:

$$Min_{TP}(b_a, c_a, C_d^i) = ATS(C, c_a) \cap DIS(c_a, C_d^i)$$

which defines the minimal transactional properties a partner b_a has at least to comply with in order to be assigned to the vertex c_a at the $i+1$ assignment step. We simply check the retriability and compensatability properties for the set $Min_{TP}(b_a, c_a, C_d^i)$:

- $failed \notin Min_{TP}(b_a, c_a, C_d^i) \Leftrightarrow b_a$ has to verify the retriability property
- $Hfailed \notin Min_{TP}(b_a, c_a, C_d^i) \Leftrightarrow b_a$ has to verify the reliability property
- $compensated \in Min_{TP}(b_a, c_a, C_d^i) \Leftrightarrow b_a$ has to verify the compensatability property

The set $ATS(C, c_a)$ is easily derived from $ATS(C)$. We need now to compute $DIS(c_a, C_d^i)$. We assume that we are at the $i+1$ step of an assignment procedure, i.e. the current partial instance of C is C_d^i. Computing $DIS(c_a, C_d^i)$ means determining if (DIS_1), (DIS_2) and (DIS_3) are true. From these three statements we can derive four properties:

1. (DIS_1) implies that state $compensated$ can definitely be reached by c_a
2. (DIS_2) implies that c_a can not $fail$
3. (DIS_2) implies that c_a can not be $canceled$
4. (DIS_3) implies that c_a can not $Hfail$

The third property is derived from the fact that if a vertex can not be $canceled$ when the failure of a vertex has occurred, then it has to finish its execution and reach at least the state $completed$. In this case, if a business partner can not be $canceled$ then it can not fail, which is the third property. To verify whether 1., 2., 3. and 4. are true, we present the following theorems.

Theorem 3.5. *The state $compensated$ can definitely be reached by c_a iff $\exists b \in [1, n] - \{a\}$ verifying:*

$$\begin{cases} b_b \textit{ not retriable (resp. reliable) is assigned to } c_b \\ \exists ats_k(C) \textit{ generator of } c_b \textit{ such that } ats_k(C, c_a) = compensated \end{cases} \quad \textbf{(3.5b)}$$

Proof. $\Leftarrow$: Since the partner b_b is not retriable (resp. reliable), it can fail (resp. hfail) and $ats_k(C)$ generator of c_b such that $ats_k(C, c_a) = compensated$ is in $TS(C_d)$.

$\Rightarrow$: Derived from (P_2) and 3.2. ■

The two following theorems are proved similarly:

Theorem 3.6. *c_a can not fail (resp. Hfail) iff $\exists b \in [1,n] - \{a\}$ verifying:*

$$\begin{cases} b_b \text{ not compensatable is assigned to } c_b \\ \exists ats_k(C) \text{ generator of } c_a \text{ such that } ats_k(C, c_b) = compensated \end{cases} \quad or \quad \textbf{(3.6b)}$$

$$\begin{cases} c_b \text{ is flexible to } c_a \\ b_b \text{ not retriable is assigned to } c_b \\ \forall ats_k(C) \text{ such that } ats_k(C, c_a) = failed \text{ (resp. } Hfailed), ats_k(C, c_b) \neq canceled \end{cases}$$

Theorem 3.7. *Let c_a, c_b such that c_a is flexible to c_b. c_a is not canceled when c_b fails (resp. Hfail) iff the following holds:*

$$\begin{cases} b_b \text{ not retriable (resp. reliable) is assigned to } c_b \\ \forall ats_k(C) \text{ such that } ats_k(C, c_b) = failed \text{ (resp. } Hfailed), ats_k(C, c_a) \neq canceled \end{cases} \quad \textbf{(3.7b)}$$

Based on the theorems 3.5, 3.6 and 3.7, in order to compute $DIS(c_a, C_d^i)$, we have to compare c_a with each of the i vertices $c_b \in C - \{c_a\}$ to which a business partner b_b has been already assigned. Two cases have to be considered: either we assign a business partner to a vertex v_k or to an abstract vertex $m_{l,p}$. This is an iterative procedure. At the initialization phase in the first case we have: since no vertex has been yet compared to $c_a = v_k$, b_a can be of type (r_l, p): $DIS(c_a, C_d^i) = \{failed\}$.

1. if c_b verifies (*3.5b*) $\Rightarrow compensated \in DIS(c_a, C_d^i)$
2. if c_b verifies (*3.6b*) $\Rightarrow failed \notin DIS(c_a, C_d^i)$
3. if c_b is flexible to c_a and verifies (*3.7b*) $\Rightarrow failed \notin DIS(c_a, C_d^i)$

In this case, the verification stops if

$$\begin{cases} failed \notin DIS(c_a, C_d^i) \\ compensated \in DIS(c_a, C_d^i) \end{cases}$$

For the vertices of type v_k, we indeed only need to check the retriability and compensatability properties.

In the second case, we have at the initialization phase: since no vertex has been yet compared to $c_a = m_{l,p}$, b_a can be of type *(ur_l)*: $DIS(c_a, C_d^i) = \{Hfailed\}$.

4. if c_b verifies (*3.6b*) $\Rightarrow Hfailed \notin DIS(c_a, C_d^i)$

In that case, the verification stops if $Hfailed \notin DIS(c_a, C_d^i)$. For the vertices of type $m_{l,p}$ we only need to check the reliability property.

Finally, when $Min_{TP}(b_a, c_a, C_d^i)$ is computed, we are able to select the appropriate business partner to be assigned to a given vertex according to transactional requirements.

Business partner assignment process

Business partners are assigned to each vertex based on an iterative process. Depending on the transactional requirements and the transactional properties of the business partners available for each vertex, different scenarios can occur:

(i) Business partners of type *(r_l, r_tc)* are available in the case of a vertex v_k or business partners of type (r_l) are available in the case of a vertex $m_{l,p}$ (i.e. all the business partners of the abstraction are of type (r_l)). It is not necessary to compute any transactional requirements as such partners match all transactional requirements.

(ii) A single partner is available for the considered vertex. We need to compute the transactional requirements associated with the vertex and either the transactional properties offered by this partner are sufficient or there is no solution.

(iii) Business partners of type *(r_l, r_t)* and *(r_l, c)* but none of type *(r_l, r_tc)* are available for a vertex v_k. We need to compute the transactional requirements associated with the vertex and we have three cases. First, *(r_l, r_tc)* is required and therefore there is no solution. Second, *(r_l, r_t)* (resp. *(r_l, c)*) is required and we assign a business partner of type *(r_l, r_t)* (resp. *(r_l, c)*) to the vertex. Third, there is no requirement.

The assignment procedure is performed by the coordinator c_1. Business partners have to be assigned to all vertices prior to the beginning of the critical zone execution. The first vertex is de facto assigned to the critical zone initiator. The idea is then to assign first business partners to the vertices verifying *(i)* and *(ii)* since there is no flexibility in the choice of the business partner. Vertices verifying *(iii)* are finally analyzed. Based on the transactional requirements raised by the remaining vertices, we first assign partners to vertices with a non-empty transactional requirements. We then handle the assignment for vertices with an empty transactional requirements. Note that the transactional requirements of all the vertices to which partners are not yet assigned are also affected (updated) as a result of the current partner assignment. If no vertex has transactional requirements then we assign the partners of type *(r_l, r_t)* to assure the completion of the remaining vertices' execution.

Theorem 3.8. *The business partner assignment procedure creates an instance of C that is acceptable with respect to a valid $ATS(C)$.*

Proof. Let C_d be an instance of C resulting from the service assignment procedure and a service b_a assigned to a task c_a in C_d. The definition 3.10 has to be verified and we therefore consider (A) and (B) using the notations of theorem 3.3:

(A) $\forall a \in [1,n]$, $failed \in ATS(C,c_a) \Rightarrow CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) \subset ATS(C)$

(B) $\forall a \in [1,n]$, $failed \notin ATS(C,c_a) \Rightarrow CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) \subset ATS(C)$

(A): We suppose that $failed \in ATS(C,c_a)$ then we have two possibilities: b_a is retriable and $CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset \subset ATS(C)$. b_a can fail and with 1, 2 and 3 we get $ts_{k_a}(C) \in TS(C_d)$ and therefore $CS(C_d, ts_{k_a}(C), (c_{a_i})_{k\in[1,z_a]}, c_a) \subset ATS(C)$ since $ATS(C)$ is valid.

(B): We suppose that $failed \notin ATS(C,c_a)$ then $failed \notin Min_{TP}(b_a, c_a, C_d^i)$ and b_a is retriable. Therefore, $CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset \subset ATS(C)$.

Finally, we get $CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) \subset ATS(C)$ and C_d is an acceptable instance of C with respect to $ATS(C)$. In this proof we only consider the $failed$ case for the sake of simplicity, the $hfailed$ case can be proved similarly. ■

3.7.3 Actual termination states of C_d

Once all the business partners have been assigned to vertices we can coordinate their execution so that they respect the defined transactional requirements. In order to do so, we need to know the actual termination states subset of ***ATS(C)*** that can be reached by the defined instance of C. Having computed $TS(C_d)$, we can deduce the coordination rules associated with the execution of C_d. This subset is determined using the following theorem.

Theorem 3.9. *Let C_d be an acceptable instance of C with respect to $ATS(C)$. We note $(c_{a_i})_{i\in[1,n_r]}$ the set of vertices to which neither a retriable nor a reliable business partner has been assigned. $ts_{k_{a_i}}(C)$ is the generator of c_{a_i} present in $ATS(C)$ and $(v_{a_{i_j}})_{j\in[1,z_{a_i}]}$ denotes the set of vertices which are not canceled when c_{a_i} fails.*

$$TS(C_d) = \{tscomp(C_d)\} \cup \bigcup_{i=1}^{n_r} (CTS(ts_{k_{a_i}}(C), c_{a_i}) - \bigcup_{j=1}^{z_{a_i}} FTS(v_{a_{i_j}}, c_{a_i}))$$

$TS(C_d)$ is indeed derived from $ATS(C)$ which contains for all vertices at most a single coordination strategy as specified in definition 3.9. As a result, whenever the failure of a vertex c_a is detected, a transactional protocol in charge of coordinating an instance C_d resulting from our approach reacts as follows. The coordination strategy $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ corresponding to c_a is identified and a unique termination state belonging to the coordination strategy $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ can be reached given the current state of the critical zone execution.

3.7.4 Discussion and performance evaluation

In order to handle the scenarios wherein more than one business partner can fail at a time, one would need to extend the definition of the termination state generator to take into account the failure of a set of partners. The compatibility definition would be also defined for termination states in which exactly the same set of partners fail at the same time so that coordination strategies are defined for possible concurrent failures. In this case, two termination states such that the set of failures in the first is a subset of the set of failures in the second are not incompatible. The composition algorithm and the coordination protocol are not affected by this configuration.

The operations that are relevant from the complexity point of view are twofold: the definition of transactional requirements by means of the acceptable termination states model and the execution of the transaction-aware business partner assignment procedure.

One can argue that building an *ATS* table specifying the transactional requirements of a business process W consists of computing the whole $TS(W)$ table, yet this is not the case. Building a $ATS(C)$ set in fact only requires for designers to identify the vertices of C that they allow to fail as part of the process execution and to select the termination state generator associated with each of those vertices that meet their requirements in terms of failure atomicity. Once this phase is complete, designers only need to select the vertices whose execution can be canceled when the former vertices may fail and complete the associated coordination strategy.

The second aspect concerns the complexity of the transaction aware assignment procedure that we presented in section 3.7.

Theorem 3.10. *Let $C = (c_a)_{a\in[1,n]}$ a critical zone. The complexity of the transaction-aware assignment procedure is $O(n^3)$.*

Proof. We can show that the number of operations necessary to compute the step i of the assignment procedure for a vertex c_a is bounded by $4 \times n \times i$. Computing the step i indeed consists of verifying the theorems 3.5, 3.6 and 3.7 and determining $ATS(C, c_a)$. On the one hand, performing the operations part of theorems 3.5 (one comparison), 3.6 (two comparisons) and 3.7 (one comparison) requires at most 4 comparisons. On the other hand, building $ATS(C, c_a)$ requires at most n operations (there is at most n generators in a $ATS(C)$ set). Therefore, we can derive that the number of operations that needs to be performed in order to compute the n steps of the assignment procedure for a critical zone composed of n tasks is bounded by $4 \times n \times \sum_{j=1}^{n} j$ which is equivalent to n^3 as $n \longrightarrow \infty$. ■

3.7.5 Examples

Two examples are outlined in this section to illustrate the transaction-aware assignment procedure presented in this chapter.

Example 3.1. We consider the critical zone C_1 of figure 3.3. Designers have defined $ATS(C_1)$ of figure 3.7 as the transactional requirements. The set of available business partners for each vertex of C_1 is the set 1 depicted in figure 3.7. The goal is to assign business partners to vertices so that the instance of C_1 is valid with respect to $ATS(C_1)$ and we apply the presented assignment procedure. The critical zone initiator assigned to v_1 uses a business partner of type (r_l, r_t) matching the transactional requirements.

We now start to assign the business partners of type $(r_l, r_t c)$ and (r_l) for which it is not necessary to compute any transactional requirements. b_{52} which is the only available business partner of type $(r_l, r_t c)$ is therefore assigned to v_4.

We then try to assign business partners to tasks for which there is no choice, and we verify whether b_{31} can be assigned to m_1. We compute:

$$\begin{aligned} Min_{TP}(b_a, m_1, C^2_{1d}) &= ATS(C_1, m_1) \cap DIS(m_1, C^2_{1d}) \text{ we have:} \\ ATS(C_1, m_1) &= \{completed, Hfailed\} \\ DIS(m_1, C^2_{1d}) &= \{Hfailed\} \end{aligned}$$

as b_{52} and b_{11} are the only business partners already assigned and the theorems 3.5, 3.6 and 3.7 are not verified. Thus $Min_{TP}(c_a, m_1, C^2_{1d}) = \{Hfailed\}$ and b_{31} can be assigned to m_1 as it matches the transactional requirements.

We get for v_3:

$$Min_{TP}(b_a, v_3, C^3_{1d}) = \{failed\}$$

The business partner b_{41} which is of type (r_l, c) verifies the transactional requirements is assigned to v_3.

Now we compute the transactional requirements of v_2 and we get:

$$Min_{TP}(b_a, v_2, C^4_{1d}) = \{failed, compensated\}$$

as theorem 3.5 is verified with the business partners b_{31}. The partner b_{21} can thus be assigned to v_2 as it matches the transactional requirements of the task. Using the created instance of C_1 we get the set $TS(C_{1d})$ of figure 3.8.

Example 3.2. We consider the critical zone C_1 of figure 3.3. Designers have defined $ATS(C_1)$ of figure 3.7 as the transactional requirements. The set of available business partners for each vertex of C_1 is the set 2 depicted in figure 3.7. The critical zone initiator assigned to v_1 uses a business partner of type (r_l, r_t) matching the transactional requirements.

We now assign the business partners of type $(r_l, r_t c)$ and (r_l) that meet any transactional requirements. b_{32} which is of type (r_l) is therefore assigned to m_1.

$TS(C_{1d})$	b_{11}	b_{21}	b_{31}	b_{41}	b_{51}
ts_1	completed	completed	completed	completed	completed
ts_{11}	completed	compensated	completed	failed	aborted
ts_{12}	completed	canceled	completed	failed	aborted
ts_{17}	completed	compensated	hfailed	aborted	aborted
ts_{18}	completed	canceled	hfailed	aborted	aborted
ts_{22}	completed	failed	completed	aborted	aborted
ts_{23}	completed	failed	canceled	aborted	aborted
ts_{27}	completed	failed	completed	compensated	aborted
ts_{28}	completed	failed	completed	canceled	aborted

Figure 3.8: $TS(C_{1d})$

We then try to assign business partners to tasks for which there is no choice, and we verify whether b_{41} can be assigned to v_3. We compute:

$$\begin{aligned} Min_{TP}(b_a, v_3, C^2_{1d}) &= ATS(C_1, v_3) \cap DIS(v_3, C^2_{1d}) \text{ we have:} \\ ATS(C_1, v_3) &= \{completed, failed, compensated\} \\ DIS(v_3, C^2_{1d}) &= \{failed\} \end{aligned}$$

as b_{11} and b_{31} are the only business partners already assigned and the theorems 3.5, 3.6 and 3.7 are not verified. Thus $Min_{TP}(c_a, v_3, C^2_{1d}) = \{failed\}$ and b_{41} can be assigned to v_3 as it matches the transactional requirements.

We verify whether b_{51} can be assigned to v_4. We compute:

$$\begin{aligned} Min_{TP}(b_a, v_4, C^3_{1d}) &= ATS(C_1, v_4) \cap DIS(v_4, C^3_{1d}) \text{ we have:} \\ ATS(C_1, v_4) &= \{completed, failed\} \\ DIS(v_4, C^3_{1d}) &= \{failed\} \end{aligned}$$

as b_{11}, b_{31}, b_{41} are the only business partners already assigned and the theorems 3.5, 3.6 and 3.7 are not verified. Thus $Min_{TP}(c_a, v_3, C^2_{1d}) = \{failed\}$ and b_{51} can be assigned to v_4 as it matches the transactional requirements.

Now we compute the transactional requirements of v_2 and we get:

$$Min_{TP}(b_a, v_2, C^4_{1d}) = \{failed\}$$

as theorems 3.5, 3.6 and 3.7 are not verified. We have the choice for the assignment of the business partner and we choose b_{22} that is retriable.

b_1^v

Receives	Sends
Completed	Compensate
Failed	Cancel
Aborted	Abort
Canceled	Leave
Compensated	Ping
Ack	
Hfailed	
Alive	

b_k^v

Receives	Sends
Compensate	Abort
Cancel	Aborted
Abort	Canceled
Leave	Cancel
Aborted	Leave
Alive	Ping
Ack	Ack
Ping	Hfailed
Canceled	Alive
Completed	Compensated
	Failed
	Completed

b_k^m

Receives	Sends
Leave	Aborted
Cancel	Canceled
Ack	Alive
Abort	Ack
Ping	Completed

Figure 3.9: Notification messages

3.8 Coordination Protocol Specification

Having introduced the method through which an instance of C is obtained by assigning partners to workflow vertices according to the transactional requirements of C, we turn to the actual coordination of partners during the execution of the critical zone. The protocol that is in charge of the coordination is specified in terms of the different actors, notification messages and coordination cases. We finally motivate the chosen solution by comparing it with existing coordination protocols.

3.8.1 Protocol actors

As mentioned in section 3.2.1 and figure 3.1 we distinguish three main entities within the coordination protocol execution:

- **The business partner** $b_1^v = c_1$: this business partner is the critical zone initiator and is in charge of performing the business partner assignment procedure and coordinating the execution of C. The coordination decisions are made using the table $TS(C_d)$ specifying the subset of termination states belonging to $ATS(C)$ that the instance C_d is actually able to reach.
- **Business partners** b_k^v: these business partners modify sensitive data and play the role of subcoordinators. They report their state of execution and the state of execution of the business partners b_k^m to b_1^v .
- **Business partners** b_k^m: these partners modify volatile data and report to the business partner b_x^v most recently executed.

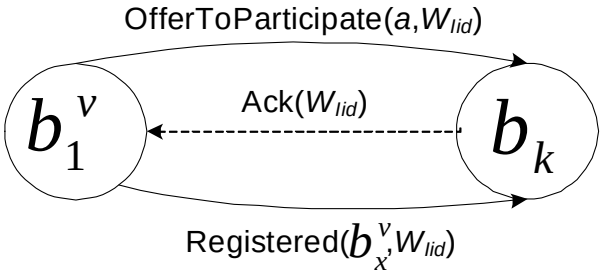

Figure 3.10: Business partner registration

Actors exchange messages for the purpose of decision making and forwarding as listed in figure 3.9. These messages are mostly derived from the state diagram of the transactional model and the respective role of the partners in the protocol. The flow of notification messages within the protocol execution and the mechanisms involved in the processing of these notification messages are stated in the next section.

3.8.2 Coordination scenarios

In this section, we detail the different phases and coordination scenarios that can be encountered during the execution of the protocol. First, we explain how partners are registered with the coordination protocol during the partner assignment phase. Then, we analyze the message flow between the different actors of the protocol in three different scenarios: normal course of execution, failure of a partner b_k^v and failure of a partner b_k^m.

Business partner registration

The first phase of the coordination protocol consists of the discovery and registration of the business partners that will be involved in the critical zone execution. The discovery process through which business partners that can be assigned to critical zone vertices are identified is performed by the business partner $c_1 = b_1^v$. The transactional requirements extraction procedure, specified in section 3.7.2 provides the coordinator with a list of suitable business partners that match the computed transactional requirements. It is then necessary to contact the business partners of this list in order to receive from of one of them the commitment to execute the requested vertex. Based on the registration handshake depicted in figure 3.10, the coordinator b_1^v contacts a business partner asking it whether it agrees to commit to execute the operation a of the workflow whose identifier is W_{iid}. Once the newly assigned business partner's coordinator is known, b_1^v sends the information. In the case of business partners b_k^v this information is known from the beginning since b_1^v is their coordinator whereas for the business partners b_k^m, the information is known when b_x^v the business partner b_k^v most recently executed has been assigned to a vertex.

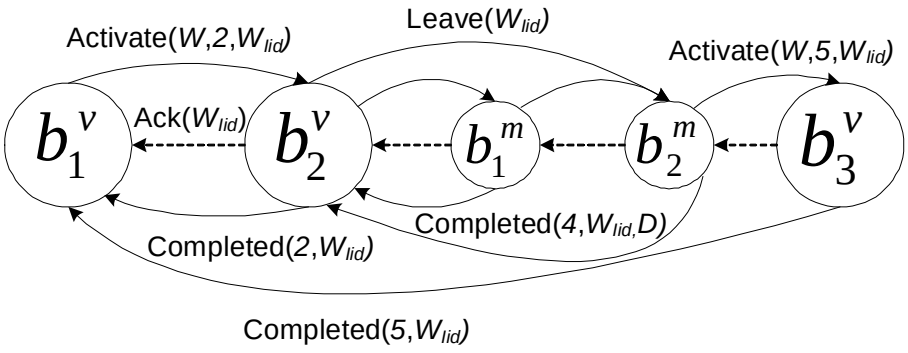

Figure 3.11: Normal execution

Normal course of execution

Once all involved business partners are known, the critical zone execution can start supported by the coordination protocol. Business partners are sequentially activated based on the workflow specification. A sample for normal execution of C is depicted in figure 3.11.

The $Activate(W, k, W_{iid}, D)$ message is in fact a simple workflow message as defined in chapter 2, it especially contains the workflow specification W, the requested vertex k to be executed, the workflow data D modified during the execution and the workflow identifier W_{iid}.

Within the critical zone execution local acknowledgments $Ack(W_{iid})$ are used. Each business partner b_k^m reports its status to the business partner b_x^v most recently executed and once its execution is complete it can leave the critical zone execution.

The $Completed(k, W_{iid}, D)$ message sent by a business partner b_k^m includes a backup copy of the volatile data modified by the business partner that can be reused later on for the recovery procedure in case of failure of a business partner b_k^m specified in section 3.8.2.

Once in the state $completed$, business partners of type b_k^m can leave the coordination as they will not be asked to compensate their execution. Depending on the transactional requirements defined for C, business partners b_k^v may leave the critical zone before the end of the critical zone execution. A business partner b_k^v is indeed able to leave the coordination if it reaches the state $completed$ regardless of possible failures in the sequel of the critical zone execution. The condition allowing a business partner b_k^v to leave the coordination is therefore stated as follows.

Theorem 3.11. *A partner b_k^v assigned to a vertex c_l can leave the execution of a critical zone C iff the partner b_k^v is in the state $completed$ and $\forall i \in [1, n]$ such that a business partner b_i not retriable (resp. not reliable) is assigned to the vertex c_i, b_i is in the state $initial$ and $ts_k(C, c_l) = completed$ where $ts_k(C)$ is the termination state generator of c_i in $TS(C_d)$.*

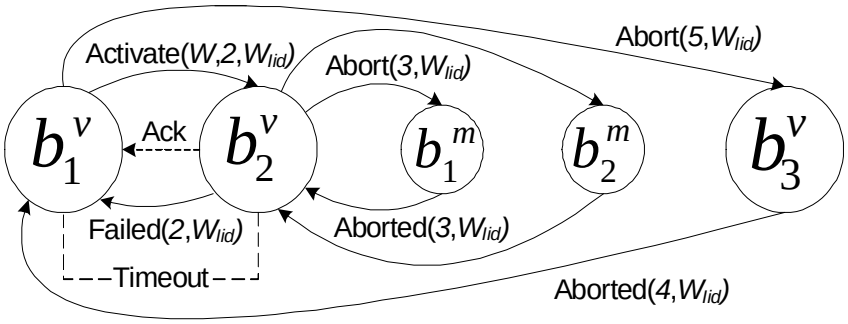

Figure 3.12: Failure of a business partner b_k^v

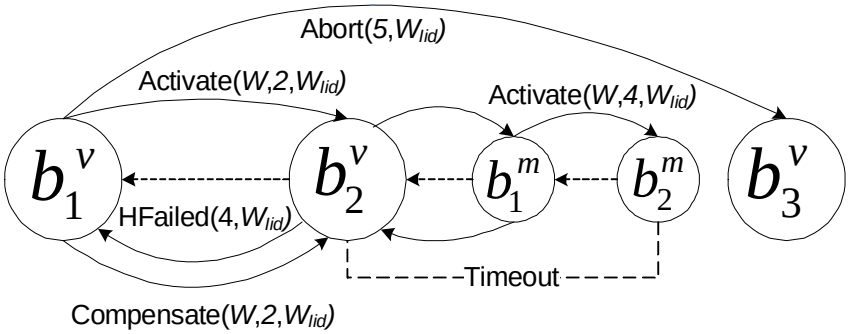

Figure 3.13: Failure of a business partner b_k^m

Failure of a business partner b_k^v

This scenario is only possible with business partners of type (r_l, p). We can encounter two situations:

- the failure is total and the business partner is not able to communicate any longer,
- the business partner is still alive and can forward a failure message to b_1^v.

Figure 3.12 depicts the two cases whereby the total failure is detected using a simple timeout in Ping/Alive message exchanges. Once the failure has been detected, the coordinator forwards the coordination decision to all involved business partners. It should be noted here that business partners of type (r_l, r_t) can also reach the state $failed$ but the retriability property implies that they have at their disposal recovery solutions ensuring that the contact is never permanently lost. Thus, the failure of business partners of type (r_l, r_t) is transparent to the rest of the coordination and does not have to be handled.

Failure of a business partner b_k^m

The failure of a business partner b_k^m is detected by its subcoordinator with a timeout. As specified in the transactional model, we indeed consider that business partners of type b_k^m can only fail because of hardware problems and failure of such partners therefore implies a loss of contact with their coordinator. The failure of a partner b_k^m is reported by its subcoordinator to the partner b_1^v. The failure detection and forwarding of the $Hfailed$ message are depicted in figure 3.13.

3.8.3 Coordination decisions and recovery

Having detailed various coordination scenarios that can occur during the execution of a critical zone, we analyze the possible recovery strategies, in particular the replacement of failed partners b_k^v and b_k^m and how coordination decisions are made upon detection of a failure.

Replacement of failed partners b_k^v

During the course of the execution new partners can be discovered and assigned to vertices in order to replace failed ones. In fact two situations can happen:

- the failure of a partner occurs while executing its assigned vertex
- the coordinator loses contact (timeout detection) prior to the activation of the partner

The first situation is specified in the previous section and no backup solution is possible as the data modified by the failed business partner are in an unknown state. In the second situation, it is possible on the contrary to assign a new business partner matching the transactional requirements to the vertex which has not yet been activated. Once the loss of contact with a business partner b_k^v is detected, no coordination decision is yet sent to business partners and the execution continues. If no business partner is found to be assigned to the vertex when its execution should be activated, the protocol coordinator considers the business partner it has lost contact with as failed.

Replacement of failed business partners b_k^m

In case of failure of a business partner b_k^m, be it before or after its activation, a recovery procedure can be executed prior to informing the coordinator of the hardware failure. It is indeed possible to assign to the vertex a new business partner so that the execution can go on. This is

possible as on the one hand the partners b_k^m only modify volatile data and on the other hand, we have a backup copy of the data modified by the partners that are part of the abstract vertex $b_{l,p}^m$. Once the failure is detected, the subcoordinator of the failed partner tries to assign a new partner to the failed vertex. In this case, only volatile data are being modified, transactional requirements are not a concern and the assignment procedure can be repeated till a business partner manages to execute the requested vertex.

Reaching consistent termination states

Once all possible recovery mechanisms have been attempted, a coordination decision is made by the coordination b_1^v. The table $TS(C_d)$ is the input to the coordination decisions that are made throughout the execution of a critical zone. Once the failure of a vertex c_a has been detected, the protocol coordinator reads in $TS(C_d)$ the set $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ listing the possible termination states reachable by C_d whereby c_a is $failed$. There is a unique element of this set that is reachable by C_d with respect to its current state of execution and b_1^v sends the appropriate messages so that the overall critical zone can reach this consistent termination state. The unicity of the termination state reachable by C_d given its current state of execution is ensured by the building process of the Acceptable Terminate State table that is presented in section 3.6.

3.8.4 Discussion

The coordination protocol integrates the semantic description of involved business partners and relies on an adaptive decision table which is computed during the assignment procedure. The coordination protocol is flexible as it completely depends on the designers' choice for the specification of Acceptable Termination States. This solution therefore offers a full support of relaxed atomicity constraints for workflow based applications and is also self-adaptable to the business partners' characteristics, which is not the case with recent efforts [La05a],[La05b].

The organization of the coordination is based on a simple hierarchical approach as in BTP [Aa05]. In that respect, the central point of the coordination is the business partner b_1^v on which relies the whole coordination. This is the main weakness of the protocol, as a failure of this business partner would cause the complete failure of the workflow execution. The role of critical zone initiator of the coordination is therefore reserved to business partners that are both reliable and retriable. Nonetheless, this centralized and hierarchical approach facilitates the management of the coordination process.

In addition to usual coordination phases such as coordination registration, business partner completion and failure, our protocol offers the possibility to replace participants at runtime depending on their role within the coordination and the volatility degree of data they have to modify during the workflow execution. This makes the protocol flexible and adapted to the pervasive paradigm whereas such recovery procedure is not specified in other transactional

protocols.

In the protocol description, we do not specify the data recovery strategy especially for the compensated states. Different approaches can be integrated with our work to support either forward error recovery or backward error recovery [LA90]. The choice of the recovery strategy basically depends on the application and its fault-handling protocol. For instance, a simple backward error recovery strategy is sufficient for workflows used for payment in the example of the chapter whereas a forward recovery strategy might be required for a hotel booking system. Existing mechanisms in this area can therefore be used to augment our transactional protocol to specify complex fault-handling and compensation scenarios [BPE], [TIRL03, TI05, BCCT05].

3.9 Implementation

In this section an implementation of the work presented in this chapter is described. The overall system architecture is depicted in figure 3.14. The basic pervasive workflow infrastructure spans over the business partners taking part in a workflow instance. A local workflow engine developed on top of BPEL [BPE] is in charge of handling, for each involved business partner, the workflow management and control tasks which mainly consist of:

- Receiving and forwarding workflow requests,
- Issuing discovery requests,
- Invoking the appropriate local services to execute workflow tasks.

In order to support the execution of pervasive workflows, we implemented in the fashion of the WS-Coordination initiative [La05c] a transactional stack composed of the following components:

- **Transactional coordinator**: this component is supported by a critical zone initiator. On the one hand it implements the transaction-aware business partner assignment procedure as part of the composition manager module and on the other hand it is in charge of assuring the coordinator role of the transactional protocol relying on the set $TS(C_d)$ outcome of the assignment procedure.
- **Transactional submanager**: this component is deployed on the other partners and is in charge of forwarding coordination messages from the local workflow to the appropriate subcoordinator or coordinator and conversely.

In the remainder of this section, our implementation is described in terms of the implementation of the transaction-aware partner assignment procedure, the internal communications that take

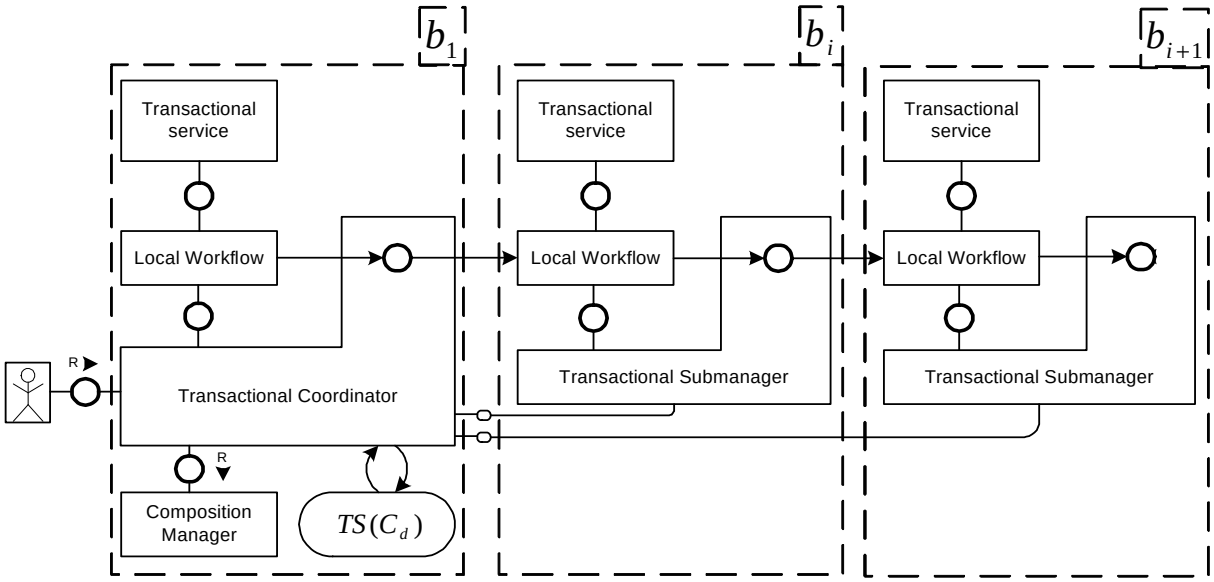

Figure 3.14: Architecture

place between the elements deployed on a business partner and the structure that the BPEL processes deployed on each business partner's workflow engine should be compliant with in order to support the coordination protocol execution.

3.9.1 OWL-S transactional and functional matchmaker

To implement the assignment procedure presented in this chapter we augmented an existing functional OWL-S matchmaker [TLJ03] with transactional matchmaking capabilities. In order to achieve our goal, the matchmaking procedure has been split into two phases. First, the functional matchmaking based on OWL-S semantic matching is performed in order to identify subsets of the available partners that meet the functional requirements for each workflow vertex. Second, the implementation of the transaction-aware partner assignment procedure is run against the selected sets of partners in order to build an acceptable instance fulfilling defined transactional requirements.

The structure of the matchmaker consists of several components whose dependencies are displayed in figure 3.15. The composition manager implements the matchmaking process and provides a Java API that can be invoked to start the selection process. It gets as input an abstract process description specifying the functional requirements for the candidate partners and a table of acceptable termination states. The registry stores OWL-S profiles of partners that are available. Those OWL-S profiles have been augmented with the transactional properties offered by business partners. This has been done by adding to the non-functional information of the OWL-S profiles a new element called *transactionalproperties* that specifies three Boolean attributes that are retriable, reliable and compensatable as depicted in table 3.1.

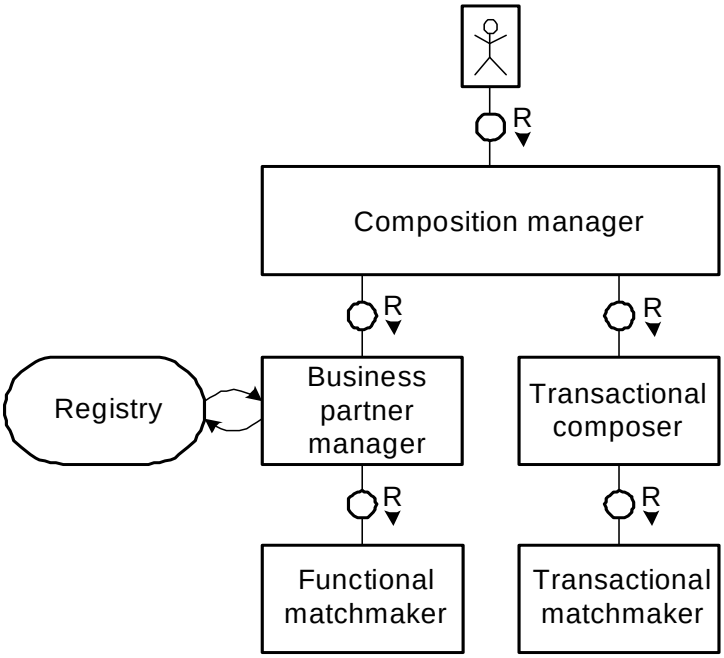

Figure 3.15: OWL-S transactional matchmaker

In the first phase of the selection procedure, the business partner manager is invoked with a set of OWL-S profiles that specify the functional requirements for each workflow vertex. The business partner manager gets access to the registry, where all published profiles are available and to the functional matchmaker which is used to match the available profiles against the functional requirements specified in the workflow. For each workflow vertex, the business partner manager returns a set of functionally matching profiles along with their transactional properties. The composition manager then initiates the second phase, passing these sets along with the process description, and the table of acceptable termination states to the transactional composer. The transactional composer starts the transaction-aware business partner assignment procedure using the transactional matchmaker by classifying first those sets into six groups:

- sets including only business partners of type (ur_l,r_t)
- sets including only business partners of type (r_l,r_t)
- sets including only business partners of type (r_l,p)
- sets including only business partners of type (r_l,c)
- sets including business partners of types (r_l,r_t) and (r_l,c)
- sets including business partners of type (r_l,r_tc)

Once those sets are formed the iterative transactional composition process takes place as specified above based on the table of acceptable termination states. Depending on the set of

```
<tp:transactionalproperties retriable="true"
                            reliable="true"
                            compensatable="true"/>
```

Table 3.1: Listing OWL-S specifying the possible transactional properties

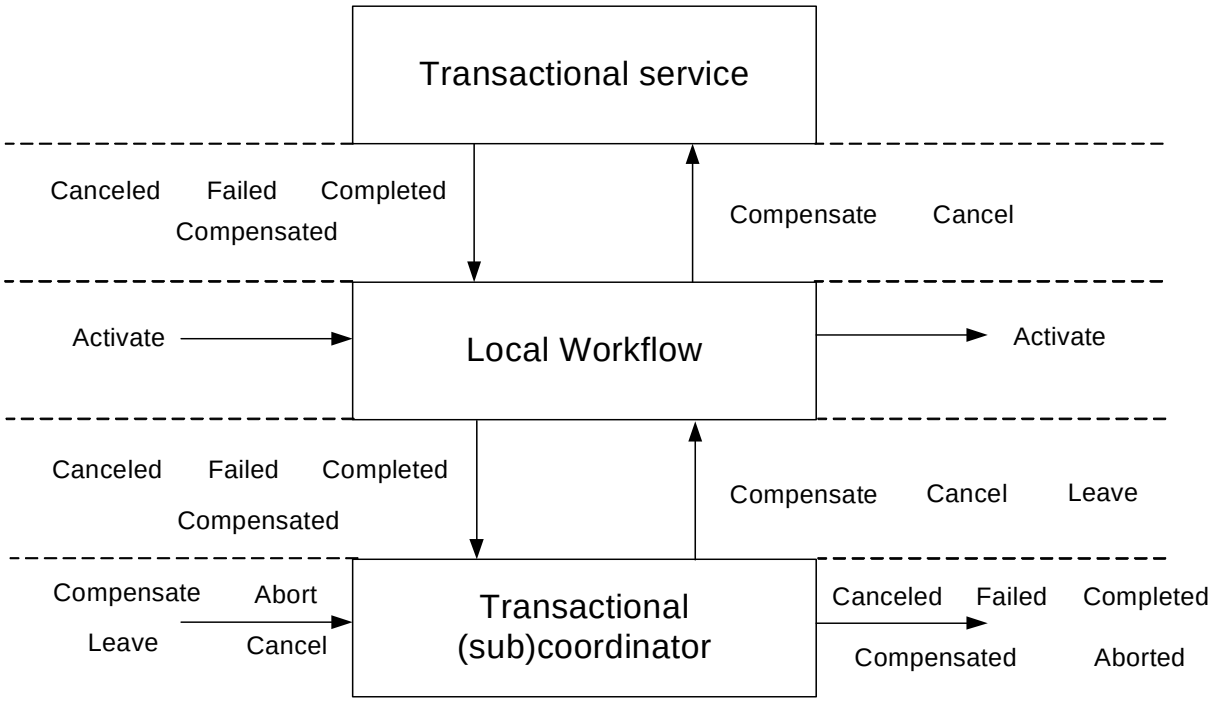

Figure 3.16: Infrastructure internal communications

available services and the specified acceptable termination states, the algorithm may terminate without finding a solution.

3.9.2 Internal communication within a business partner infrastructure

In the infrastructure that is deployed on each business partner to implement the transactional protocol presented in this chapter, the transactional coordinator or subcoordinator plays the role of interface between the business process and the other business partners when it comes to managing the notification messages exchanged during the execution of the transactional protocol. Some of these messages received by the transactional coordinator should be forwarded to the local business process to take appropriate actions while some others are only relevant to the local transactional (sub)coordinator. The business process may also require to issue a notification to its local transactional (sub)coordinator when a failure occurs. As for the notification messages exchanged between the business partners involved in a workflow, the messages exchanged between these three layers are derived from the state model depicted in figure 3.5. The infrastructure deployed on a given business partner basically consists of three layers:

- **The transactional service layer** representing the business partner's available operations,
- **The local workflow layer** corresponding to the local workflow engine,
- **The coordination layer** implementing the local (sub)coordinator module.

The message exchanges that can take place on a given business partner between these three layer are depicted in figure 3.16.

- **Activate**: the activate message is basically issued by the local workflow engine to the local workflow engine of the next business partner involved in the workflow. In fact this message instantiates the process execution on the business partner side.
- **Compensate, Cancel**: the compensate and cancel messages are received at the coordination layer and forwarded to the local workflow layer that forwards them in a second time to the transactional service layer to perform to corresponding functions i.e. compensation or cancellation of an operation.
- **Compensated, Canceled, Completed**: these messages simply notify that the corresponding events have occurred: compensation, cancellation, or completion of an operation. Issued at the transactional service layer, they are forwarded to the coordination layer in order to be dispatched to the coordinator associated with the business partner.
- **Failed**: issued at the transactional service layer, the failed message is forwarded to the coordination layer in order to be dispatched to the coordinator associated with the business partner. If the operation performed at the transactional service layer is retriable, no failed message is forwarded to the local workflow layer as we consider that the retry primitive is inherent to any retriable operation.
- **Leave**: the leave message is received at the coordination layer and the business partner can leave the execution of the pervasive workflow execution. The leave message is forwarded to the local workflow layer if the business partner implements an operation that is compensatable. In this case, the business process deployed on the local workflow engine can have two possible outcomes, either the results produced by its execution are compensated or it can leave the process execution.
- **Abort, Aborted**: the abortion message is received at the coordination layer and acknowledged with an aborted message. Upon receipt of this message, the business simply leaves the pervasive workflow execution, no message is forwarded to the other layers since the local workflow has not yet been instantiated.

The other types of notification messages specified in figure 3.9 are only processed by the local coordinator and are as a result not propagated within the infrastructure deployed on a business partner.

```
<receive createInstance="yes" operation="launch" partnerLink="PLT"
         portType="PT" variable="Data">
    <correlations>
      <correlation initiate="yes" pattern="in" set="CS1"/>
    </correlations>
</receive>
```

Table 3.2: Listing BPEL associated with the process instantiation

```
<eventHandlers>
  <onMessage partnerLink="PLT" portType="PT"
             operation="Cancel" variable="workflowid">
      <correlations>
        <correlation set="CS1"/>
      </correlations>
    <terminate/>
  </onMessage>
</eventHandlers>
```

Table 3.3: Listing BPEL associated with the cancel message

3.9.3 Specification of transactional BPEL processes

In our implementation, the local workflow engine is implemented using BPEL as the workflow specification language. In order to support the message exchanges identified in section 3.9.2 the structure of BPEL business processes has to match some templates that we describe in this section. Using the constructs available in the BPEL language, the specification of these transactional BPEL processes is straightforward.

The business process activation is performed using the usual BPEL process instantiation construct `<receive>` described in table 3.2.

The cancel message can be received at any moment during the execution of the process and is thus handle using the `<eventHandlers>` construct as depicted in table 3.3. Of course the BPEL process has to expose a dedicated operation to receive the cancel message.

In order to detect the failure of an operation that is not retriable, the `<scope>` and the `<faultHandlers>` constructs are used as depicted in table 3.4. The failure of the operation is forwarded to the transactional coordination layer inside the `<faultHandlers>`.

Finally, if the business process implements an operation that is compensatable, the process execution can lead to two possible outcomes depending on whether a compensate or a leave message is received. We use the `<pick>` construct to express this choice as depicted in table 3.5. It should be noted that in the listings depicted in this section, we use BPEL correlation sets because the coordination messages are received asynchronously during the process execution and need to be mapped to the appropriate instance of the workflow to be processed by the

```
<scope name="invokation_try">
  <faultHandlers>
    <catchAll>
       <invoke inputVariable="failedid" name="1"
               operation="transacFailed" partnerLink="PLT"
               portType="LocalAdminImpl"/>
    </catchAll>
  </faultHandlers>
  <invoke inputVariable="DataInc" outputVariable="DataOut"
          name="invoke1" operation="Addition"
          partnerLink="PLT" portType="Add">
  </invoke>
</scope>
```

Table 3.4: Listing BPEL associated with an operation that can fail

```
<pick>
  <onMessage partnerLink="PLT" portType="publicPT"
             operation="Leave" variable="workflowid">
    <correlations>
      <correlation set="CS1"/>
    </correlations>
    <empty/>
  </onMessage>
  <onMessage partnerLink="PLT" portType="PT"
             operation="Compensate" variable="workflowid">
    <correlations>
       <correlation set="CS1"/>
    </correlations>
    <invoke inputVariable="serviceid" name="invoke1"
            operation="Compensate" partnerLink="PLT" portType="Add"/>
  </onMessage>
</pick>
```

Table 3.5: Listing BPEL associated with the leave message

engine.

These BPEL listings can be combined in the design of transactional BPEL processes depending of course on the transactional properties offered by business partners. Two examples of transactional BPEL processes are depicted in figure 3.17. For instance, if the task executed by a business partner is not compensatable, the associated BPEL process only ends with the completed notification since it is not required to wait for a leave message. Similarly, a task which is retriable is not surrounded by `<scope>` constructs as there is no fault to catch.

3.10 Related work

Transactional consistency of distributed systems such as workflows and database systems has been an active research topic over the last 15 years [GR93] yet it is still an open issue in the area of distributed processes within the Service Oriented Computing paradigm (SOC) [CKM+03, Gud04, Lit03, TKM04, SK02]. In this chapter we specified a composition algorithm that makes it possible to build consistent workflow instances with respect to transactional requirements and a transactional protocol for the pervasive workflow architecture presented in chapter 2. In this section, we compare our results with existing solutions in these two research fields.

3.10.1 Integration of transactional requirements into workflows

The execution of distributed processes wherein business partners are not assigned at design time raises new requirements for transactional systems such as dynamicity, semantic description and relaxed atomicity [SABS02]. Existing transactional models for advanced applications and workflows [ERS99, Elm92] do not offer sufficient flexibility to integrate these requirements [AAA+96, GC02] especially when it comes to handling long-running transactions. For instance in the saga [GMGK+91] model, it is required to identify the subtransactions of an application to support its execution and of course this is not compatible with composite applications that are built on the fly. Moreover, the saga model assumes that subtransactions are compensatable which is not an assumption adapted to current business applications. In contrast, our solution allows the specification of transactional requirements supporting relaxed atomicity for an abstract workflow specification and the selection of semantically described business partners or services fulfilling the defined transactional requirements. In addition, we provide the means to compute a coordination protocol suited to the workflow instance resulting from our business partner assignment procedure.

In [BPG05, BGP05], the first approach specifying relaxed atomicity requirements for Web services based workflow applications using the *ATS* tool and a transactional semantic is presented. Despite a solid contribution, this work provides only some means to verify the consistency of composite services but it does not take into account transactional requirements at

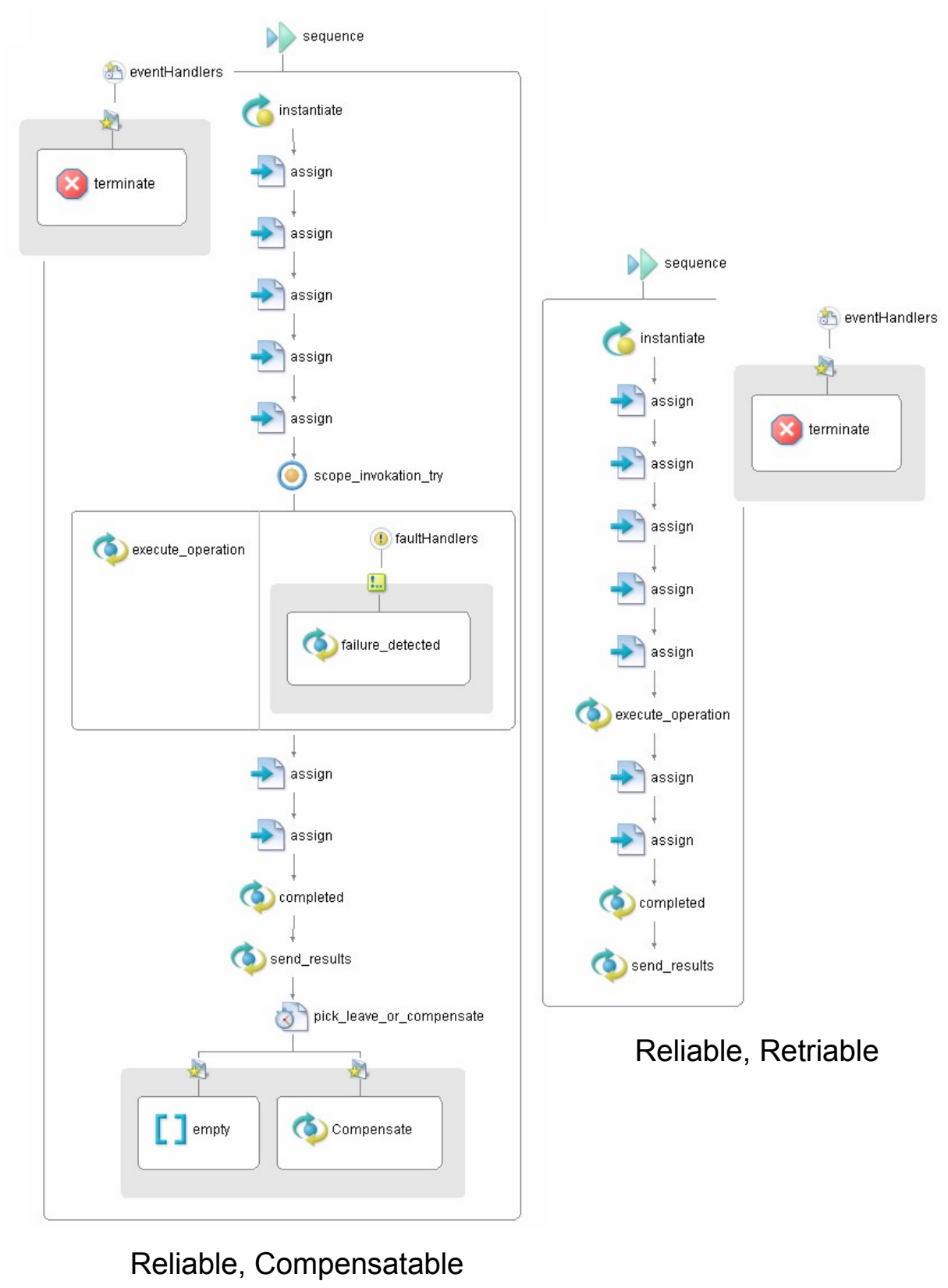

Figure 3.17: Transactional BPEL processes (Process graphs from ActiveBPEL engine)

the composition phase. This work therefore appears to be limited when it comes to the possible integration into dynamic and distributed business processes. In this approach, transactional requirements do not play any role in the component business partners selection process which may result in several attempts to determine a valid workflow instance. As opposed to this work, our solution provides a systematic procedure enabling the creation of valid workflow instances by means of a transaction-aware business partner assignment procedure. Transactional Web service composition framework are also presented in [FDDB05] and [LZ04] yet these approaches do not allow to define coordination strategies as fine-grained as the termination state model underpinning our composition algorithm. A hierarchical approach based on nested transactions [WS92] has been recently presented in [HMR07] to handle concurrency issues in composite applications. The results of this work can perfectly be used to extend our transaction-aware composition procedure when it comes to the selection of component business partners that themselves implement composite applications. This way fine-grained transactional requirements spanning over each and every component service part of a workflow instance could be specified.

3.10.2 Transactional protocols and frameworks

The transactional protocol proposed in this chapter offers suitable means to respond to the constraints introduced by environments where heterogeneous business partners share resources in a collaborative manner. Using relaxed atomicity features, the protocol indeed offers the flexibility for business partners to release their resources as soon as their participation to the workflow is no longer required. Moreover using a flexible semantic, business partners are able to advertise their capabilities so that they can assume a role suited to any workflow in which their resources can be used. Current efforts in the design of transactional framework supporting the coordination of business processes in a Service Oriented Computing paradigm [La05a, La05b, ACKM03, Pap03] do not offer such flexibility. They suffer from the lack of tools for the specification of transactional requirements and their integration into a dynamic business partners' selection process. Furthermore, no recovery procedure is specified as part of the protocol for the replacement of business partners in case of failure.

Fault handling within BPEL processes has also been an active research area over the past years [FHF05]. In [CCKM05, NK03] a procedure is specified for the placement of fault handlers and event handlers in decentralized executions of BPEL processes. Starting from a centralized BPEL process wherein transactional requirements are specified, this work presents a procedure that outputs a decentralized execution of BPEL processes that is equivalent in terms of fault and exception to the centralized one. Transactional properties of the component BPEL processes within the decentralized execution are however not integrated in this procedure as opposed to the approach we have presented in this chapter.

In [MTR02] the WSTx framework is proposed that is a transactional framework supporting the execution of composite applications. WSTx implements a transactional protocol that is based on the transactional properties or transactional attitudes offered by the business partners

involved in a collaborative application. In contrast to our work, this framework does not propose any systematic procedure to automate the composite application building process based on transactional requirements specified by clients or designers.

3.11 Conclusion

This chapter presents an adaptive transactional protocol to support the execution of pervasive workflows. Our solution enables first the selection of the business partners taking part in the workflow execution based on transactional requirements defined at the workflow design phase. Transactional requirements are defined by designers and serve as an input to define reliable workflow instance. Using transactional properties offered by selected business partners and the defined transactional requirements, a decision table is computed as a basis for the coordination of the execution. The coordination protocol itself offers a framework that supports relaxed atomicity requirements and takes into account the respective role and characteristics of each business partner involved in the workflow execution.

Besides, a complete transactional framework based on the Web services technologies has been implemented as a proof of concept of our theoretical results. On the one hand the business partner assignment procedure we designed can be used to augment existing composition systems [ADK+05] as it can be fully integrated in existing functional match-making procedures. On the other hand, our approach defines adaptive coordination rules that can be deployed on recent coordination specifications [La05c] in order to increase their flexibility.

Chapter 4

Security of Pervasive Workflows

If your enemy is secure at all points, be prepared for him. If he is in superior strength, evade him. If your opponent is temperamental, seek to irritate him. Pretend to be weak, that he may grow arrogant. If he is taking his ease, give him no rest. If his forces are united, separate them. If sovereign and subject are in accord, put division between them. Attack him where he is unprepared, appear where you are not expected.
- Sun Tzu -

4.1 Introduction

In this thesis so far, we introduced the pervasive workflow concept that is a fully decentralized workflow management system and a transactional framework providing reliability to pervasive workflow instances. However, in order to build a complete workflow management system, security solutions are required. Because of their dynamicity and unusual execution environments, pervasive workflows raise new security requirements as opposed to usual centralized workflow management systems. Pervasive workflows can not indeed rely on a trusted centralized coordination mechanism to manage the most basic execution primitives such as message routing between business partners. As a result, basic security features such as integrity of workflow execution assuring the compliance of the overall sequence of operations with the pre-defined workflow execution plan are no longer guaranteed. In addition, tracing back the identity of the business partners involved in a pervasive workflow instance becomes an issue without a trusted centralized coordination mechanism selecting workflow participants. Yet, existing decentralized workflow management systems do not incorporate the appropriate mechanisms to meet the new security requirements in addition to the ones identified in the centralized setting. Even

though some recent research efforts in the field of distributed workflow security have indeed been focusing on issues related to the management of rights in business partner assignment or detecting conflicts of interest [ACM01],[CLW05],[KPF01], basic security issues related to the security of the overall workflow execution such as integrity and evidence of execution have not yet been addressed.

In this chapter, we present security solutions supporting the secure execution of pervasive workflows. These solutions do not only answer specific requirements introduced by the pervasive workflow model but are sufficiently generic to be applied to other workflow architectures in the decentralized setting such as [BMR96, BM04]. These mechanisms capitalize on onion encryption techniques [SGR97] and security policy models in order to assure the integrity of the distributed execution of workflows, to prevent business partners from being involved in a workflow instance forged by a malicious peer and to provide business partners' identity traceability for sensitive workflow instances. The design of the suggested mechanisms is strongly coupled with the runtime specification of decentralized workflow management systems which eases their integration into existing distributed workflow management solutions.

The remainder of the chapter is organized as follows. Section 4.2 outlines the security requirements associated with the pervasive workflow model. In section 4.3 our solution is specified while in section 4.4 the runtime specification of the secure distributed workflow execution is presented. Section 4.5 presents the security analysis of the proposed mechanisms. In section 4.6 we present how the security solutions we propose in this chapter can be integrated within the transactional protocol we specified in chapter 3. Finally section 4.8 discusses related work and section 4.9 presents the conclusion.

4.2 Security requirements

As opposed to centralized workflow management systems the distributed execution of workflows raises security constraints due to the lack of a dedicated infrastructure assuring the management and control of the workflow execution. As a result, security features such as compliance of the workflow execution with the pre-defined plan are no longer assured. We group the security requirements we identified for the pervasive workflow system into three main categories: authorization, proofs of execution and data protection.

4.2.1 Authorization

The main security requirement for a workflow management system is to ensure that only authorized business partners are assigned to workflow tasks during an instance. In the decentralized setting, the assignment of workflow tasks is managed by partners themselves relying on a service discovery mechanism. In this case, the business partner assignment procedure enforces

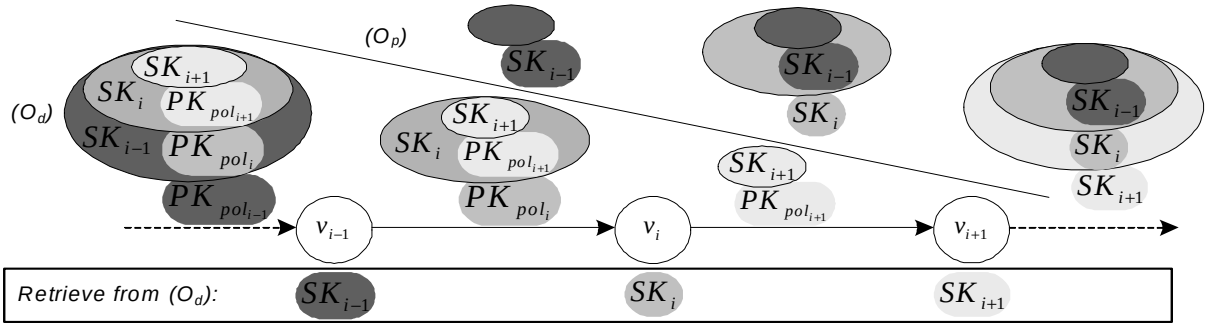

Figure 4.1: Key management

a matchmaking procedure whereby business partners' security credentials are matched against security requirements for tasks.

4.2.2 Execution proofs and traceability

A decentralized workflow management system does not offer any guarantee regarding the compliance of actual execution of workflow tasks with the pre-defined execution plan. Without any trusted coordinator to refer to, the business partner b_i assigned to the vertex v_i needs to be able to verify that the vertices scheduled to be executed beforehand were actually executed according to the workflow plan. This is a crucial requirement to prevent any malicious peer from forging a workflow instance.

In our workflow execution model, candidate business partners are selected at runtime based on their compliance with a security policy. Partners' involvement in a business process can thus remain anonymous as their identity is not assessed in the partner selection process. In some critical business scenarios however, disclosing partners' identity may be required so that in case of dispute or conflict on the outcome of a sensitive task the stakeholders can be identified. In this case, the revocation of business partners' anonymity should only be feasible for some authorized party in charge of arbitrating conflicts, preserving the anonymity of identity traces is thus necessary.

4.2.3 Workflow data protection

In the case of decentralized workflow execution, the set of workflow data denoted $D = (d_k)_{k\in[1,j]}$ is transferred from one business partner to another. This raises major requirements for workflow data security in terms of integrity, confidentiality and access control as follows:

- **Data confidentiality**: for each vertex v_i, the business partner b_i assigned to v_i should only be authorized to read a subset D_i^r of D

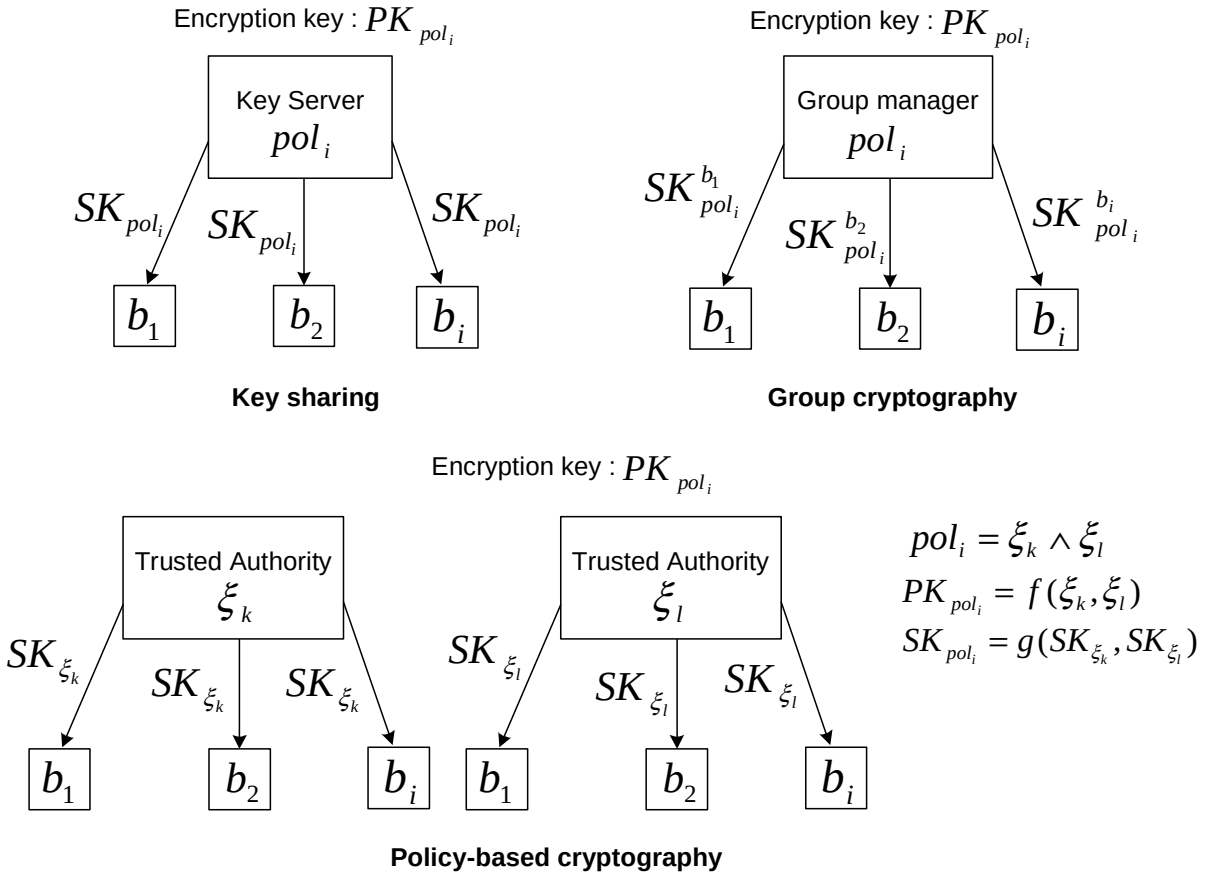

Figure 4.2: Policy private key distribution schemes

- **Data integrity**: for each vertex v_i, the business partner b_i assigned to v_i should only be authorized to modify a subset D_i^w of D_i^r
- **Access control**: the subsets D_i^r and D_i^w associated with each vertex v_i should be determined based on the security policy of the workflow

The solution we developed towards meeting these security requirements is presented in the next section.

4.3 The solution

In this section the mechanisms we designed in order to meet the security requirements we identified for the pervasive workflow management system are specified. The solution is mainly described in terms of the key management, data protection, execution proofs and communication protocol.

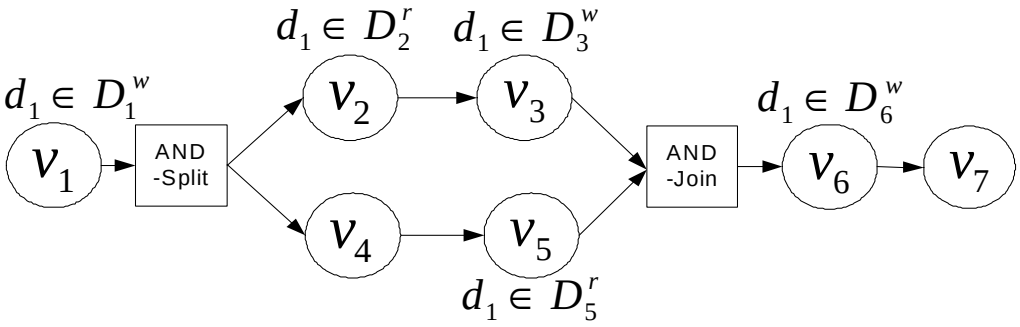

Figure 4.3: Workflow example

4.3.1 Key management

Two types of key pairs are introduced in our approach. Each vertex v_i is first associated with a policy pol_i defining the set of credentials a candidate partner needs to satisfy in order to be assigned to v_i. The policy pol_i is mapped to a key pair (PK_{pol_i}, SK_{pol_i}) where SK_{pol_i} is the policy private key and PK_{pol_i} the policy public key. Thus satisfying the policy pol_i means knowing the private key SK_{pol_i}, the inverse may however not be true depending on the policy private key distribution scheme as explained later on in section 4.5. The policy private key SK_{pol_i} can indeed be distributed by different means amongst which we distinguish three main types depicted in figure 4.2:

- **Key sharing**: a policy pol_i is associated with a single policy private key that is shared amongst principals satisfying pol_i. A simple key server KS_{pol_i} associated with pol_i can be used to distribute the policy private key SK_{pol_i} based on the compliance of business partners with the policy pol_i. In this case, the business partners satisfying pol_i thus share the same policy private key SK_{pol_i} associated with the encryption key PK_{pol_i}.

- **Policy-based cryptography**: a policy pol_i is expressed in a conjunctive-disjunctive form specifying the combinations of credentials ξ_k a principal is required to satisfy to be compliant with the policy:

  $$pol_i = \wedge_{i=1}^{m}[\vee_{j=1}^{m_i}[\wedge_{k=1}^{m_{i,j}}\xi_{i,j,k}]]$$

  where $\wedge$ represents a conjunction (AND) and $\vee$ a disjunction (OR). A cryptographic scheme [BM05] is used to map credentials to keys denoted credential keys SK_{ξ_k} that can be combined to encrypt, decrypt and sign messages based on a given policy. Some trusted authorities TA are in charge of distributing credential keys to requesters when the latter satisfies some assertions (that can be expressed in a conjunctive-disjunctive form e.g. (jobtittle=director)$\wedge$(company=xcorp)). This scheme provides direct mapping between a policy and some key material and thus eases policy management as opposed to key sharing. No anonymity-preserving traceability solution is however offered as principals satisfying a given assertion may possess the same credential key.

- **Group cryptography**: a policy pol_i is mapped to a group structure in which a group manager distributes different policy private keys to group members satisfying the policy pol_i. A single encryption key PK_{pol_i} is used to communicate with group members who however use their personal private key to decrypt and sign messages. This mechanism offers an identity traceability feature as only the group manager can retrieve the identity of a group member using a signature issued by the latter [ACJT00]. The policy private key of the business partner b_k is denoted $SK_{pol_i}^{b_k}$. We note GM_{pol_i} the group manager of the group whose members satisfy the policy pol_i. The management of policy key pairs is as complex as for the key server solution since a group structure is required for each specified policy.

Second, we introduce vertex key pairs $(PK_i, SK_i)_{i\in[1,n]}$ to protect the access to workflow data. We suggest a key distribution scheme wherein a business partner b_i whose identity is *a priori* unknown retrieves the vertex private key SK_i upon his assignment to the vertex v_i. Onion encryption techniques with policy public keys PK_{pol_i} are used to distribute vertex private keys. Furthermore, execution proofs have to be issued along with the workflow execution in order to ensure the compliance of the execution with the pre-defined plan. To that effect, we also leverage onion encryption techniques in order to build an onion structure with vertex private keys to assure the integrity of the workflow execution. The suggested key distribution scheme (O_d) and the execution proof mechanism (O_p) are depicted in figure 4.1 and specified later on in the paper.

In the sequel of the chapter, $\mathcal{M}$ denotes the message space, $\mathcal{C}$ the ciphertext space and $\mathcal{K}$ the key space. Using a key $K \in \mathcal{K}$ on a message $m \in \mathcal{M}$ is noted $\{m\}_K$ (e.g. encryption with a public key, signature with a private key) and h, h_1, h_2, h_3, h_4 denote one-way hash functions.

4.3.2 Data protection

The role of a business partner b_i assigned to a vertex v_i consists in processing the workflow data that are granted read-only and read-write access during the execution of v_i. We define a specific structure depicted in figure 4.4 called data block to protect workflow data accordingly. Each data block consists of two fields:

- the actual data: d_k
- a signature: $sign_a(d_k) = \{h(d_k)\}_{SK_a}$

We note $B_k^a = (d_k, sign_a(d_k))$ the data block including the data segment d_k that has last been modified during the execution of v_a. The data block B_k^a is also associated with a set of signatures denoted H_k^a that is computed by b_a assigned to v_a.

$$H_k^a = \left\{ \{h(\{B_k^a\}_{PK_l})\}_{SK_a} | l \in R_k^a \right\}$$

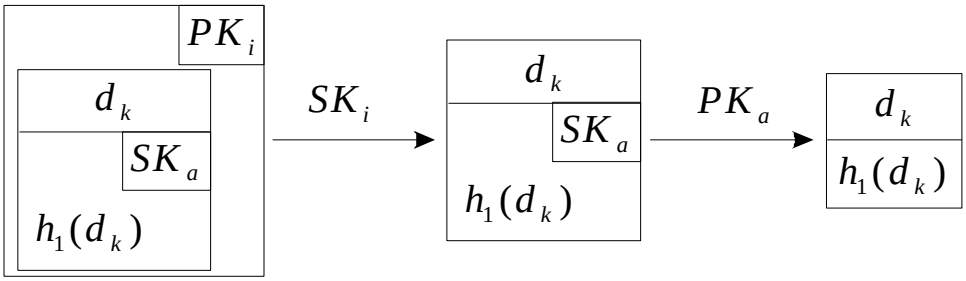

Figure 4.4: Access to workflow data

where R_k^a is the set defined as follows. Let $l \in [1, n]$.

$$l \in R_k^a \Leftrightarrow l \text{ satisfies } \begin{cases} d_k \in D_l^r \\ v_l \text{ is executed after } v_a \\ v_l \text{ is not executed after } v_{p_{(a,l,k)}} \end{cases}$$

where $v_{p_{(a,l,k)}}$ denotes the first vertex executed after v_a such that $d_k \in D^w_{p_{(a,l,k)}}$ and that is located on the same branch of the workflow as v_a and v_l. For instance, consider the example of figure 4.3 whereby d_1 is in D_1^w, D_2^r, D_3^w, D_5^r and D_6^w, $v_{(1,2,1)} = v_3$, $R_1^1 = \{2, 3, 5, 6\}$ and $R_1^3 = \{6\}$.

When the business partner b_i receives the data block B_k^a three cases can occur:

- b_i is granted read access on d_k: B_k^a is encrypted with PK_i and b_i decrypts the structure using SK_i in order to get access to d_k and $sign_a(d_k)$. b_i is then able to verify the integrity of d_k using PK_a, i.e. that d_k was last modified after the execution of v_a.
- b_i is granted write access on d_k: b_i can (in addition to what is possible in the first case since write access implies read access) update the value of d_k and compute $sign_i(d_k)$ yielding a new data block B_k^i and a new set H_k^i.
- b_i has no right on d_k: b_i receives B_k^a encrypted with PK_m (in this case v_m is executed after v_i) and he can only verify the integrity of $\{B_k^a\}_{PK_m}$ by matching $h(\{B_k^a\}_{PK_m})$ against the value contained in H_k^a.

The integrity and confidentiality of data access thus relies on the fact that the private key SK_i is made available to b_i only, prior to the execution of v_i. The corresponding distribution mechanism is presented in the next section.

4.3.3 Vertex private key distribution mechanism

The objective of the vertex private key distribution mechanism is to ensure that only the business partner b_i assigned to v_i at runtime and whose identity is *a priori* unknown can access the vertex

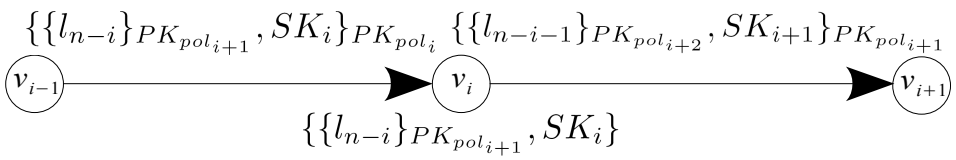

Figure 4.5: SEQUENCE pattern

private key SK_i. To that effect, the workflow structure in terms of execution patterns is mapped with an onion structure O_d so that at each step of the execution a layer of O_d is peeled off using SK_{pol_i} and SK_i is revealed.

Definition 4.1 (Onion Structure). Let X a set. An onion O is a multilayered structure composed of a set of n subsets of X $(l_k)_{k\in[1,n]}$, such that $\forall k \in [1,n]$ $l_k \subseteq l_{k+1}$. The elements of $(l_k)_{k\in[1,n]}$ are called layers of O, in particular, l_1 and l_n are the lowest and upper layers of O, respectively. We note $l_p(O)$ the layer p of an onion O.

Definition 4.2 (Onion wrapping). Let $A = (a_k)_{k\in[1,j]}$ and $B = (b_k)_{k\in[1,l]}$ two onion structures, A is said to be wrapped by B, when $\exists k \in [1,l]$ such that $a_j \subseteq b_k$.

We first present how vertex private keys are distributed to partners with respect to various workflow patterns including SEQUENCE, AND-SPLIT, AND-JOIN, OR-SPLIT and OR-JOIN before describing how those are combined in the execution of a complete workflow.

SEQUENCE workflow pattern

Vertex private keys are sequentially distributed to business partners. In this case, an onion structure assuring the distribution of vertex private keys is sequentially peeled off by business partners. Considering a sequence of n vertices $(v_i)_{i\in[1,n]}$, the business partner b_1 assigned to the vertex v_1 initiates the workflow execution with the onion structure O defined as follows.

$$O : \begin{cases} l_1 = \{SK_n\} \\ l_i = \left\{\{l_{i-1}\}_{PK_{pol_{n-i+2}}}, SK_{n-i+1}\right\} \text{ for } i \in [2,n] \\ l_{n+1} = \left\{\{l_n\}_{PK_{pol_1}}\right\} \end{cases}$$

The onion layers are iteratively wrapped to match the SEQUENCE pattern and the workflow execution proceeds as depicted in figure 4.5. For $i \in [2, n-1]$ the business partner b_i assigned to the vertex v_i receives $\{l_{n-i+1}(O)\}_{PK_{pol_i}}$, peels one layer off by decrypting it using SK_{pol_i}, reads $l_{n-i+1}(O)$ to retrieve SK_i and sends $\{l_{n-i}(O)\}_{PK_{pol_{i+1}}}$ to b_{i+1}.

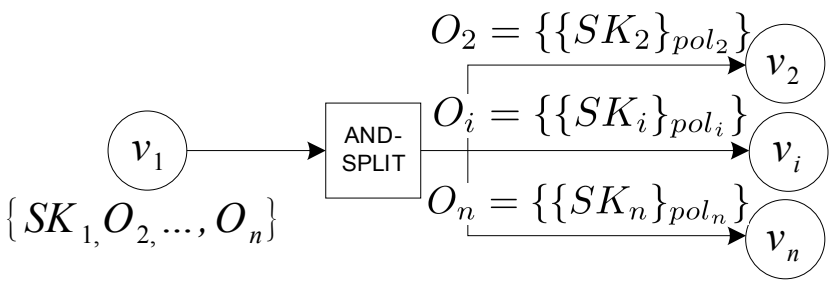

Figure 4.6: AND-SPLIT pattern

AND-SPLIT workflow pattern

In the case of the AND-SPLIT pattern, the business partners $(b_i)_{i\in[2,n]}$ assigned to the vertices $(v_i)_{i\in[2,n]}$ are contacted concurrently by b_1 assigned to the vertex v_1. In this case, $n-1$ vertex private keys should be delivered to $(b_i)_{i\in[2,n]}$ and the upper layer of the onion O_1 available to b_1 therefore wraps SK_1 and $n-1$ onions $(O_i)_{i\in[2,n]}$ to be sent to $(b_i)_{i\in[2,n]}$ as depicted in figure 4.6.

$$\begin{aligned} O_1 &= \{SK_1, O_2, O_3, .., O_n\} \\ O_i &= \left\{\{SK_i\}_{PK_{pol_i}}\right\} \text{ for } i \in [2, n] \end{aligned}$$

AND-JOIN workflow pattern

Since there is a single workflow initiator, the AND-JOIN pattern is preceded in the workflow by an AND-SPLIT pattern. In this case, the vertex v_n is executed by the business partner b_n if and only if the latter receives $n-1$ messages as depicted in figure 4.7. In order to meet this requirement, the vertex private key SK_n is divided into $n-1$ parts and defined by

$$SK_n = SK_{n_1} \oplus SK_{n_2} \oplus ... \oplus SK_{n_{n-1}}$$

The key SK_{n_i} is simply included in the onion O_i sent by b_i to b_n. Besides, in order to avoid redundancy, the onion structure λ associated with the sequel of the workflow execution right after v_n is only included in one of the onions received by b_n. Each $(b_i)_{i\in[1,n-1]}$ therefore sends the onion O_i defined as follows to b_n.

$$\begin{aligned} O_1 &= \{\{\lambda, SK_{n_1}\}_{PK_{pol_n}}\} \\ O_i &= \left\{\{SK_{n_i}\}_{PK_{pol_n}}\right\} \text{ for } i \in [2, n-1] \end{aligned}$$

OR-SPLIT workflow pattern

This is an exclusive choice and the business partner b_1 assigned to the vertex v_1 only needs to send one message.

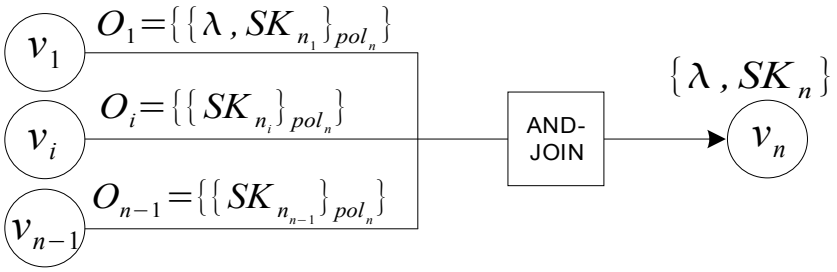

Figure 4.7: AND-JOIN pattern

$$O_1 = \{SK_1, O_2, O_3, .., O_n\}$$
$$O_i = \left\{\{SK_i\}_{PK_{pol_i}}\right\} \text{ for } i \in [2, n]$$

The onion O_1 is available to the business partner b_1. This is the same structure as the AND-SPLIT pattern, yet the latter only sends the appropriate onion O_i to the business partner assigned to the vertex v_i depending on the result of the condition associated with the OR-SPLIT pattern.

OR-JOIN workflow pattern

Since there is a single workflow initiator, the OR-JOIN is preceded in the workflow by an OR-SPLIT pattern. The business partner assigned to v_n receives in any cases a single message thus a single vertex private key is required that is sent by one of the business partners $(b_i)_{[1,n-1]}$ depending on the choice made at the previous OR-SPLIT in the workflow. the business partner b_n thus receives in any cases the onion O defined as follows.

$$O = \left\{\{\lambda, SK_n\}_{PK_{pol_n}}\right\}$$

where λ is an onion structure associated with the sequel of the workflow execution right after v_n.

Complete key distribution scheme

The procedure towards building an onion structure corresponding to the workflow structure can be implemented using for instance a breath first search algorithm starting from the last vertex of the workflow and wherein the workflow graph is read backward. This is rather straightforward and the procedure is only sketched throughout an example. Let's consider the workflow depicted in figure 4.3. The onion O_d enabling the vertex private key distribution during the execution of the workflow is defined as follows.

$$O_d = \{\{SK_1, \underbrace{\{SK_2, \{SK_3, \{SK_{6_1}, \overbrace{\{SK_7\}_{PK_{pol_7}}}^{\text{Sequel after } v_6}\}_{PK_{pol_6}}\}_{PK_{pol_3}}\}_{PK_{pol_2}}}_{\text{First AND-SPLIT branch}},$$
$$\underbrace{\{SK_4, \{SK_5, \{SK_{6_2}\}_{PK_{pol_6}}\}_{PK_{pol_5}}\}_{PK_{pol_4}}}_{\text{Second AND-SPLIT branch}}\}_{PK_{pol_1}}\}$$

The onions associated with the two branches forming the AND-SPLIT pattern are wrapped by the layer corresponding to the vertex v_1. Only the first AND-SPLIT branch includes the sequel of the workflow after v_6. The overall structure of the onion is of course based on the SEQUENCE pattern.

4.3.4 Execution proofs and traceability

Along with the workflow execution, an onion structure O_{p_i} is built at each execution step i with vertex private keys in order to allow business partners to verify the integrity of the workflow execution and optionally to gather anonymity-preserving traces when traceability is required during the execution of a workflow. Based on the properties we introduced in section 4.3.1, group cryptography is the only mechanism that meets the needs of the policy private key distribution when identity traceability is needed. In that case, we define for a workflow instance, the workflow arbitrator role that is assumed by a trusted third party.

Definition 4.3 (Workflow arbitrator). The workflow arbitrator, denoted W_{ar}, is a trusted third party able to disclose business partners' identity in case of dispute. The workflow arbitrator is contacted to revoke the anonymity of some business partners only in case of dispute, this is an optimistic mechanism.

The onion structure O_p is initialized by the business partner b_1 assigned to v_1 who computes $O_{p_1} = \left\{\{h(P_W)\}_{SK_{pol_1}}\right\}$ where P_W is called workflow policy and is defined as follows.

Definition 4.4 (Workflow Policy). The workflow specification S_W denotes the set S_W defined by $S_W = \{W, (J_i^r, J_i^w, pol_i)_{i\in[1,n]}, h\}$ where

$$J_i^r = \{k \in [1, j] | d_k \in D_i^r\} \text{ and } J_i^w = \{k \in [1, j] | d_k \in D_i^w\}$$

The sets J_i^r and J_i^w basically specify for each vertex the set of data that are granted read-only and read-write access, respectively. S_W is defined at workflow design phase.

The workflow policy P_W denotes the set defined by:

$$P_W = S_W \cup \{W_{iid}, W_{ar}, h_1, h_2, h_3, h_4\} \cup \{PK_i | i \in [1, n]\}$$

P_W is a public parameter computed by the workflow initiator b_1 and that is available to the business partners involved in the execution of W.

The onion structure O_p is initialized this way so that it cannot be replayed as it is defined for a specific instance of a workflow specification. If traceability is required during the execution of some business processes, the signatures of business partners with policy private keys are collected during the building process of O_p so that anonymity can be later on revoked in case of dispute. Group encryption is used in this case to distribute policy private keys and the business partner b_1 is in charge of contacting a trusted third party when the workflow is instantiated, sending it $(h(P_W), P_W)$ to play the role of workflow arbitrator for the instance.

At the step i of the workflow execution, b_i receives $O_{p_{i-1}}$ and encrypts its upper layer with SK_i to build an onion O_{p_i} which he sends to b_{i+1} upon completion of v_i. If traceability is required, b_i signs $\{O_{p_{i-1}}, \{h(P_W)\}_{SK_{pol_i}^{b_i}}\}$ with SK_i instead. Considering a set $(v_i)_{[1,n]}$ of vertices executed in sequence assigned to the business partners $(b_i)_{[1,n]}$ and assuming that traceability is needed (i.e. group cryptography is used) we get:

$$
\begin{aligned}
O_{p_1} &= \left\{\{h(P_W)\}_{SK_{pol_1}^{b_1}}\right\} \\
O_{p_2} &= \left\{\{O_{p_1}, \{h(P_W)\}_{SK_{pol_2}^{b_2}}\}_{SK_2}\right\} \\
O_{p_i} &= \left\{\{O_{p_{i-1}}, \{h(P_W)\}_{SK_{pol_i}^{b_i}}\}_{SK_i}\right\} \text{ for } i \in [3, n]
\end{aligned}
$$

The building process of O_{p_i} is based on workflow execution patterns ; yet since it is built at runtime contrary to the onion O_d, this is straightforward:

- There is no specific rule for OR-SPLIT and OR-JOIN patterns as during the workflow execution, this will result in the execution of a single branch (exclusive choice),
- When encountering an AND-SPLIT pattern, the same structure O_{p_i} is concurrently sent while in case of an AND-JOIN, the $n-1$ onions received by a partner b_n are wrapped by a single structure: $O_{p_n} = \left\{\{O_{p_1}, O_{p_2}, .., O_{p_{n-1}}, \{h(P_W)\}_{SK_{pol_n}^{b_n}}\}_{SK_n}\right\}$.

Considering the example depicted in figure 4.3 and assuming traceability is not required, at the end of the workflow execution the onion O_p is defined as follows.

$$
O_p = \{\{\underbrace{\{\{\{\{h(P_W)\}_{SK_{pol_1}}\}_{SK_2}\}_{SK_3}}_{\text{First AND-SPLIT branch}}, \underbrace{\{\{\{h(P_W)\}_{SK_{pol_1}}\}_{SK_4}\}_{SK_5}\}}_{\text{Second AND-SPLIT branch}}\}_{SK_6}\}_{SK_7}\}
$$

$\{h(P_W)\}_{SK_{pol_1}}$ is sent by b_1 assigned to v_1 to both b_2 and b_4 assigned to v_2 and v_4, respectively. The onion structure associated with the two branches forming the AND-SPLIT pattern thus includes $\{h(P_W)\}_{SK_{pol_1}}$ twice. The overall structure is mapped to the SEQUENCE pattern and layers are iteratively wrapped by a new layer as the workflow instance proceeds further.

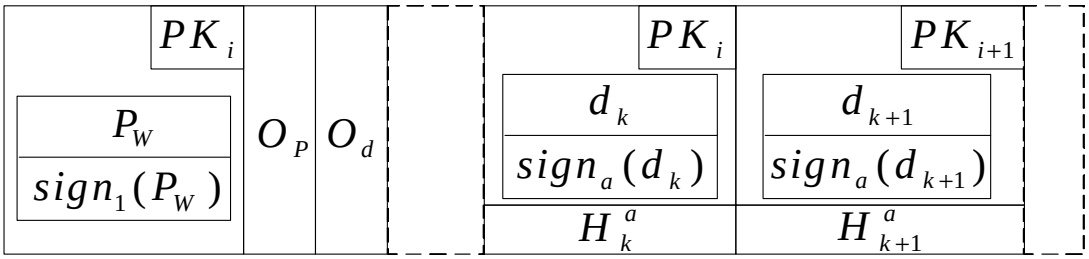

Figure 4.8: Workflow message structure

In order to verify that the workflow execution is compliant with the pre-defined plan when he starts the execution of the vertex v_i, the business partner b_i assigned to v_i just peels off the layers of $O_{p_{i-1}}$ using the vertex public keys of the vertices previously executed based on S_W. Doing so he retrieves the value $\{h(P_W)\}_{SK_{pol_1}}$ that should be equal to the one he can compute given P_W, if the workflow execution has been so far executed according to the plan. In case traceability is required by the execution, b_i also verifies the signatures of the business partners assigned to the vertices $(v_{j_p})_{p\in[1,k_i]}$ executed right before him i.e. b_i verifies $\{h(P_W)\}_{SK_{pol_p}^{b_p}}$ for all $p \in [1, k_i]$. If b_i detects that a signature is missing he contacts the workflow arbitrator W_{ar} to declare the workflow instance inconsistent. In fact, business partners are in charge of contacting the workflow arbitrator when a signature is not valid and those who do not declare corrupted signatures are held responsible in place of partners whose signature is missing. In case of conflict on the outcome of some workflow tasks, the onion O_p is sent to the workflow arbitrator who is able to retrieve the signatures with policy private key of the stakeholders using P_W and with the help of some group managers the corresponding identities.

4.3.5 Vertex key pair generation

Vertex key pairs have to be defined for a single instance of a workflow specification in order to avoid replay attacks. To that effect, we capitalize on identity-based encryption (IBE) techniques [BF01] in the specification of the set $(PK_i, SK_i)_{i\in[1,n]}$. We start with a short description of the main concepts of the IBE cryptosystem before specifying the vertex key pairs. In what follows, $(\mathbb{G}_1, +)$ and $(\mathbb{G}_2, .)$ denote two groups of order q for a prime q.

Definition 4.5 (Admissible bilinear map)**.** A map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$ is an admissible bilinear map iff $\hat{e}$ satisfies the following properties:

- **Bilinear**: for $P, Q \in \mathbb{G}_1$ and for $a, b \in \mathbb{Z}_q^*$, $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$
- **Non-degenerate**: for P generator of $\mathbb{G}_1$, $e(P, P) \neq 1$ and therefore $\hat{e}(P, P)$ is a generator of $\mathbb{G}_2$
- **Computable**: there exists an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}_1$

The IBE cryptosystem [BF01] consists of the following four algorithms:

Setup : Given a security parameter $k \in \mathbb{Z}^+$,

Step 1: Generate a tuple $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e})$ using a Bilinear Diffie-Hellman parameter generator [BF01] and pick P a random generator of $\mathbb{G}_1$. The Bilinear Diffie-Hellman parameter generator ensures that the generated tuple $(q, \mathbb{G}_1, \mathbb{G}_2, \hat{e})$ is such that the BDH-assumption [BF01] is valid.

Step 2: Pick a random $s \in \mathbb{Z}_q^*$ and set $P_{pub} = sP$

Step 3: Define two hash functions: $h_1 : \{0,1\}^* \longrightarrow \mathbb{G}_1^*$ and $h_2 : \mathbb{G}_2 \longrightarrow \{0,1\}^n$ for some $n \in \mathbb{N}^*$

Step 4: Set $\mathcal{P} = (q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, n, h_1, h_2)$ to be the system public parameters

Extract : For a given string $ID \in \{0,1\}^*$,

Step 1: Compute $Q_{ID} = h_1(ID) \in \mathbb{G}_1^*$

Step 2: Set the private key $d_{ID} = sQ_{ID}$ where s is called master key

Encrypt : In order to encrypt a message M with the public key ID,

Step 1: Compute $Q_{ID} = h_1(ID)$

Step 2: Define a random $r \in \mathbb{Z}_q^*$

Step 3: Set the ciphertext to be $C = \langle rP, M \oplus h_2(g_{ID}^r) \rangle$ where $g_{ID}^r = \hat{e}(Q_{ID}, P_{pub}) \in \mathbb{G}_2^*$

Decrypt : Let $C = \langle U, V \rangle$ be a ciphertext encrypted using the public key ID. In order to decrypt C using the private key $d_{ID} \in \mathbb{G}_1^*$,

Step 1: Compute $V \oplus h_2(\hat{e}(d_{ID}, U)) = M$

We define for all $i \in [1, n], ID_i = W_{iid} \oplus S_W \oplus v_i$. Using the notations defined in the description of the IBE cryptosystem, the vertex key pair (PK_i, SK_i) is specified as follows.

$$\begin{cases} PK_i = h_3(W_{iid} \oplus S_W \oplus v_i) \\ SK_i = sh_1(PK_i) \end{cases}$$

where h_3 is a hash function defined by: $h_3 : \{0,1\}^* \longrightarrow \{0,1\}^n$ for some $n \in \mathbb{N}^*$

The master key s is held by the vertex private key generator who is in our case the workflow initiator. The signature scheme proposed in [Pat02] can be used to compute the ID-based signatures required by the mechanisms we proposed. Having run the Setup algorithm of the IBE cryptosystem, the signature scheme consists of the two following algorithms:

Sign : In order to sign a message M with the vertex private key SK_i,

Step 1 : Pick a random $k \in \mathbb{Z}_q^*$ and define a hash function $h_4 : \{0,1\}^* \longrightarrow \mathbb{Z}_q^*$

Step 2 : Set the signature to be $\{M\}_{SK_i} = \langle kP, k^{-1}(h_4(M)P + SK_i)\rangle$

Verify : Let $sign = \langle U, V\rangle$ be a purported signature on a message M issued using the private key SK_i. To verify the validity of $sign$,

Step 1 : Compare $\hat{e} = (U, V)$ to the value $\hat{e}(P, P)^{h_4(M)}.\hat{e}(P_{pub}, PK_i)$

Step 2 : Return valid if the values match, invalid otherwise

Using this vertex key pair specification, the public parameters $\mathcal{P}$ of the IBE cryptosystem and the hash functions h_3 and h_4 should be of course included in P_W. This vertex key pair specification has a double advantage. First vertex key pairs cannot be reused during any other workflow instance and second vertex public keys can be directly retrieved from W and W_{iid} when verifying the integrity of workflow data or peeling off the onion O_p.

4.3.6 Communication protocol

In order to support a coherent execution of the mechanisms presented so far, workflow messages exchanged between business partners consist of the set of information that is depicted in figure 4.8.

- **Workflow data**: the set $(d_k)_{k\in[1,j]}$ of workflow data is transported between business partners and each piece of data satisfy the data block specification. A single message may include several copies of the same data block structure that are encrypted with different vertex public keys based on the execution plan. This can be the case with AND-SPLIT patterns. Besides, workflow data can be stored in two different ways depending on the requirements for the execution. Either we keep the iterations of data resulting from each modification in workflow messages till the end of the execution or we simply replace data content upon completion of a vertex. The bandwidth requirements are higher in the first case since the size of messages increases as the workflow execution proceeds further.
- P_W: P_W is required to retrieve vertex and policy public keys and specifies the workflow execution plan.
- O_d and O_p: the two onion structures O_d and O_p are also included in the message.

Upon receipt of the message depicted in figure 4.8 a business partner b_i assigned to v_i retrieves first the vertex private key from O_d. He then checks that P_W is genuine i.e. that it was initialized by the business partner initiator of the workflow assigned to v_1. He is later on able to verify the compliance of the workflow execution with the plan using O_p and the integrity of workflow data. Finally he can process workflow data.

4.4 Secure execution of decentralized workflows

In this section we specify how the mechanisms presented so far in this chapter are combined to support the secure execution of a workflow in the decentralized setting. After an overview of the execution steps, the secure workflow execution is described in terms of the workflow initiation and runtime specifications.

4.4.1 Execution process overview

Integrating security mechanisms to enforce the security requirements identified for the pervasive workflow architecture and decentralized workflow execution in general requires a process strongly coupled with both workflow design and runtime specifications. At the workflow design phase, the workflow specification S_W is defined in order to specify for each vertex the sets of data that are accessible in read and write access and the credentials required by potential business partners to be assigned to workflow vertices. At workflow initiation phase, the workflow policy P_W is specified and the onion O_d is built. The workflow initiator builds then the first set of workflow messages to be sent to the next partners involved. This message generation process consists of the initialization of the data blocks and that of the onion O_p.

At runtime, a business partner b_i chosen to execute a vertex v_i receives a set of workflow messages. Those messages are processed to retrieve SK_i from the onion O_d and to access workflow data. Once the vertex execution is complete b_i builds a set of workflow messages to be dispatched to the next partners involved in the execution. In this message building process, the data and the onion O_p are updated.

The set of functional operations composing the workflow initiation and runtime specifications is precisely specified later on in this section. In what follows N_k^i denotes the set defined as follows. Let $l \in [1, n]$

$$l \in [1,n] \Leftrightarrow l \text{ satisfies } \begin{cases} d_k \in D_l^r \\ v_l \text{ is executed right after } v_i \end{cases}$$

Consider the example of figure 4.3: d_1 is accessed during the execution of the vertices v_1, v_2 and v_5 thus $N_1^1 = \{2, 5\}$.

4.4.2 Workflow initiation

The workflow is initiated by the business partner b_1 assigned to the vertex v_1 who issues the first set of workflow messages $(M_{1 \rightarrow j_p})_{p \in [1, z_1]}$. The workflow initiation mainly consists of the

following steps.

Workflow initiation :

Step 1 : Workflow policy specification: generate $(PK_i, SK_i)_{i\in[1,n]}$ and assign W_{ar}

Step 2 : Initialization of the onion O_d

Step 3 : Data block initialization: compute $\forall k \in [1, j]\ sign_1(d_k)$

Step 4 : compute N_k^1 and $R_k^1\ \forall k \in [1, j]$

Step 5 : Data block encryption: compute $\forall k \in [1, j], \forall l \in N_k^1\ \{B_k^1\}_{PK_l}$

Step 6 : Data block hash sets: compute $\forall k \in [1, j], \forall l \in R_k^1\ \{h(\{B_k^1\}_{PK_l})\}_{SK_1}$

Step 7 : Initialization of the onion O_p: compute O_{p_1}

Step 8 : Message generation based on W and $(N_k^1)_{k\in[1,j]}$

The steps one and two are presented in sections 4.3.5 and 4.3.3, respectively. The workflow messages are generated with respect to the specification defined in figure 4.8 and sent to the next business partners involved. This includes the initialization of the onion O_p and that of data blocks which are encrypted with appropriate vertex public keys.

4.4.3 Workflow message processing

A business partner b_i being assigned to a vertex v_i proceeds as follows upon receipt of the set of workflow messages $(M_{j_p\rightarrow i})_{p\in[1,k_i]}$ sent by the k_i business partners assigned to the vertices $(v_{j_p})_{p\in[1,k_i]}$ executed right before v_i.

Workflow message processing :

Step 1 : Retrieve SK_i from O_d

Step 2 : Data block decryption with SK_i based on J_i^r

Step 3 : Execution proof verification: peel off the onion O_p

Step 4 : Data integrity check based on W and P_W

Step 5 : Vertex execution

Step 6 : compute $N_k^i\ \forall k \in J_i^r$ and $R_k^i\ \forall k \in J_i^w$

Step 7 : Data block update: compute $\forall k \in J_i^w\ sign_i(d_k)$ and update d_k content

Step 8 : Data block encryption: compute $\forall k \in J_i^r, \forall l \in N_k^i\ \{B_k^i\}_{PK_l}$

Step 9 : Data block hash sets: compute $\forall k \in J_i^w, \forall l \in R_k^i\ \{h(\{B_k^i\}_{PK_l})\}_{SK_i}$

Step 10 : Onion O_p update: compute O_{p_i}

Step 11 : Message generation based on W and $(N_k^i)_{k\in[1,j]}$

After having retrieved SK_1 from O_d, b_i verifies the integrity of workflow data and that the execution of the workflow up to his vertex is consistent with the onion O_p. Workflow data are then processed during the execution of v_i and data blocks are updated and encrypted upon completion. Finally b_i computes O_{p_i} and issues the set of workflow messages $(M_{i\rightarrow j_p})_{p\in[1,z_i]}$ to the intended business partners.

4.5 Security analysis

The parameters that are relevant to the security properties offered by the mechanisms presented in this chapter are mainly twofold. First, there are several alternatives with respect to the management of the key pair (PK_{pol_i}, SK_{pol_i}), including simple key distribution based on the policy compliance, group key management or policy-based cryptography, on which the security properties verified by our solution depend. In fact, the main difference between the three policy private key distribution schemes we identified comes from the number of business partners sharing the same policy private key. As a matter of fact, the more partners share a given private key the easier it is for some unauthorized peer to get this private key and get access to protected data. Besides, the trustworthiness of business partners can not be controlled, especially when it comes to sharing workflow data with unauthorized peers once the vertex private key has been retrieved. In this context, the mechanisms presented in this chapter verify some properties that do not depend on the underpinning policy private key distribution scheme while some other do. In the security evaluation of our solution, we make two assumptions:

- **Security of policy keys**: the public key encryption scheme used in the specification of the policy key pair (PK_{pol_i}, SK_{pol_i}) is semantically secure against a chosen ciphertext attack and the associated signature scheme achieves signature unforgeability.
- **Security of vertex keys**: the public key encryption scheme used in the specification of the vertex key pair (PK_i, SK_i) is semantically secure against a chosen ciphertext attack and the associated signature scheme achieves signature unforgeability.

4.5.1 Inherent security properties

Theorem 4.1 (Integrity of execution). *Vertex private keys are retrieved by business partners knowing policy private keys associated with the policies specified in the workflow.*

Assuming in addition that business partners do not share vertex private keys, the integrity of the distributed workflow execution is assured i.e. workflow data are accessed and modified based on the pre-defined plan specified by means of the sets J_i^r and J_i^w.

Proof. This property is ensured by the onion O_d which assures distribution of the vertex keys used for accessing workflow data based on the workflow execution plan.

Assuming that a workflow initiator builds O_d based on the methodology specified in 4.3.3 and under the policy key security assumption, we claim that it is not feasible for an adversary $\mathcal{A}$ to extract the vertex private key SK_i from O_d if $\mathcal{A}$ does not know the set of policy private keys $(SK_{pol_{i_k}})_{k\in[1,l]}$ associated with the set of vertices $(v_{i_k})_{k\in[1,l]}$ executed prior to v_i in W. This is true as the structure of O_d is mapped to W. ■

Theorem 4.2 (**Resilience to instance forging**). *Upon receipt of a workflow message, a business partner is sure that a set of business partners knowing policy private keys associated with the policies specified in the workflow have been assigned to the vertices executed so far provided that he trusts the business partners satisfying the policy* pol_1.

Proof. This property is enforced by the onion O_p whose building process is based on the workflow structure and vertex private keys. As stated in the previous theorem, vertex private keys can only be retrieved by business partners knowing some policy private keys. We also claim that an adversary that does not verify a policy can not forge a workflow instance, i.e. that the adversary can not produce a workflow message pertaining to a valid workflow instance.

Assuming that a workflow initiator builds O_p based on the methodology specified in 4.3.4 and under the policy key security assumption, we claim that the onion structure O_p is unforgeable. The unforgeability property relies on two further properties:

1. a genuine onion structure O_p built during a previous instance of a workflow can not be replayed ;
2. an onion structure O_p can not be built by an adversary that is not trusted by business partners.

The first property is enforced by the fact that an onion structure O_p properly built is bound to a specific workflow policy P_W and thus can not be reused during an attempt to execute a malicious workflow instance. The second property is straightforward under the policy key security assumption as the policy-based signature scheme achieves signature unforgeability. Thus an adversary can not produce a valid onion $O_{p_1} = \left\{\{h(P_W)\}_{SK_{pol_1}}\right\}$. ■

Theorem 4.3 (**Data Integrity**). *Assuming that business partners do not share vertex private keys they retrieve from the onion* O_d*, our solution achieves the following data integrity properties:*

- *Data truncation and insertion resilience: any business partner can detect the deletion or the insertion of a piece of data in a workflow message*
- *Data content integrity: any business partner can detect the integrity violation of a data block content in a workflow message*

Proof. The first property is ensured as the set of workflow data blocks that should be present in a workflow message is specified in P_W, the workflow message formatting has thus to be compliant with the workflow specification. The second property is assured by the fact that an adversary can not modify a given data block without providing a valid signature on this data block. This property relies on the unforgeability of the signature scheme used in the data block and hash set specifications. ■

These three security properties are sufficient to enable a coherent and secure execution of distributed workflows provided that business partners are trustworthy and do not share their policy or vertex private keys. The latter assumption is in fact hard to assess when sensitive information are manipulated during the workflow. We therefore introduced the traceability mechanism to meet the requirements of sensitive workflow executions.

4.5.2 Revocation of a business partner anonymity

The main flaw of the basic security mechanisms we outlined is that the involvement of business partners in a workflow can remain anonymous thus preventing the detection of potential malicious peers who somehow got access to some policy private keys. To overcome this limitation when required, traceability with group cryptography has to be used during the execution of a business process. In this case the anonymity revocation mechanism provided with group cryptography can be seen as a penalty for business partners thus preventing potential malicious behaviors such as vertex private key sharing with unauthorized peers. Besides, policy private keys distributed by a group manager are intended for individual use which makes key leakage highly unlikely.

The following theorems hold when the policy private key distribution scheme is based on group encryption techniques and traceability is required in the execution of workflows. As corollary of this assumption, we assume that vertex private keys are not shared with unauthorized peers, theorem 4.3 is thus verified.

Theorem 4.4 (**Integrity of execution**). *The integrity of the distributed workflow execution is ensured or the workflow instance is declared inconsistent by the selected workflow arbitrator. Integrity of the distributed workflow execution consists in this case in performing the following tasks:*

- *workflow data are accessed and modified based on the pre-defined plan specified by means of the sets J_i^r and J_i^w ;*
- *signatures with policy private key are stored by the business partners involved in the workflow execution.*

Proof. Anonymity revocation is here a means to force business partners to behave properly during the execution of a workflow. If any malicious business partner is involved, he will

not store his signature and we claim that the workflow instance will no longer be a valid one. The mechanism we proposed for anonymity revocation is as we mentioned optimistic and four scenarios can actually occur:

- A business partner detects that a signature is missing during the course of the workflow execution
- Each business partner stored his signature
- A set of business partners did not store their signature while some other partners did not declare the missing signatures to the workflow arbitrator
- A set of business partners assigned to vertices contiguously executed till the end of the workflow did not store their signature

In the first case, the workflow instance will be declared inconsistent by the workflow arbitrator. In the second case, trustworthy business partners have been involved in the workflow and their identity can be easily traced back by the workflow arbitrator. In the third case which is in fact highly unlikely to occur, the business partners who have not declared the missing signatures become responsible in place of the business partners who cheated. In the last case, nobody can be held responsible as apparently a group of untrustworthy business partners was involved in a fraud attempt and the workflow instance is declared inconsistent. ■

4.5.3 Discussion

As mentioned in the security analysis, group cryptography associated with anonymity revocation provides a full-fledged solution that meets the requirements of sensitive workflow instances. The other policy private key distribution schemes can be in fact used when the workflow execution is not sensitive or the partners satisfying the policies required by the workflow are deemed trustworthy. Our solution can still be optimized to avoid the replication of workflow messages. A business partner may indeed send the same workflow message several times to different partners satisfying the same security policy resulting in concurrent executions of a given workflow instance. Multiple instances can be detected by the workflow arbitrator when traceability is required or a solution based on a stateful service discovery mechanism can be also envisioned to solve this problem.

4.6 Integration within the transactional protocol

In this section we discuss the possible integration of the security mechanisms specified in this chapter within the transactional protocol presented in chapter 3. We thus assume in what follows

that the term pervasive workflow instance refers to the execution of a pervasive workflow that is supported by the transactional protocol presented in chapter 3 and that implements the security mechanisms outlined in this chapter.

There are various types of security faults that can be raised during the execution of the security mechanisms we have just specified and that need to be handled by the coordination layer so that a pervasive workflow instance can recover when these security faults are caught. In the fashion of the ATS model a failure recovery strategy has to be defined for security faults. In a first approach, one could regard security faults as failures to execute and directly integrate them into the transactional protocol execution. This approach would however lead to consider a strict atomicity within a workflow instance as the security mechanisms we specified are executed by all the business partners involved in a workflow instance thus making the latter prone to failures. As a consequence, no business partner would in this case claim that he verifies the retriability property. This approach is as a result not suited to meet the requirements that are relevant to assuring consistency of a pervasive workflow instance with respect to relaxed atomicity constraints. We thus choose to define security-fault handling mechanisms that rely on the workflow arbitrator and thus group cryptography techniques introduced in section 4.3.4, in order to manage the recovery procedure when security faults occur. In this case, the failure recovery strategy associated with the complete failure of a workflow instance due to a security fault consists in penalizing the business partner that caused the fault. The specification of possible penalties that can be issued to business partner is out of the scope of this thesis but one could think to legal compensations.

The security-fault handling mechanisms that we designed are integrated into the ones defined in the context of transactional failures in order to first recover from security faults that do not lead to the complete failure of a workflow instance without canceling the workflow instance as well as penalize business partners that may cause security faults from which it is not possible to recover. This approach not only enables the detection of security faults but also their recovery whenever possible while preserving relaxed atomicity constraints identified in chapter 3. Security solutions are indeed often only considered a means to detect inconsistencies within the execution of applications but the way to properly handle these inconsistencies is left aside. On the contrary, we consider in this work these two aspects, security inconsistencies do not indeed always imply complete failure within the pervasive workflow architecture.

We first describe the different types of security faults that can be raised during the security mechanism execution before specifying the recovery mechanisms designed to handle the latter.

4.6.1 Security faults

There are six main types of security faults that can be encountered during the execution of a pervasive workflow instance as follows.

- **Data decryption fault**: a business partner is not able to decrypt a piece of data that

he is allowed to access based on the workflow specification during the execution of the vertex to which he is assigned. The data decryption fault can be raised at anytime during a workflow instance and is forwarded to the coordination layer as it keeps a business partner from carrying out a vertex execution.

- **Data integrity fault**: a business partner detects that a piece of data has been altered in the workflow message he has just received. The data integrity fault can be raised at anytime during a workflow instance and is forwarded to the coordination layer as it keeps a business partner from carrying out a vertex execution.

- **Vertex private key retrieval fault**: a business partner can not retrieve a vertex private key from the onion O_d. The vertex private key retrieval fault can be raised at anytime during a workflow instance and is forwarded to the coordination layer as it keeps a business partner from carrying out a vertex execution.

- **Proof of execution fault**: a business partner can not assess the validity of the onion O_p i.e. that all signatures are valid. The proof of execution fault can be raised at anytime during a workflow instance and is forwarded to the coordination layer as it keeps a business partner from carrying out a vertex execution.

- **Discovery fault**: no candidate business partner satisfying the policy associated with a vertex can be found. The discovery fault can only be raised during the replacement of business partners of type b_k^v that are not retriable or these of type b_k^m that are not reliable since the transactional protocol execution requires source discovery mode. This fault is in fact equivalent to the failure of the associated vertex and of course is critical enough to be forwarded to the coordination layer, we however consider it a transactional failure rather than a security fault.

- **Encryption, decryption or signature fault**: an error occurred during the encryption, the decryption or the signature process of some data that is not due to an invalid signature or a key material issue. The "Encryption, decryption or signature fault" only refers to computational or accidental faults that may occur during the operation execution and we consider that these operations are retriable. This fault is therefore not forwarded to the coordination layer.

These various types of faults are basically derived from the sequence of operations specified in section 4.4.3 and that is executed by all business partners involved in a pervasive workflow instance. The four types of security faults that are forwarded to the coordination layer, can be in some cases handled without having to declare the complete failure of a workflow instance if these faults occurred accidentally. The workflow execution can indeed recover using a simple backup of corrupted workflow messages since the operations that can raise these faults are executed prior to any workflow data processing. When however it is not possible to recover from them using basic fault handling mechanisms, the workflow arbitrator has to be contacted. Our goal towards designing appropriate security-fault handling mechanisms therefore consists in the enforcement of the following property:

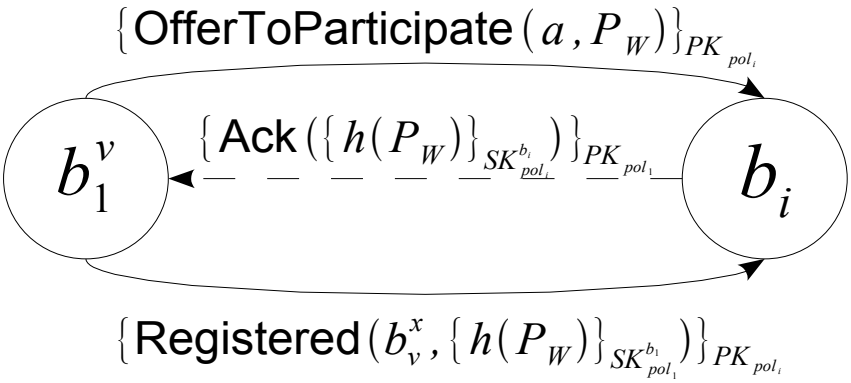

Figure 4.9: Business partner registration when security mechanisms are used

(P_3) Should a security fault from which it is not possible to recover occur, the identity of the business partner that made an error or behaved maliciously can be traced back

The design of the corresponding security-fault handling mechanism that is presented next thus assumes that the policy private key distribution scheme implemented is group cryptography so that business partners' identities can be easily traced back using the mechanisms specified in this chapter.

4.6.2 Business partner registration

During the course of the normal business partner registration phase that takes place within the transactional protocol execution, we add a security handshake wherein the business partner selected to execute a given vertex transmits to the critical zone initiator his signature with policy private key of the workflow policy. This handshake can be seen as a commitment that the business partner will behave correctly during the workflow instance. The business partner registration is depicted in figure 4.9. The critical zone initiator contacts a candidate business partner asking him whether he agrees to commit to execute the operation a of the workflow whose workflow policy is P_W. In case he accepts and this means that he trusts the business partner initiator of the critical zone who satisfies the policy pol_1, the candidate business partner acknowledges with a signature on the workflow policy with his policy private key $SK^{b_i}_{pol_i}$. Once the newly assigned business partner's coordinator for the transactional protocol execution is known, b_1^v sends the information and acknowledges the registration with a signature on P_W with his policy private key. The following registration mechanism assumes as for the basic security mechanisms that the selected business partner trusts business partners satisfying the policy associated with the first vertex of a critical zone.

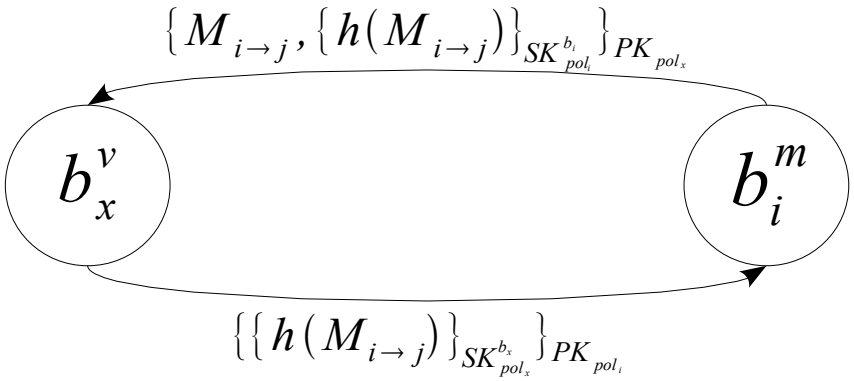

Figure 4.10: Workflow message backup when security mechanisms are used

4.6.3 Workflow message backup process

The workflow data backup procedure implemented by the basic transactional protocol is also modified in order to handle the security faults we identified. In this case, we store the workflow message that has been issued by a business partner of type b_k^m to the next business partner involved in the workflow instance. The workflow message backup copy is still handled by the business partner b_x^v most recently executed. The workflow message backup procedure is depicted in figure 4.10. The business partner b_i^m is in charge of assessing the validity of the message that he sends to the business partner b_x^v most recently executed, this is why he signs the message he sends with his policy private key $SK_{pol_i}^{b_i}$. The business partner b_x^v acknowledges the receipt of the workflow message signing it with his policy private key $SK_{pol_x}^{b_x}$. Of course at each step of the procedure both business partners verify the validity of the signature provided by the other and may ask for a new transmission if they do not match.

4.6.4 Recovering from security-faults

Security faults which occur accidentally, such as these that result from transmission errors and the like, can be simply recovered using backup workflow messages that are stored by the business partners of type b_x^v and that should be valid. It is indeed the responsibility of the business partner that backed up the workflow message that should be retransmitted to ensure the validity of the latter. This statement is actually enforced by the signature provided by business partners during the message backup procedure. If backup workflow messages are valid, there exists a restoration point in the workflow execution such that the execution is still consistent from both transactional and security perspectives so that the worfklow execution can be restarted from that point if required. If the backup workflow message is however not valid, the recovery procedure fails and the workflow instance arbitrator is contacted to mediate the case as specified in the next section.

4.6.5 Handling security-faults when the recovery procedure fails

When security faults can not be recovered after several retransmissions of a backup workflow message, the mediation of the workflow instance arbitrator is required. The reasons why the recovery procedure fails are in fact not limited to malicious behaviors and our procedure to handle recovery failures does not depend on these various reasons as our primary goal is the anonymity revocation of the business partner that caused the fault. There are mainly four situations that require the mediation of the workflow arbitrator:

- **Failure to recover from a "Data decryption fault"**: based on the signatures and workflow messages sent by the two business partners that were involved in the backup procedure of the corrupted workflow message, the workflow arbitrator determines whether the business partner of type b_x^v that stored this message has caused the corruption or whether the other business partner backed up a message wherein a piece of data was not properly encrypted with the appropriate vertex public key. Based on the outcome of the mediation, the workflow arbitrator can either ask for a new generation of the corrupted workflow message with workflow data correctly encrypted or declare the workflow instance inconsistent and penalize the business partner who caused the fault.
- **Failure to recover from a "Vertex private key retrieval fault"**: based on the signatures and workflow messages sent by the two business partners that were involved in the backup procedure of the corrupted workflow message, the workflow arbitrator determines whether the business partner of type b_x^v that stored this message has caused the corruption, whether the other business partner backed up a message that was corrupted or whether the workflow initiator generated an onion O_d that was corrupted in the first place. Based on the outcome of the mediation, the workflow arbitrator can either ask for a retransmission of the corrupted workflow message should a valid copy of it be available, ask the workflow initiator to generate a new onion O_d corresponding to the current state of the workflow execution (i.e. the upper layer of this onion is associated with the vertex during the execution of which the fault was raised) or declare the workflow instance inconsistent and penalize the business partner that caused the fault.
- **Failure to recover from a "Data integrity fault"**: based on the signatures and workflow messages sent by the two business partners that were involved in the backup procedure of the corrupted workflow message, the workflow arbitrator determines whether the business partner of type b_x^v that stored this message has caused the corruption or whether the other business partner backed up a message that was corrupted in the first place. Based on the outcome of the mediation, the workflow arbitrator can either ask for a retransmission of the corrupted workflow message should a valid copy of it be available or declare the workflow instance inconsistent and penalize the business partner that caused the fault.
- **Failure to recover from a "Proof of execution fault"**: based on the signatures and workflow messages sent by the two business partners that were involved in the backup procedure of the corrupted workflow message, the workflow arbitrator determines whether the business partner of type b_x^v that stored this message has caused the corruption or whether

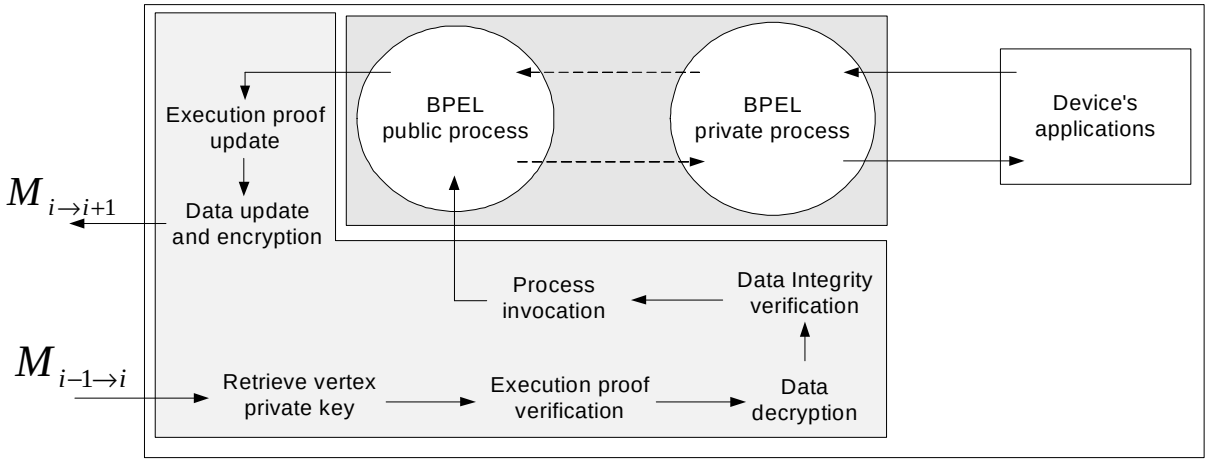

Figure 4.11: Integration of the security mechanisms within the engine wrapper implementation

the other business partner backed up a message that was corrupted in the first place. Based on the outcome of the mediation, the workflow arbitrator can either ask for a retransmission of the corrupted workflow message should a valid copy of it be available or declare the workflow instance inconsistent and penalize the business partner that caused the fault. In case the fault is the result of a signature missing in the onion O_p, the business partner that did not store it can be easily identified since the workflow initiator gathers all signatures prior to the workflow instantiation.

It should be noted that these situations wherein workflow message retransmission is not sufficient to recover are in fact highly unlikely to occur since the integration within the business partner registration process of a signature retrieval mechanism enabling anonymity revocation can be seen as a penalty for business partners that may cause security faults.

4.7 Implementation

We developed a security library in order to implement the mechanisms presented in this chapter. The implementation supports the following encryption algorithms:

- **IBE**: we use the Java Cryptography Extension (JCE) provider presented in [ODD04] that is an implementation of the ID-based cryptosystem specified in [BF01],
- **RSA-ECB**: we use the Bouncy Castle [Bou07] JCE provider that implements the RSA-ECB algorithm.

The RSA-ECB encryption algorithm is only used for demonstration purposes as the IBE implementation is resource-consuming ; the RSA-CBC scheme would be more suitable for de-

		IBE	RSA/ECB	Without security
4 vertices	2 data	25'31	1'39	1'10
4 vertices	4 data	35'44	2'01	1'16
4 vertices	6 data	51'18	2'07	1'21
2 vertices	2 data	8'30	0'48	0'37
4 vertices	2 data	25'31	1'39	1'10
6 vertices	2 data	46'50	4'28	2'03

Figure 4.12: Security mechanisms execution overhead

ployment but is unfortunately not implemented by the JCE provider we have used. The security library has been integrated into the engine wrapper implementation based on the sequence of operations specified in the sections 4.4.2 and 4.4.3. The basic principles of this implementation are depicted in figure 4.11. The complete security mechanism execution is handled by the engine wrapper that forwards decrypted data only to the BPEL workflow engine. Once the results are sent back by the BPEL engine to the engine wrapper, the latter processes data and builds the workflow messages that are sent to the next business partners involved in the workflow execution.

A complete specification of the implementation work can be found in appendix C.

4.7.1 Performance analysis

In order to evaluate the execution overhead resulting from the integration within the pervasive workflow runtime of the security mechanisms presented in this chapter, we measured the time required to complete the execution of different workflow specifications. The results are presented in figure 4.12. We basically performed two sets of measurements: the number of workflow vertices remains constant while the number of data processed by workflow changes and conversely. One needs to compare these results with the application that the pervasive workflow architecture is meant to support. For long running workflows that can span over days, the overhead introduced by the IBE encryption scheme is negligible whereas RSA based encryption schemes are more suitable for short running workflows.

4.8 Related work

Security of cross-organizational workflows in both centralized and decentralized settings has been an active research field over the past years [Coa98, KR01, LP03, BFA99] mainly focusing on access control, separation of duty and conflict of interests issues. However, in the decentralized setting issues related to the integrity of workflow execution and workflow instance forging, which are presented in this chapter have been left aside. This section discusses related work in the access control, separation of duty and conflict of interests research fields, before presenting

related work in the area of mobile agents and analysing how our results can leverage existing work in secure service composition.

4.8.1 Separation of duty and conflict of interests

The management of conflict of interests within a workflow execution consists in enforcing that workflow data which are sensitive for a business partner can not be accessed by business partners that are competitors (in the marketing sense) of the latter within the workflow instance. In the centralized setting whereby workflow management and control tasks rely on a trusted coordinator, solutions can be found to manage workflow data accordingly, however in the decentralized setting wherein all workflow data are transferred between partners this is a challenging issue.

In [CLW05],[ACM01] mechanisms are proposed for the management of conflict of interests [BN89] during the distributed execution of workflows. These pieces of work specify solutions in the design of access control policies to prevent business partners from accessing data that are not part of their classes of interest. These approaches do not address the issue of policy enforcement with respect to integrity of execution in fully decentralized workflow management systems yet they appear to complement our security mechanism design. The access control policy models suggested in [CLW05],[ACM01] can indeed be used to augment our work especially in the specification of the sets J_i^r and J_i^w at workflow design time so that the mechanisms we designed to enforce access control on workflow data integrate solutions for the management of conflict of interests.

The separation of duty principle within a workflow execution consists in ensuring that two or more distinct business partners should be involved in the completion of a set of related workflow tasks [SZ97]. As for the management of conflict of interests, existing solutions such as the one presented in [Hun04] can be used to augment our work at workflow design time in the specification of more fine-grained workflow policies associated with vertices.

4.8.2 Access control within workflow management systems

The enforcement of access control policies within workflow management systems has been a quite active research field over the past years especially in the centralized setting, only few contributions do however tackle this issue in the decentralized setting.

Most approaches rely on the Role Based Access Control model [SCFY96, WT04] to implement access control policies within the execution of a workflow [ASKP00]. In the centralized setting, many access control infrastructures have been as well proposed for business processes executed in the Service Oriented Architecture paradigm [TLC05, WT04, AkH96].

In [KM03, KPF01] centralized infrastructures are proposed which protect resources that should be accessed during the execution of a workflow based on credentials provided by business partners involved in the workflow instance. In [BCP06] a similar approach based on XACML [XAC05] is presented to enforce access control policies so that BPEL activities are executed by authorized users during the execution of BPEL processes. A methodology is specified in [WKRL06] that allows to determine the set of credentials required by a user to be able to execute some tasks of a workflow based on requirements specified in terms of security policies. Despite solid contributions, these approaches do not however meet the requirements introduced in the decentralized setting as they rely on a centralized component to issue access control decisions. They are as a matter fact designed to protect local resources rather than workflow data that are transferred between business partners without a centralized point of coordination in charge of business partner assignment and access control policy enforcement.

Some pieces of work have also tackled security issues within distributed collaborative applications. In [PH03] a solution enforcing RBAC policies is presented to protect the access to peers part of a peer-to-peer community. In this approach peers are able to make access control decisions autonomously without relying on an external policy decision point. In [TAK03] an access control model enabling the definition of dynamic RBAC policies enforcing dynamic separation of duties constraints is presented. The enforcement of these access control policies however relies on a dedicated middleware which acts as a centralized component to issue access control decision in the overall approach. Finally in [HK03] an access control model is also proposed that however do not include the notion of role and thus is not appropriate to meet the requirements we identified for distributed workflows.

4.8.3 Mobile agents and distributed applications

Onion encryption techniques have been introduced in [SGR97] and are widely used to enforce anonymity in network routing protocols [KH03] or mobile agents [KSY02]. In the approach presented in this chapter, we apply onion encryption techniques within a new application domain that is workflow-based applications and that introduces new requirements. We map onion structures with workflow execution patterns in order to build proofs of execution and enforce access control on workflow data. As a result, more complex business scenarios are supported by our solution than usual onion routing solutions. Furthermore, combined with policy encryption techniques, our solution provides a secure runtime environment for the execution of fully decentralized workflows supporting runtime assignment of business partners, a feature which had not been tackled so far. Existing solutions based on onion ring encryption developed in other application domains can not indeed be directly used to meet security requirements specified within decentralized workflow management systems.

A lot of secure data collection protocols have been designed in the mobile code research field [LMP01]. However these solutions do not offer an adequate support to assure that data transferred between sites or business partners are modified based on a predefined plan, their only purpose is as a matter of fact to enforce that once a piece of data has been stored by a peer

within the mobile agent it can only be modified an identified recipient.

4.8.4 Secure composition of business partners

The integration of security requirements within the building process of workflow instances is also of interest to the work presented in this chapter. In the fashion of the approach we specified in chapter 3 for transactional requirements, the pieces of work presented in [CAA04, Liu05, YHLC05] specify workflow instance building processes wherein parameters such as trust evaluation are integrated into the business partner selection process so that security requirements specified for a given workflow are met. The business partner selection algorithms specified in there can be used to augment our approach and integrated into the workflow vertex assignment procedure.

4.9 Conclusion

We presented mechanisms towards meeting the security requirements raised by the execution of workflows in the decentralized setting. Our solution, capitalizing on onion encryption techniques and security policy models, protects the access to workflow data with respect to the pre-defined workflow execution plan and provides proofs of execution to business partners. In addition, those mechanisms combined with group cryptography provide business partners' identity traceability for sensitive workflow instances and can easily be integrated into the runtime specification of decentralized workflows. Our approach is suitable for any business scenarios in which business roles can be mapped to security policies that can be associated with key pairs. It can thus be easily integrated into existing security policy models such as the chinese wall [BN89] security model.

We also showed how these security mechanisms can be integrated into the transactional framework that we presented in chapter 3. The goal of this approach is to make sure that detecting security faults does not necessarily mean the complete failure of the process execution. We presented the corresponding security-fault mechanisms relying on the workflow arbitrator to handle critical situations.

Finally, an implementation work based on Web services technologies and identity-based encryption techniques has been pursued as a proof of concept of our theoretical work.

Conclusions and Perspectives

The best way to predict the future is to invent it.
- Alan Kay -

In this thesis we presented the design of the pervasive workflow architecture, a decentralized workflow management system supporting runtime assignment of business partners to workflow tasks. We also specified solutions to offer the guarantees required by the execution of workflow-based applications supported by the pervasive workflow infrastructure in terms of security and transactional consistency. This conclusion chapter summarizes the contributions described in this report and is organized as follows. We first outline our theoretical results and their practical implementation. The possible modes of execution supported by the pervasive workflow model are then specified and we finally present some possible research directions.

Theory

The theoretical results presented in this thesis encompass the three following research areas.

Design of distributed workflow management systems As opposed to existing workflow management systems, the distributed execution of workflows in environments that do not offer a dedicated infrastructure can not rely on a centralized component to assure the workflow coordination task. To that effect, we suggested an architecture, denoted pervasive workflow, supporting distributed execution of workflows without the need of an ambient infrastructure implementing pre-configured services. The pervasive workflow architecture features a fully decentralized control and supports dynamic assignment of workflow tasks to business partners so that it meets the following requirements:

- **Fully decentralized**: the management of the workflow execution is distributed amongst the business partners taking part in a workflow instance in order to cope with the lack of dedicated infrastructure in the pervasive setting,
- **Dynamic assignment of business partners to workflow tasks**: the business partners in charge of executing the workflow can be discovered at runtime based on available resources or context.

The architecture design we presented is not only suited for the execution of workflows in the pervasive setting but also offer adequate support for the execution of workflows that cross organizational boundaries over traditional mediums including the Internet.

Composition and coordination of transactional business partners We presented an adaptive transactional protocol to support the execution of cross-organizational workflows. This approach actually meets the requirements that are relevant to assuring consistency of the execution of cross-organizational processes which are mainly twofold:

- **Relaxed atomicity**: atomicity of the workflow execution can be relaxed as intermediate results produced by the workflow may be kept intact despite the failure of one partner.
- **Dynamic selection of business partners**: the execution of cross-organizational workflows may require the execution of a composition procedure wherein candidate business partners offering different characteristics are assigned to tasks depending on functional and non-functional requirements associated with the workflow specification.

The execution of the protocol we proposed takes place in two phases. First, business partners are assigned to workflow tasks using an algorithm whereby partners are selected based on functional and transactional requirements. Given an abstract representation of a process wherein business partners are not yet assigned to workflow tasks, this algorithm enables the selection of partners not only according to functional requirements but also to transactional ones. The resulting workflow instance is compliant with the defined consistency requirements and its execution can be easily coordinated as our algorithm also provides coordination rules. The workflow execution further proceeds through a hierarchical coordination protocol managed by the workflow initiator and controlled using the coordination rules computed as an outcome of the partner assignment procedure. This transactional protocol thus offers a full support of relaxed atomicity constraints for workflow-based applications and is also self-adaptable to business partners' characteristics.

Security of distributed workflow management systems We specified the design of security mechanisms that support the secure execution of decentralized workflows. The latter meets the security requirements associated with the execution of distributed workflows that we identified. As opposed to traditional workflow management systems the distributed execution of workflows raises security constraints due to the lack of a dedicated infrastructure assuring the management and control of the workflow execution. These security requirements mainly consists of three categories:

- **Authorization**: only authorized business partners should be assigned to workflow tasks during a workflow instance,
- **Proofs of execution and traceability**: the workflow execution should be compliant with the workflow specification and it should be possible to trace back the business partners' identity,
- **Data protection**: this consists of assuring data confidentiality, ensuring data integrity and enforcing access control on data.

The mechanisms we presented capitalize on onion encryption techniques and security policy models in order to assure the integrity of the distributed execution of workflows, to prevent business partners from being involved in a workflow instance forged by a malicious peer and to provide business partners' identity traceability for sensitive workflow instances. The design of the suggested mechanisms is strongly coupled with the runtime specification of decentralized workflow management systems which eases their integration into existing distributed workflow management solutions.

Implementation

The theoretical results presented in thesis have been implemented based on the Service Oriented Architecture paradigm and Web services technologies. The implementation work we pursued is summarized in this section.

Pervasive workflow infrastructure Featuring a dynamic assignment of tasks to workflow partners, the pervasive workflow architecture allows users to initiate workflows in any environment where surrounding users' resources can be advertised by various means including a service discovery mechanism. The implementation of the pervasive workflow architecture we have pursued within the Service Oriented Computing paradigm is based on existing Web services technologies including BPEL that are adapted to meet the constraints of our architectural design. We designed a BPEL engine wrapper that implements the mechanisms and communication protocols we designed towards enabling the execution of workflows in the decentralized setting. This engine wrapper should be deployed on each business partner that wants to support the pervasive workflow model. In addition, we proposed an extension of the BPEL workflow description language that makes it possible to specify distributed workflows whereas the standard BPEL language can be only used in the specification of centralized workflows.

Pervasive workflow transactional coordination framework We implemented a transactional framework in the fashion of the WS-Coordination specification that enables on the one hand the composition of transactional business partners based on the OWL-S language and on the other hand the transactional coordination of cross-organizational business processes. The coordination protocol we implemented is adaptive in that the coordination

rules it is based on depend on the transactional properties that the business partners involved in a workflow instance offer. In fact, it overcomes the current limitations of recent efforts [La05a],[La05b] as our protocol offers the adequate flexibility to meet the requirements of the Service Oriented Architecture paradigm in terms of relaxed atomicity. We also presented transactional BPEL process specifications that match the transactional model underpinning our theoretical results.

Deployment of security mechanisms within the pervasive workflow infrastructure We developed a security library that implements the security mechanisms we designed to support the execution of decentralized workflows. The implementation is based on a Java Cryptography Extension provider that implements the ID-based cryptosystem. The security library has been integrated into the engine wrapper implementation based on the mechanism design. Our implementation can also be used with the RSA-ECB encryption scheme to optimize performances when required.

Execution modes supported by the pervasive workflow model

Based on the mechanisms we introduced in chapters 2, 3 and 4, we can now summarize the various execution modes that are supported by the pervasive workflow infrastructure.

- **Lightweight pervasive workflow infrastructure**: neither the transactional protocol nor the security mechanisms are implemented during the execution of workflow instances. The discovery mode can either be runtime or source mode.

- **Secure pervasive workflow infrastructure**: Security mechanisms are implemented during the execution of workflow instances. The discovery mode can either be runtime or source mode.

- **Secure pervasive workflow infrastructure with traceability**: security mechanisms with group cryptography are implemented during the execution of workflow instances. The discovery mode can either be runtime or source mode.

- **Transactional pervasive workflow infrastructure**: the transactional protocol supports the execution of workflow critical zones. The discovery mode is by default source mode.

- **Secure and Transactional pervasive workflow infrastructure**: the transactional protocol and the security mechanisms with group cryptography support the execution of workflow critical zones. The discovery mode is by default source mode.

These possible execution modes are basically selected based on the requirements associated with the workflow specification.

Perspectives

Finally, we give an overview of the possible lines of research that could be carried out based on the results presented in this thesis.

Workflow modeling Within this thesis we only considered workflows featuring simple execution patterns namely the SEQUENCE, AND-SPLIT/JOIN and OR-SPLIT/JOIN patterns. Besides, we did not consider synchronization issues between concurrent branches. It could interesting to study how the infrastructure and the solutions we designed could be extended to support more complex execution scenarios and to define the corresponding extensions within our distBPEL language.

Secure business partner composition The business partner selection process in terms of security credentials is in the scope of this thesis limited to a simple match-making process between the security credentials required by the workflow execution and those provided by business partners. We could in addition integrate some other parameters such as trust evaluation in the fashion of the methodology we designed for transactional properties offered by business partners.

Integration of complex security policies As we mentioned in the related work section of chapter 4, it would be interesting to integrate complex security policy models at workflow design time to avoid conflict of interests between business partners and satisfy separation of duty constraints.

Appendix A

Engine wrapper implementation

We present in this appendix the pervasive workflow implementation we developed as a proof of concept for the theoretical results presented in chapter 2. The implementation work we pursued consists of the development of the engine wrapper whose design is outlined in chapter 2 and that of an Ajax-based workflow visualization tool that enables the runtime visualization of a pervasive workflow execution.

A.1 Engine wrapper interface and UML [UML] diagrams

We first outline the overall classes that are implemented in order to support the pervasive workflow architecture execution. The interactions between those classes are then illustrated within two sequence diagrams. Finally the Application Programming Interface (API) exposed by the workflow engine wrapper is commented and justified based on the model design.

A.1.1 Class and sequence diagrams

Figure A.1 depicts the engine wrapper class diagram. The engine wrapper interface is implemented by the *MessageHandler* class. The workflow execution on a given business partner is initialized upon receipt of a *WorkflowMessage* sent to the *processMessage* operation. This results in the instantiation of the *MessageHandler* class. This instance of *MessageHandler* then contacts the *InstanceAdministration* class and retrieves the corresponding *WorkflowInstance* that is a class wherein workflow control data for a given instance of a workflow are stored. The *InstanceAdministration* class is implemented based on the singleton design pattern [GHJV07] in order to make sure that a single instance of this class is created regardless of the number of workflow messages received by a given business partner. The main goal of this design is in

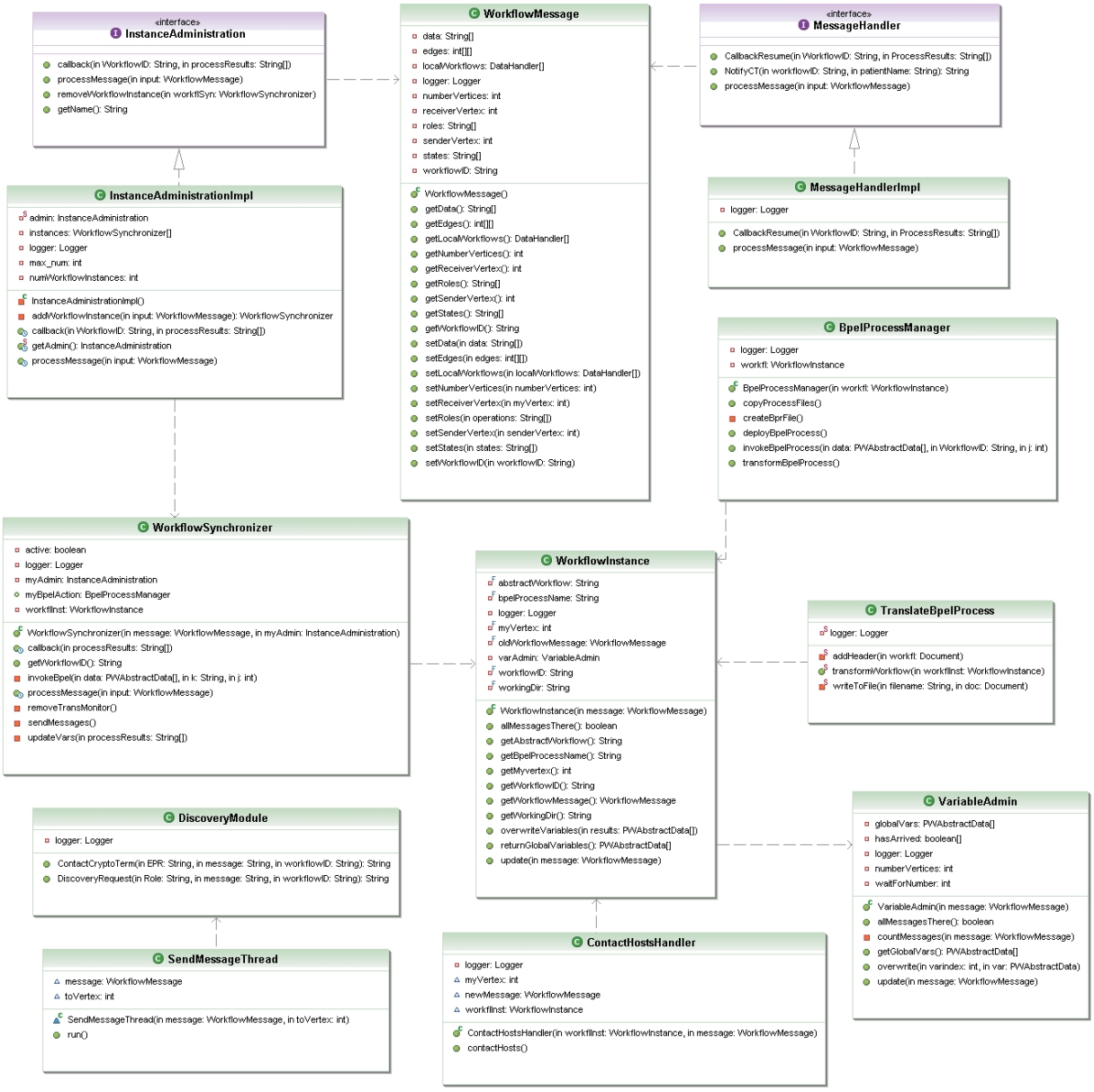

Figure A.1: Pervasive Workflow: class diagram

fact to support asynchronous message exchanges within workflow instances that can span over hours or days.

Each *WorkflowInstance* is wrapped by a *WorkflowSynchronizer* class that serves as an access guard and prevents that several instances of the *MessageHandler* class try access to same instance concurrently e.g. in the case of an AND-Join workflow pattern. The *WorkflowMessage* is then transferred to the *WorkflowInstance* it belongs to. One should notice here that thanks to this design, the *InstanceAdminstration* class is able to handle different instances of the same workflow at the same time. The main thread of the program for a given workflow instance takes place in the *WorkflowSynchronizer* class, implemented within the *processMessage* operation. The latter consists of the update of workflow data handled by the *VariableAdmin* class and the deployment and invocation of the public BPEL process implemented within the *BPELProcessManager* class.

The communications between the engine wrapper and the BPEL engine at the business partner level are asynchronous in order to support the execution of processes that can span over hours. The *MessageHandler* class therefore implements a callback procedure which simply resumes the execution of a workflow instance upon completion of the BPEL processes. The workflow execution further proceeds with the discovery of the next business partners involved in the workflow execution (*Discoverymodule* class) and the invocation of the business partners that have just been discovered and assigned to further workflow tasks (*ContactHostsHandler* class).

The sequence diagrams corresponding to the *processMessage* and *callback* operations of the *MessageHandler* class that specify the engine wrapper business logic are depicted in figures A.2 and A.3, respectively.

A.1.2 Engine wrapper interface

The engine wrapper communicates with other business partners involved in a workflow through the *MessageHandler* interface which is thus exposed as a Web service. This interface implements the two following asynchronous operations:

- **ProcessMessage** that takes as input a *WorkflowMessage* compliant with the XML schema depicted in table A.1,
- **CallbackResume** that resumes the workflow execution upon completion of the BPEL processes locally executed by a business partner. This callback procedure takes as input the workflow instance identifier and the data sent back by the local applications.

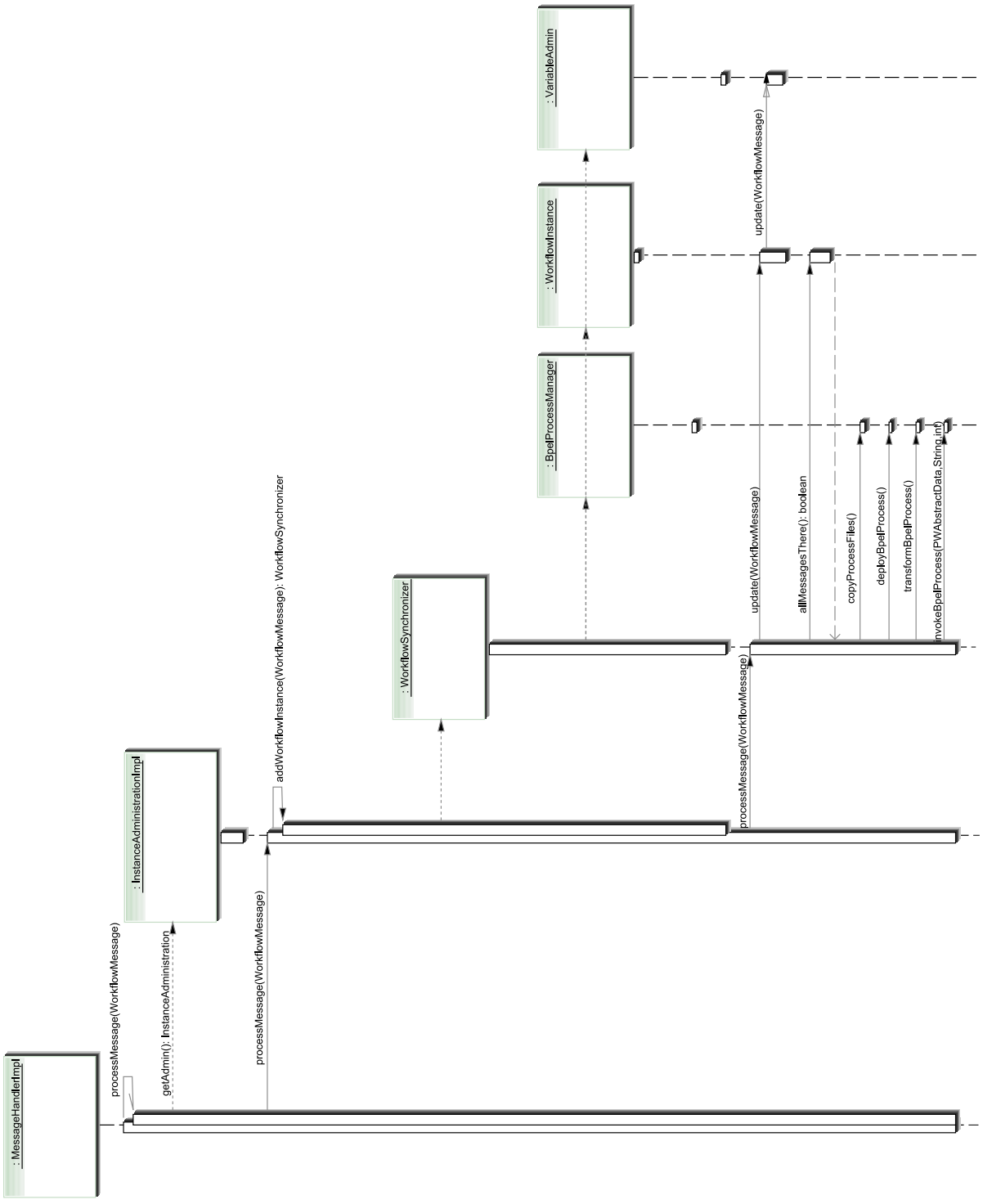

Figure A.2: Process message: sequence diagram

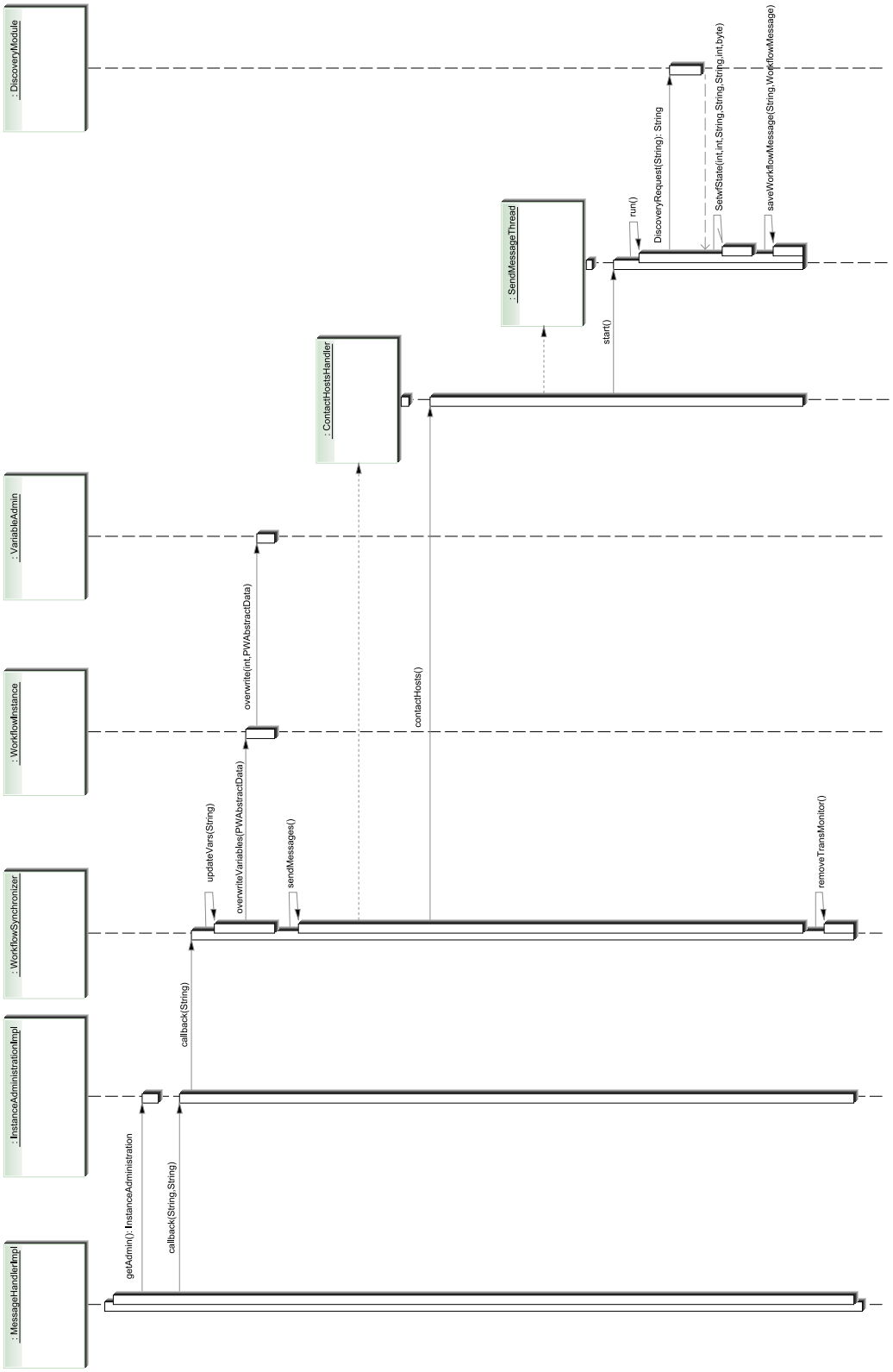

Figure A.3: Callback: sequence diagram

```
<complexType name="WorkflowMessage">
 <sequence>
  <element name="data" type="ArrayOf_xsd_string" />
  <element name="edges" type="ArrayOfArrayOf_xsd_int" />
  <element name="localWorkflows" type="ArrayOf_xsd_base64Binary" />
  <element name="numberVertices" type="xsd:int" />
  <element name="receiverVertex" type="xsd:int" />
  <element name="roles" type="ArrayOf_xsd_string" />
  <element name="senderVertex" type="xsd:int" />
  <element name="workflowID" type="xsd:string" />
 </sequence>
</complexType>
```

Table A.1: Workflow message XML schema

A.2 Workflow visualization tool

In order to visualize the execution and the business logic of a pervasive workflow instance, we developed a graphical tool leveraging Web technologies including SVG [SVG07] and Ajax [Aja]. The principles of this visualization application are depicted in figure A.4. The visualization application is composed of the two following components:

- **Application server**: the application server is the core of the visualization application. It publishes an SVG/Ajax document that is a graphical representation of the workflow execution and implements JSP pages that are used to store the state of pervasive workflow executions.

- **Web browser**: this is the client side that allows to visualize the events occurring during a pervasive workflow instance. The browser displays an SVG document whose structure (DOM [DOM05]) is dynamically updated by an Ajax script based on the state of the workflow execution. The SVG document and the Ajax script are downloaded once from the application server, the browser then locally executes Ajax routines.

At runtime, the business partners involved in a workflow instance update the state of the workflow execution communicating with the application server by means of JSP pages [JSP05]. On the client side, the Ajax script periodically queries the application server by means as well of JSP pages in order to retrieve the current state of the workflow execution and updates the SVG document structure accordingly. We designed two SVG documents to monitor the execution of pervasive workflow instances. The first SVG document depicted in figure A.5 is a graphical representation of the execution steps of a pervasive workflow. Of course, the graphical layout depends on the execution patterns of the particular workflow whose execution is monitored. The second SVG document depicted in figure A.6 illustrates the business logic of the engine wrapper we implemented.

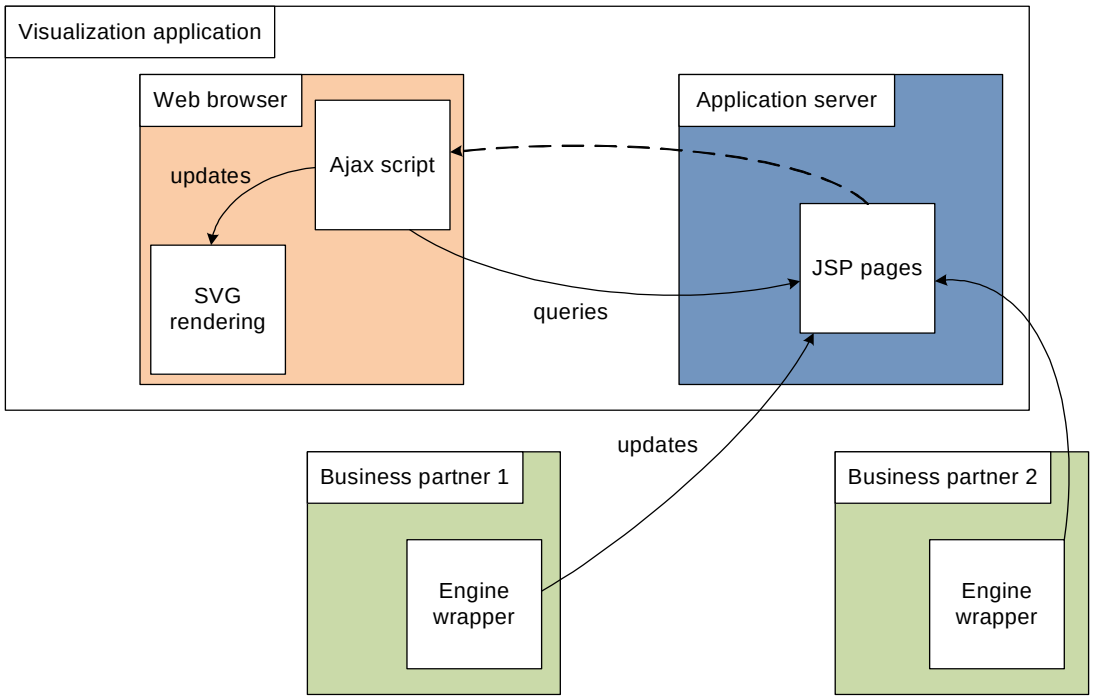

Figure A.4: Pervasive workflow visualization application principles

A.3 Deployment

The implementation presented in this chapter has been developed based on the following technologies:

- **J2EE**: The prototype has been implemented based on Java [Jav]
- **Tomcat**: Application server [Tom]
- **AXIS**: SOAP processing engine [AXI]
- **Active-BPEL**: BPEL engine

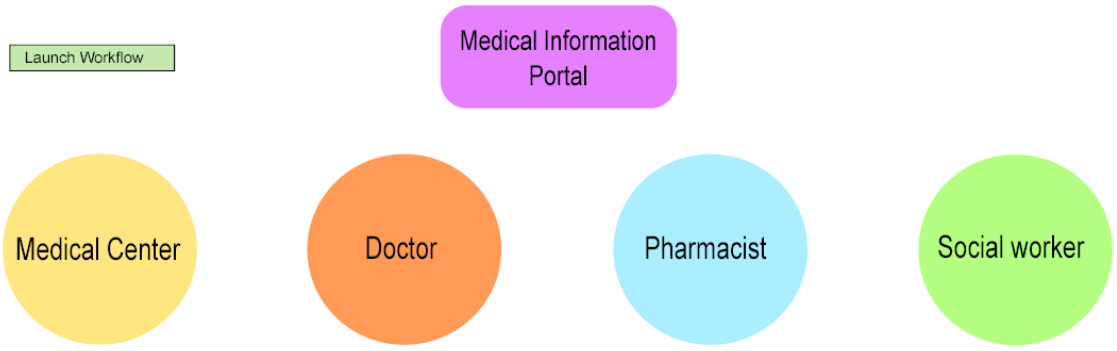

Figure A.5: Pervasive workflow execution: Screen shot of the visualization application

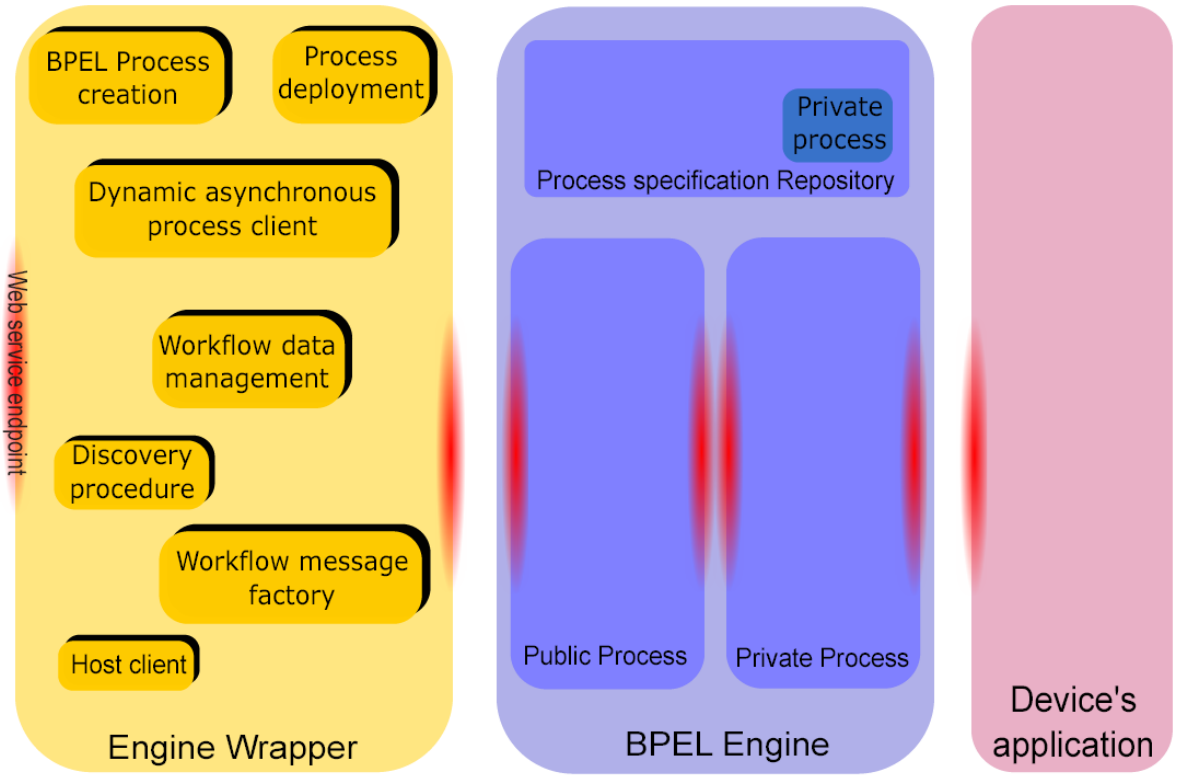

Figure A.6: Engine wrapper business logic: Screen shot of the visualization application

Appendix B

Transactional framework implementation

Chapter 3 already provides a complete specification of the coordination framework we designed and implemented towards meeting the consistency requirements we identified for the pervasive workflow management system. In this section we thus only give some details on the composition algorithm we designed and on the demonstrator we implemented to illustrate the coordination protocol execution.

B.1 Transaction-aware composition algorithm

The transaction-aware composition algorithm we implemented is presented in figure B.1. As specified in chapter 3, the algorithm (limited to the assignment of business partners to vertices of type v_k) consists of the following steps:

1. Assign business partners of type $(r_l, r_t c)$.
2. Compute the transactional requirements associated with vertices for which only business partners of type (r_l, p) are available, return false if (r_l, p) is not sufficient.
3. Compute the transactional requirements associated with vertices for which only business partners of type (r_l, r_t) are available, return false if (r_l, r_t) is not sufficient.
4. Compute the transactional requirements associated with vertices for which only business partners of type (r_l, c) are available, return false if (r_l, c) is not sufficient.
5. For each remaining vertex to which no business partner has been assigned yet compute transactional requirements:
    - If $(r_l, r_t c)$ is required return false.

- If (r_l, r_t) is required assign a business partner of type (r_l, r_t).
- If (r_l, c) is required assign a business partner of type (r_l, c).

Each new assignment of a business partner to a vertex changes the transactional requirements associated with vertices to which no business partner has been assigned yet; thus repeat this step until only vertices requiring (r_l, p) remain.

6. For each remaining vertex to which no business partner has been assigned yet assign a business partner of type (r_l, r_t).

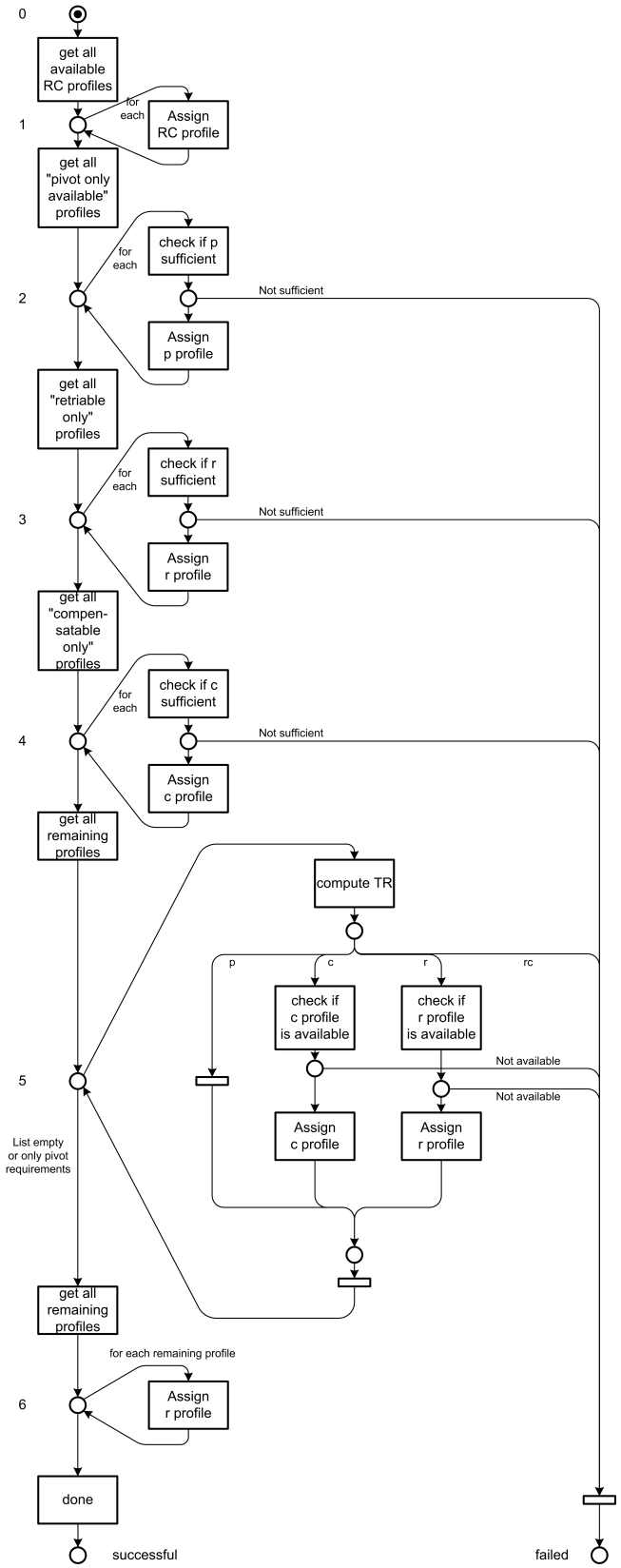

Figure B.1: Transaction-aware business partner assignment procedure

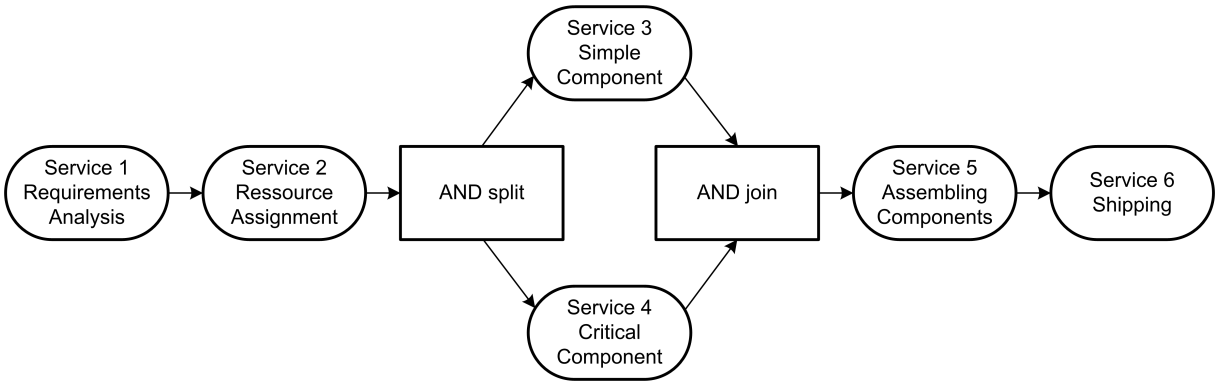

Figure B.2: Simple workflow example

B.2 Demonstrator

We implemented a stand alone prototype to demonstrate the transactional framework whose design is presented in chapter 3. The demonstrator features a simple workflow depicted in figure B.2 and the scenario execution takes place in two phases. Service providers are first selected based on the transaction-aware selection algorithm we designed and integrated into an OWL-S matchmaker (refer to section 3.9.1 for details). The resulting workflow instance is then executed, supported by the transactional coordination protocol presented in chapter 3. Chapter 3 already provides a thorough description of the implementation work we pursued we thus only comment some screen shots from the visualization application that we adapted to monitor message exchanges between the components of the transactional framework. The following execution scenario is based on the acceptable termination state table depicted in table B.2 and assumes that the set of service providers depicted in table B.1 is assigned to the workflow vertices. Furthermore, we consider that service provider 4 fails to complete the vertex it is assigned to.

	Retriable	Compensatable	Reliable
Service 1	yes	no	yes
Service 2	yes	yes	yes
Service 3	yes	yes	yes
Service 4	no	yes	yes
Service 5	yes	no	yes
Service 6	yes	no	yes

Table B.1: Service providers assigned to the vertices of the workflow depicted in figure B.2

Business partner registration In the first phase of the coordination protocol execution, business partners register to the coordinator as depicted in figure B.3.

Runtime At runtime, service providers are sequentially activated and complete the execution

	Service 1	Service 2	Service 3	Service 4	Service 5	Service 6
ats_1	completed	completed	completed	completed	completed	completed
ats_2	completed	failed	aborted	aborted	aborted	aborted
ats_3	completed	compens.	compens.	failed	aborted	aborted
ats_4	completed	compens.	canceled	failed	aborted	aborted
ats_5	completed	compens.	compens.	compens.	failed	aborted

Table B.2: Transactional requirements associated with the workflow depicted in figure B.2

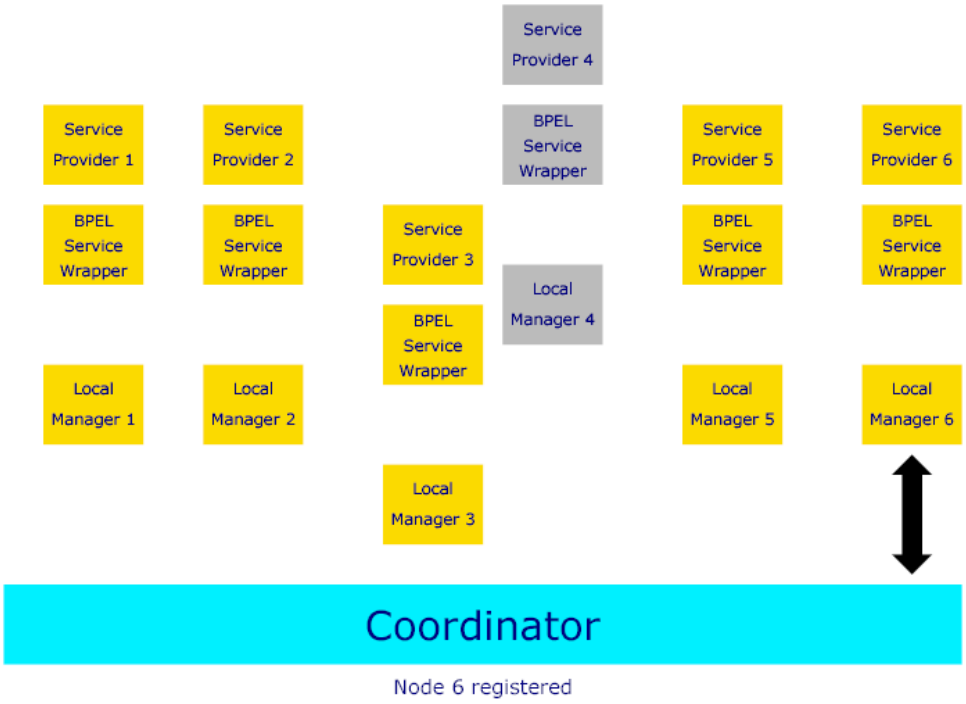

Figure B.3: Service registration

of the vertex they have been assigned to as depicted in figure B.4. Upon failure of service provider 4, a failure message is forwarded to the coordinator as depicted in figure B.5.

Coordination strategy Upon receipt of the failure notification sent by service provider 4, the coordinator sends appropriate abortion, cancellation and compensation messages to service providers based on the *ATS* table specifying the transactional requirements associated with the workflow. For instance, the execution of the vertex assigned to the service provider 3 is canceled as depicted in figure B.6.

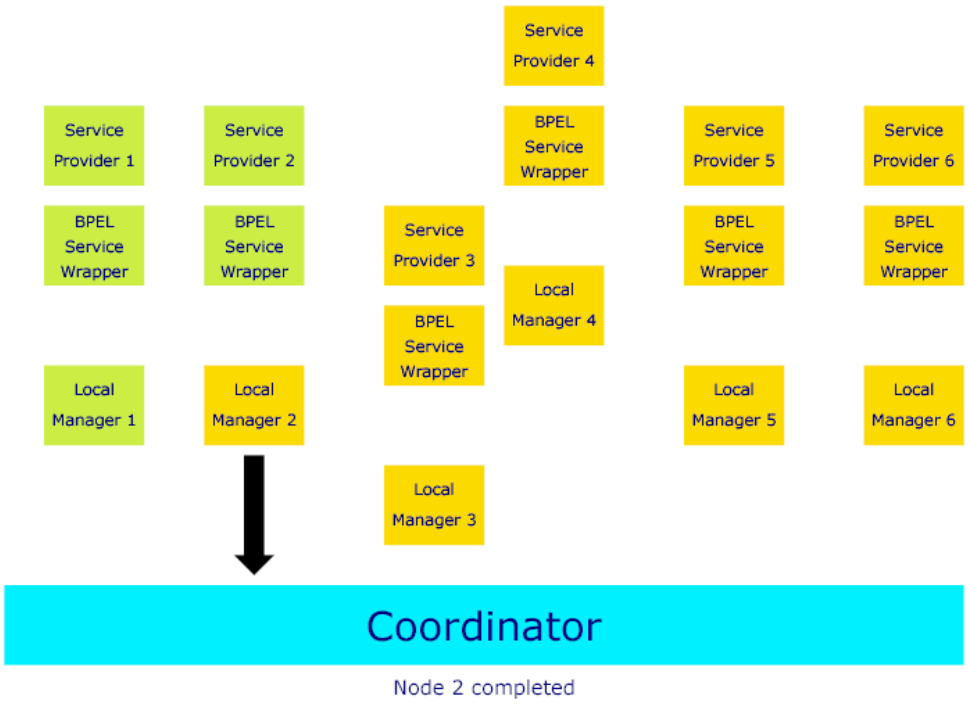

Figure B.4: Service completion

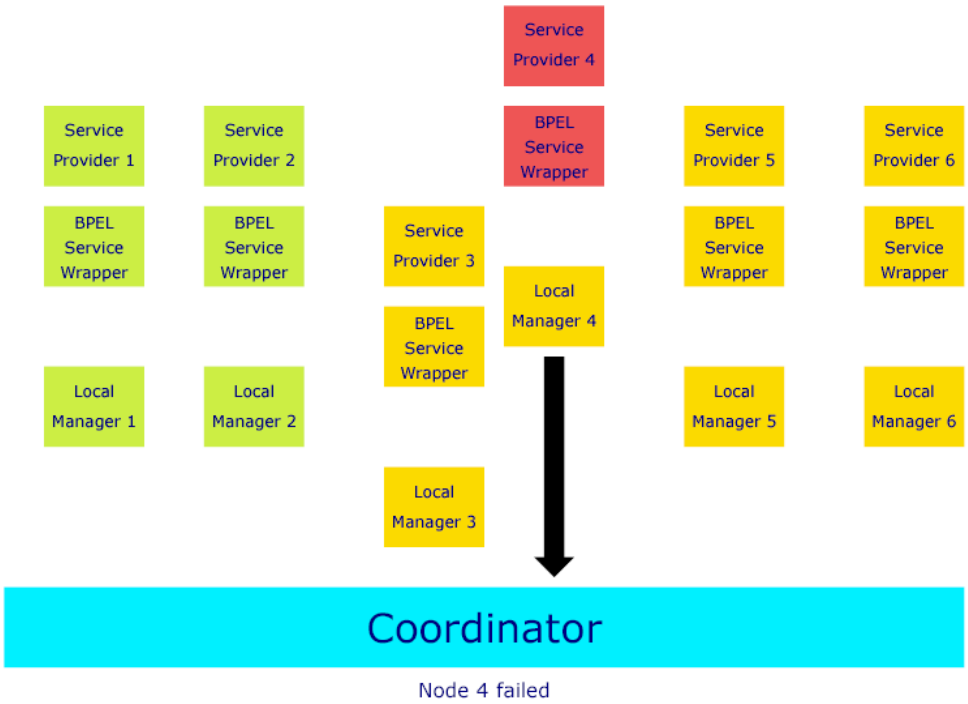

Figure B.5: Service failure

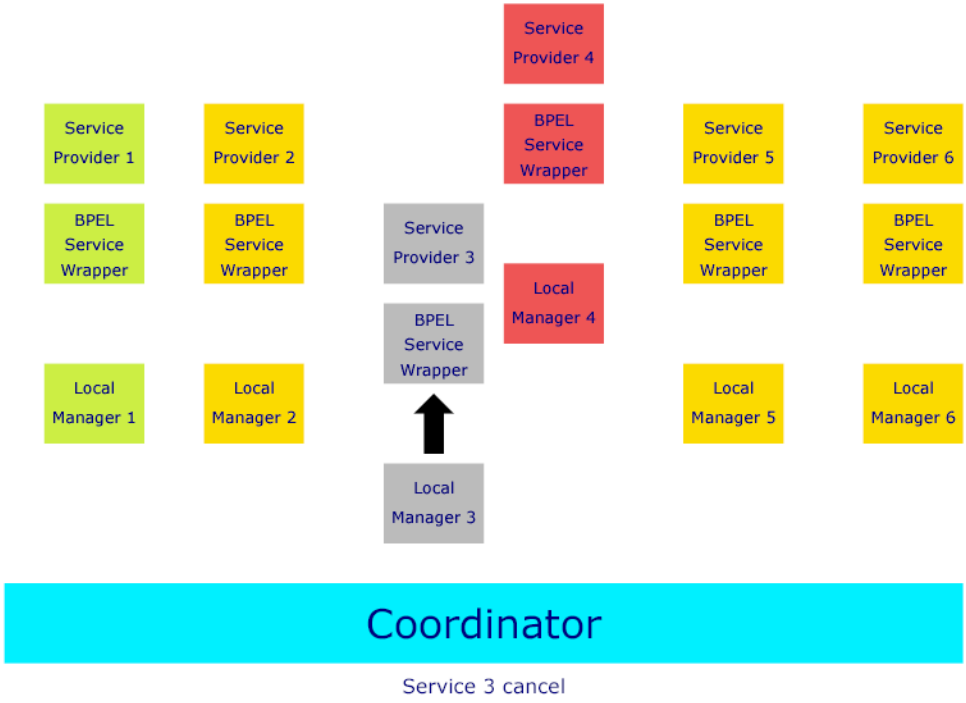

Figure B.6: Service cancellation

Appendix C

Security library implementation

We present in this appendix the implementation of the security solutions specified in chapter 4 to secure the execution of pervasive workflows. The security mechanisms we designed have been integrated within the engine wrapper implementation presented in appendix A. In what follows, insights are provided on this integration work and on the algorithms associated with the management of the onion structures specified in chapter 4.

C.1 Integration of security primitives into the business logic of the engine wrapper

The implementation of the security mechanisms we designed is composed of the classes depicted in figure C.1. The latter can be grouped as follows.

- **Key management**: the *InitKeyManger* class implements the operations required by the workflow initiator to create the set of keys and security parameters required by the workflow execution. The *KeyManagerRSA* and *KeyManagerIBE* classes implement the encryption and signature primitives associated with the RSA and IBE cryptosystems, respectively.
- **Onion management**: the *ManagementOpRSA*, *ManagementOdRSA*, *ManagementOpIBE* and *ManagementOdIBE* classes implement the execution primitives associated with the onion structures O_d and O_p.
- **Workflow data management**: the integration of the security mechanisms into the engine wrapper business logic required to modify data structures and to design operations to manage the latter. The *DataBlock*, *DataWorkflowManagement*, *EncryptedDataStructure* implement these new functionalities.

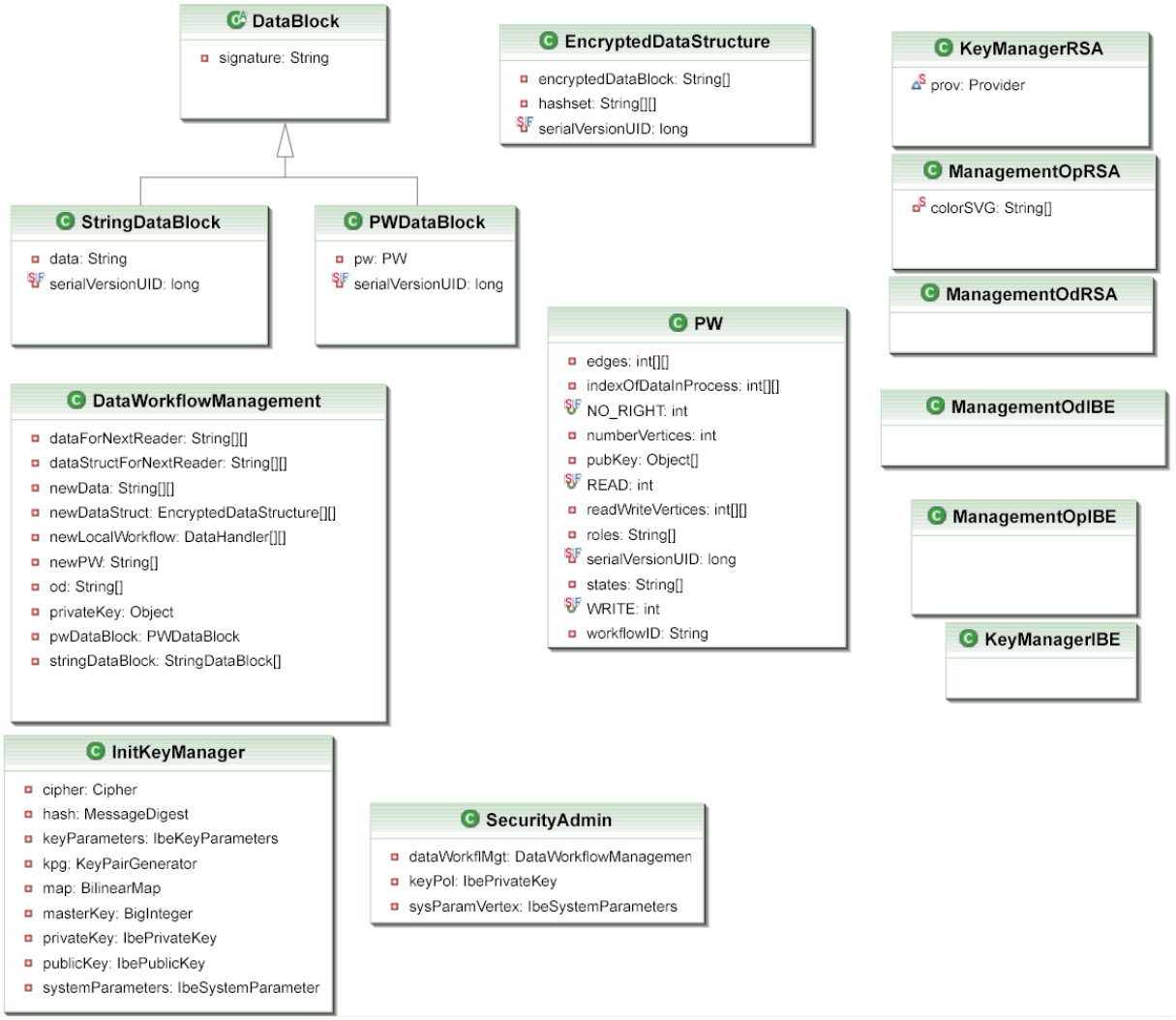

Figure C.1: Security library class diagram

- **Workflow policy**: the *PW* class defines the data type associated with the workflow policy P_W.

The *SecurityAdmin* class makes the glue between this set of classes and the engine wrapper classes when it comes to integrating the primitives associated with the execution of the security mechanisms, as depicted in the sequence diagrams of figures C.2 and C.3.

The workflow message structure has been also modified to match the one defined in figure 4.8 as depicted in table C.1. Some workflow control data including the workflow adjacency matrix (*edges* element), the workflow instance identifier (*workflowID* element), the message sender and recipient vertices are sent as clear text in order to support AND-JOIN workflow patterns. The engine wrapper should indeed be able in this case to determine how many workflow messages it has to wait for prior to executing a vertex.

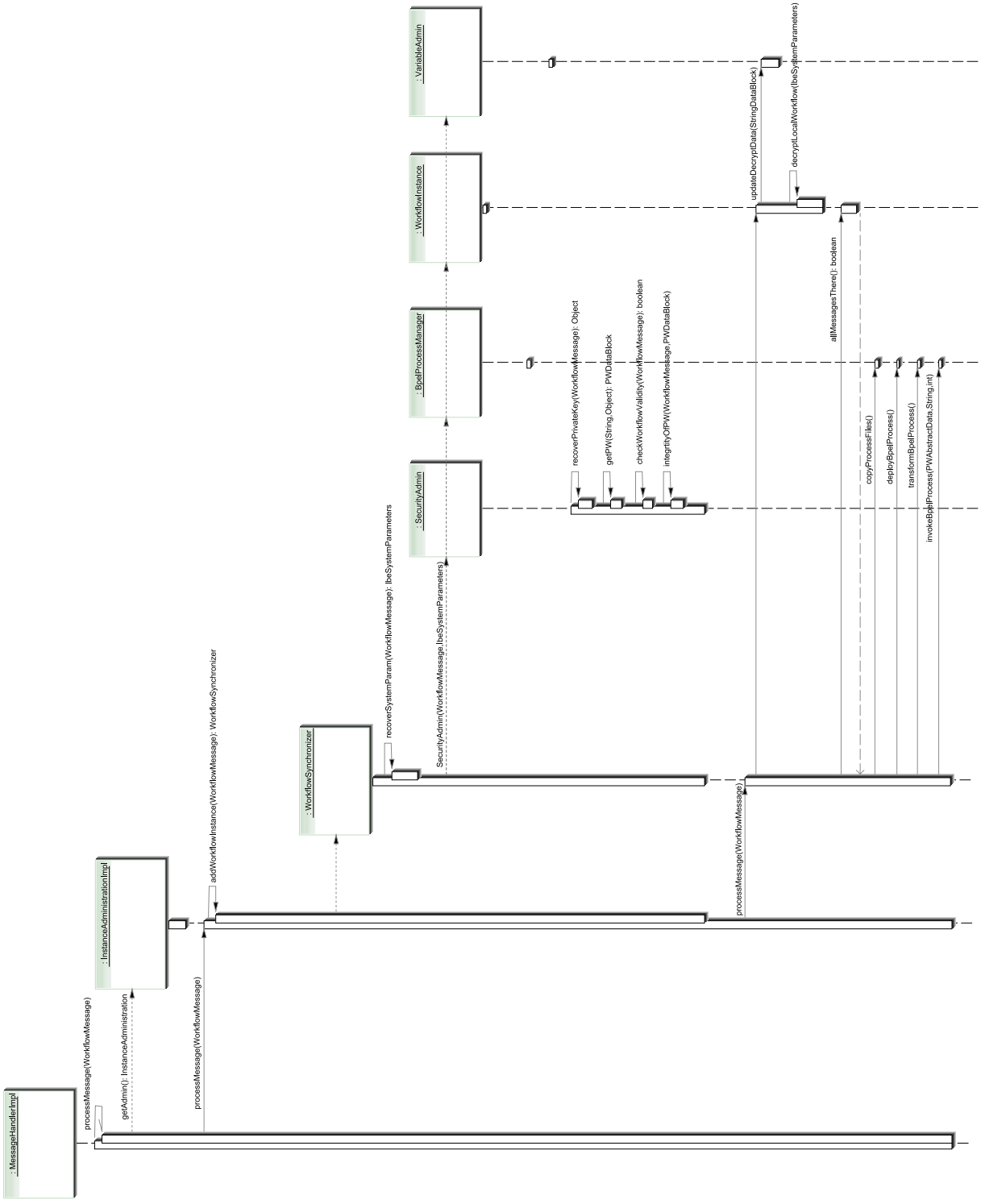

Figure C.2: Process message with security mechanisms: sequence diagram

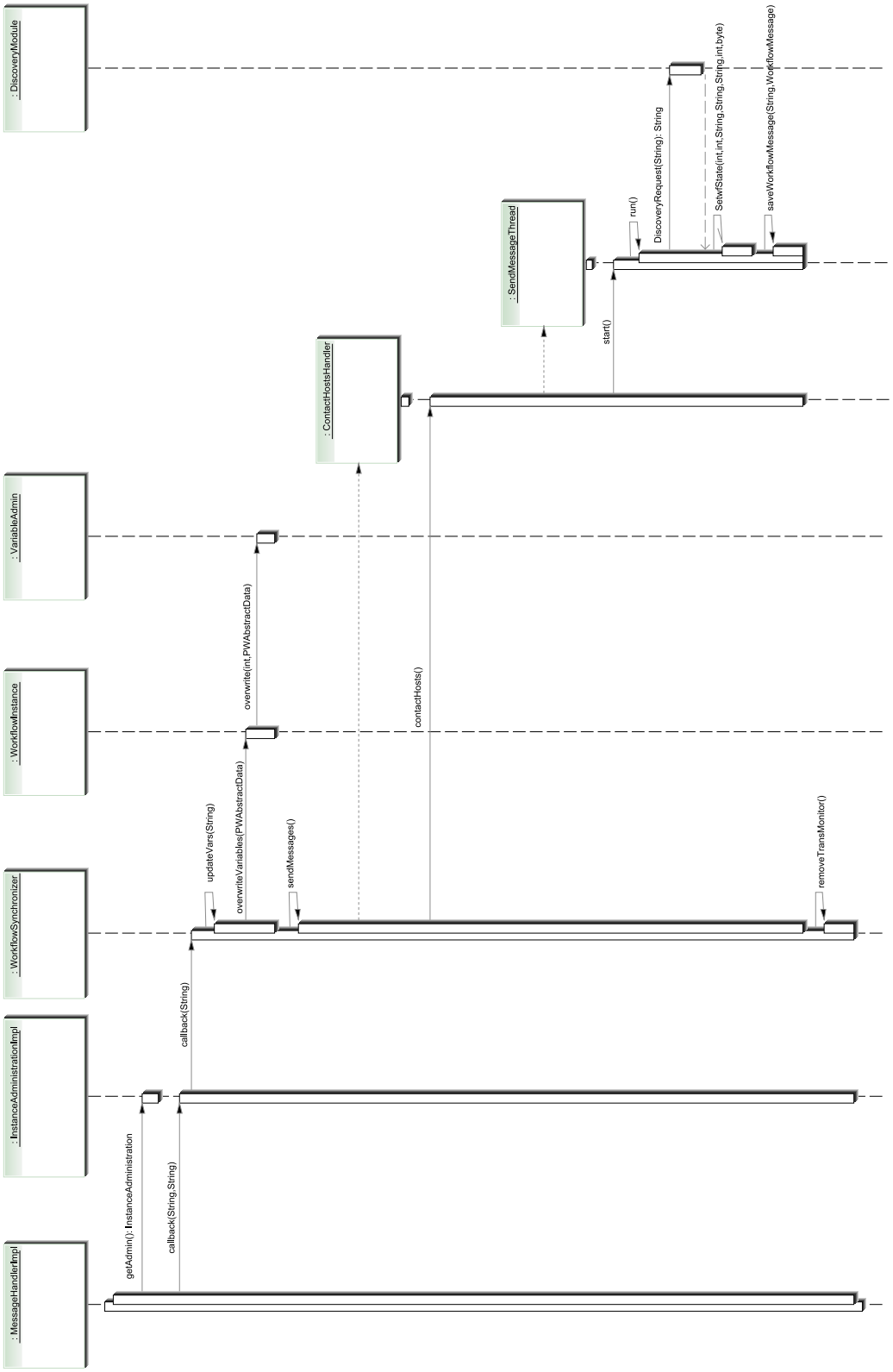

Figure C.3: Callback with security mechanisms: sequence diagram

```
<complexType name="WorkflowMessage">
 <sequence>
  <element name="PW" type="xsd:string"/>
  <element name="edges" type="impl:ArrayOfArrayOf_xsd_int"/>
  <element name="encryptedDataBlock" type="impl:ArrayOf_xsd_string"/>
  <element name="HashSets" type="xsd:string"/>
  <element name="encryptionSystemParameters" type="xsd:string"/>
  <element name="encryptionType" type="xsd:string"/>
  <element name="localWorkflows" type="impl:ArrayOf_DataHandler"/>
  <element name="od" type="impl:ArrayOf_xsd_string"/>
  <element name="op" type="impl:ArrayOfArrayOf_xsd_string"/>
  <element name="receiverVertex" type="xsd:int"/>
  <element name="senderVertex" type="xsd:int"/>
  <element name="workflowID" type="xsd:string"/>
 </sequence>
</complexType>
```

Table C.1: Workflow message XML schema integrating security mechanisms

C.2 Integration of security primitives into the visualization tool

The visualization tool we implemented to monitor the execution of a pervasive workflow has been adapted in order to visualize the execution of the security mechanisms we developed. The corresponding SVG pictures are shown in figures C.4 and C.5. The former illustrates the peeling off process of O_d and the building process of O_p throughout the workflow execution while the latter shows how the security mechanisms are integrated into the engine wrapper business logic.

C.3 Onion processing

In this section the algorithms we designed to assure the management of the onion structures O_d and O_p are specified. The building process of the onion O_d is first presented, the peeling off process of the onion O_p is then described.

C.3.1 Onion O_d building process

We implemented the building process of the onion structure O_d that is performed by the workflow initiator using a breath-first algorithm. Starting from the last vertex of the workflow, the layers of O_d are sequentially encrypted based on the workflow execution pattern associated with the current vertex, as depicted in figure C.6. The peeling off process of O_d only consists of the

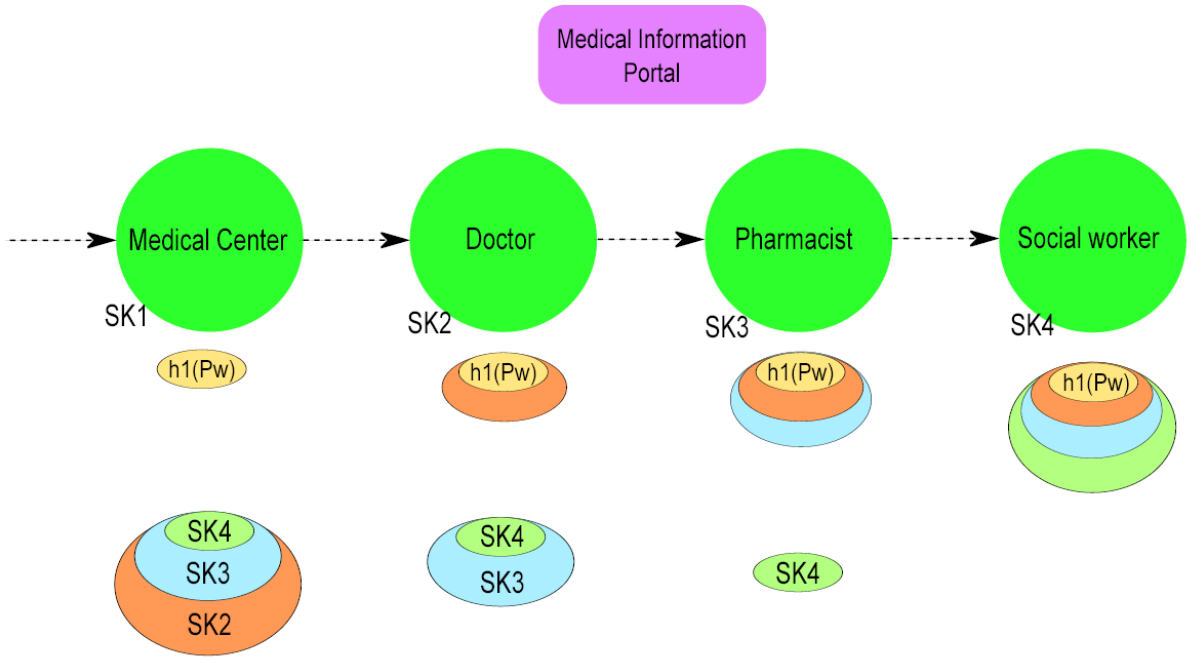

Figure C.4: O_d peeling off process and O_p building process: Screen shot of the visualization application

decryption of a single layer and is thus not mentioned.

C.3.2 Onion O_p peeling off process

Upon receipt of a workflow message requesting the execution of a vertex, business partners verify the integrity of the workflow execution by completely peeling off the onion O_p. We implemented the peeling off process of O_p using a recursive procedure as depicted in figure C.7. The latter is a depth-first algorithm: starting from the last vertex of the workflow, this procedure identifies all workflow execution branches and verifies the integrity of each of the latter.

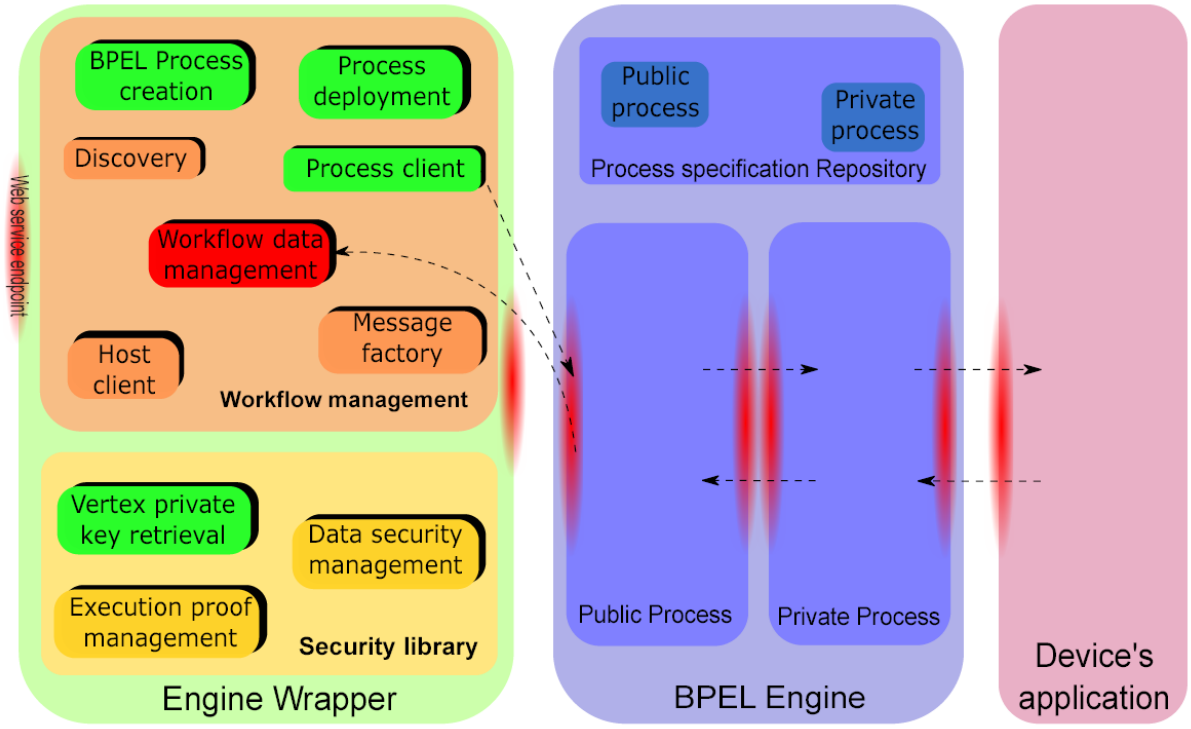

Figure C.5: Integration of the security mechanisms into the engine wrapper business logic: Screen shot of the visualization application

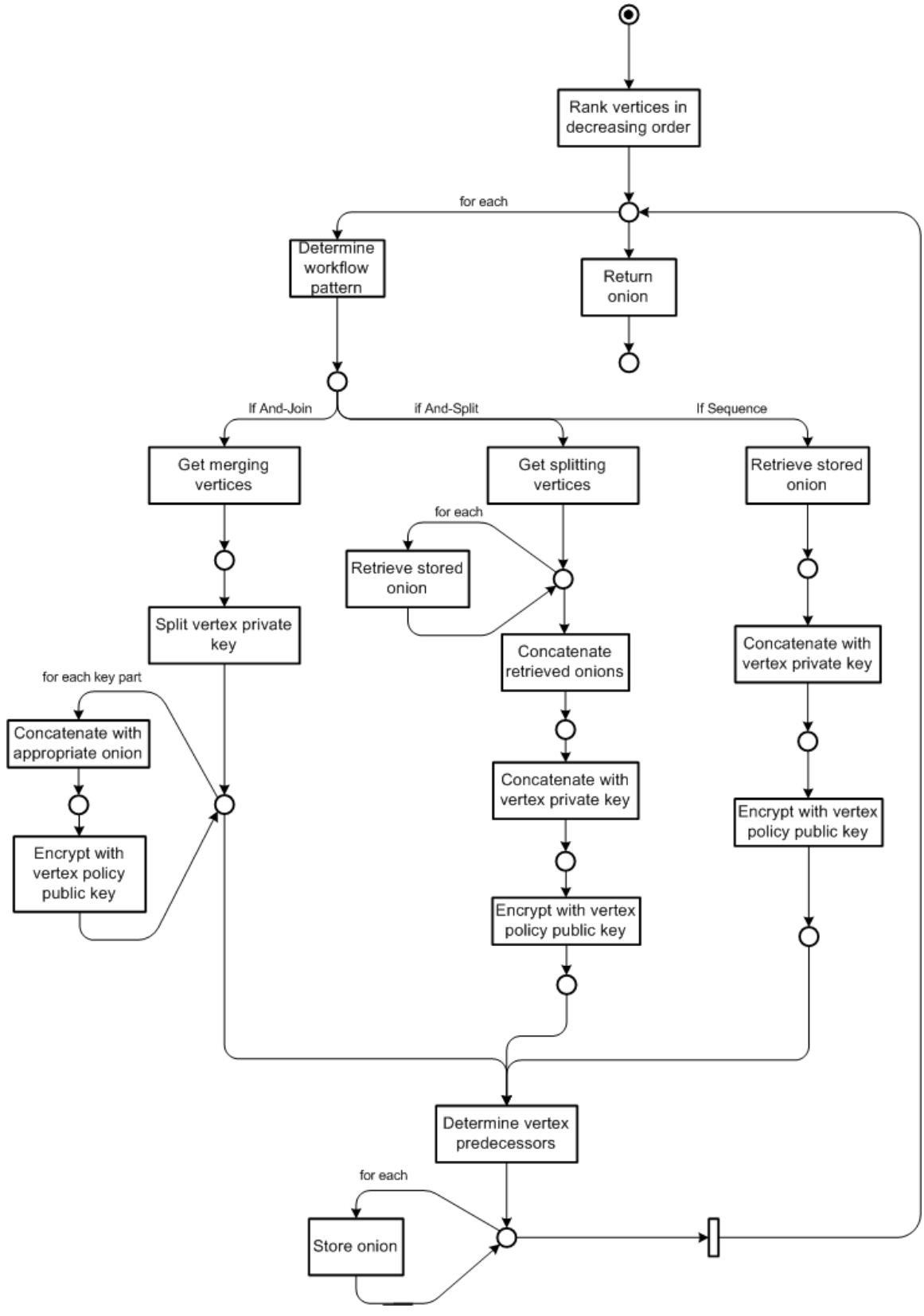

Figure C.6: Onion O_d building process

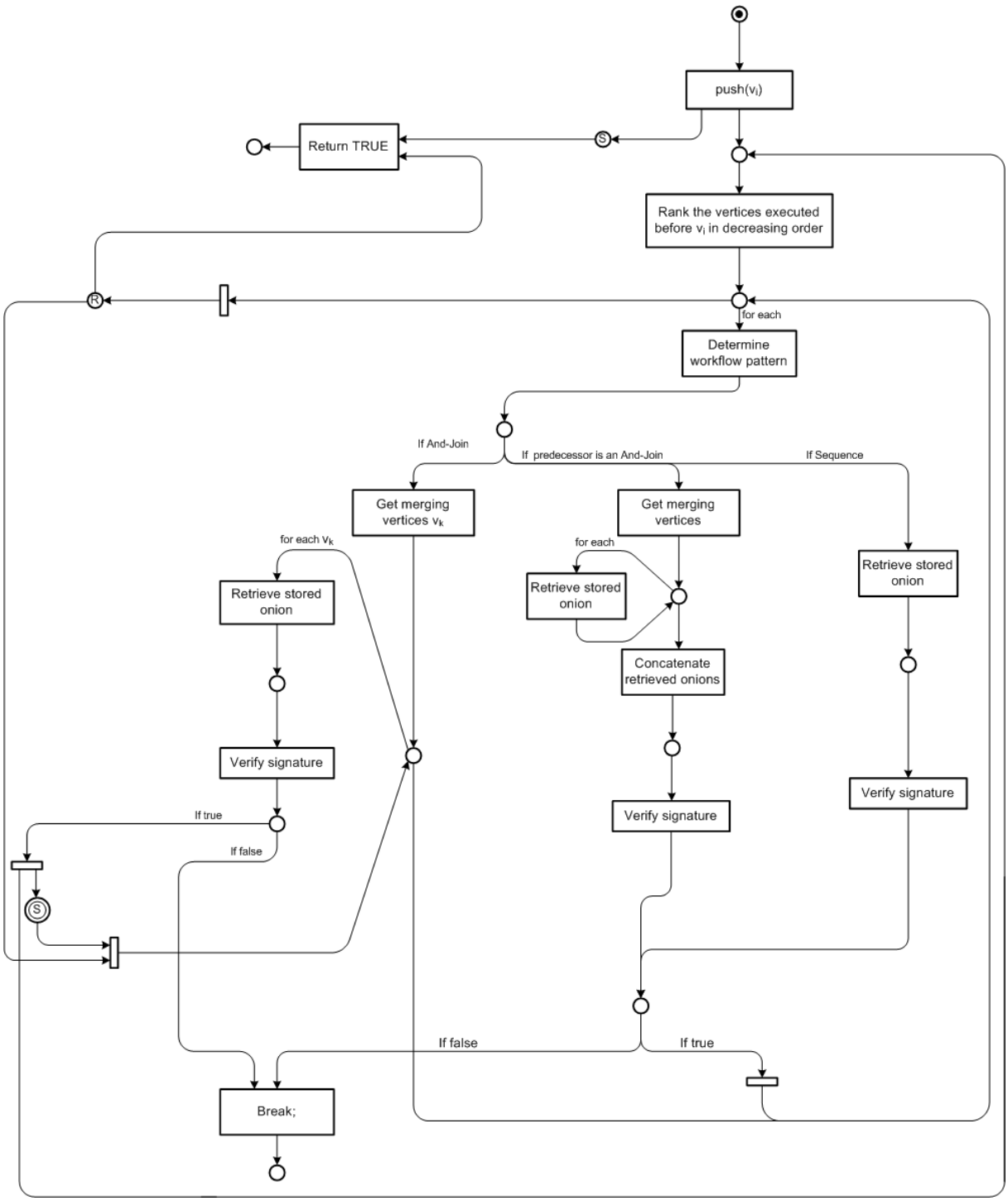

Figure C.7: Onion O_p peeling off process

Appendix D

Prototype developed in the context of the MOSQUITO project

The pervasive workflow infrastructure was designed in the context of the MOSQUITO IST European project [mos]. The main outcome of the MOSQUITO project was the development of a collaborative application in the eHealthcare application domain, whose enabling technology is the pervasive workflow management system. In this appendix, we present the prototype developed within the project as a proof of concept. The prototype demonstration scenario outlines the execution of a workflow wherein a physician, a pharmacist and a social worker collaborate to provide medical care to a patient. The demonstrator scenario is first detailed. The prototype infrastructure components are then described. Finally, we provide technical details on each step of the workflow execution.

D.1 Scenario

A patient Bob, who has a restricted mobility and stays more or less at home, has subscribed a monitoring service that assures assistance in case of illness. Bob's health status is monitored by several sensors, e.g. temperature or pulse sensors. A monitoring application analyses the data so that a workflow is initiated to schedule a house visit to the patient and provide medication whenever necessary. The first step of this workflow consists in finding physicians available to make a house visit. A list of physicians is selected for the task by means of a discovery service. Once one of them has committed to perform the task, the assigned physician visits the patient and issues a prescription. The physician may require access to the patient's medical record which is stored in a Medical Information Portal (MIP) in order to store the prescription. The latter is issued electronically and sent automatically to the nearest pharmacy. As soon as the medication is ready for dispatch, a social worker is assigned for delivery to the patient. The prototype demonstration scenario is depicted in figure D.1.

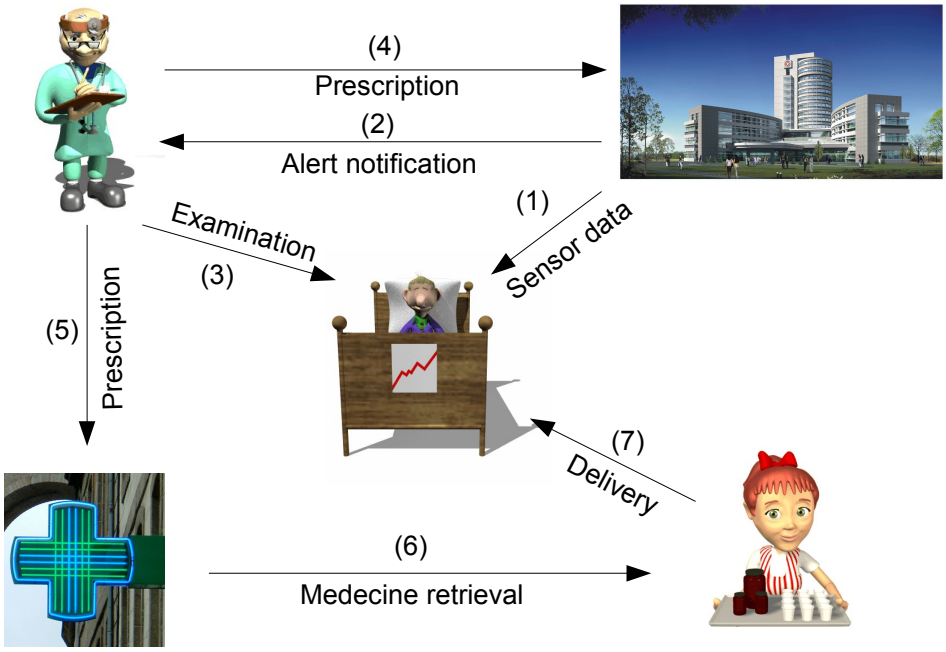

Figure D.1: Prototype demonstration scenario

D.2 Demonstrator execution

The different actors involved in the scenario presented in section D.1 can be seen as business partners assigned to the vertices of the pervasive workflow depicted in figure C.4. We detail in this section the execution steps of this workflow.

D.2.1 Workflow instantiation

The workflow is initiated by a medical center when a patient is in need of medical care. The medical center thus assigns a physician to the case using a discovery service mechanism and builds a workflow message that it sends to the discovered physician. The workflow instance is here identified using the patient's identity.

D.2.2 Doctor Vertex

Upon receipt of a new assignment the physician being logged on his healthcare application receives a new message in his mail box, as depicted in figures D.2 and D.3. He is able to browse the patient's medical record and add a prescription once the patient examination is finished as shown in figures D.4 and D.5. The prescription issued by the physician is transferred to the

Figure D.2: Medical Information Portal login

engine wrapper and handled as any workflow data. A pharmacy having the required medicines in stock is then assigned to the sequel of the workflow by the physician.

D.2.3 Pharmacist Vertex

Upon receipt of the request the pharmacist assigned to the case gets a new item in his "to-do" list and prepares the medicines specified by the physician as depicted in figure D.6. Once the prescribed medicines are ready for dispatch, he assigns a social worker to deliver the latter to the patient.

D.2.4 Social worker Vertex

The social worker assigned to the case retrieves the medicines and delivers them to the patient.

Figure D.3: Alert received by a physician

Figure D.4: Access to patient data

Figure D.5: Prescription management page

Figure D.6: Pharmacist interface

Appendix E

Curriculum Vitae

Frederic Montagut
8, rue Max Jacob
81100 Castres, France

Phone: +33 (0)4 97 21 96 55
Mobile: +33 (0)6 86 73 67 43
fred.montagut@googlemail.com

Date of birth: May, 7^{th} 1981
Citizenship: French

Education

- Oct. 2004 - Oct. 2007. **Ph.D. in Computer Science**, Ecole Nationale Superieure des Telecommunications, Paris. joint programme with Institut Eurecom, Sophia-Antipolis, France.
  Supervisor: Pr. Refik Molva, Institut Eurecom.
  Thesis contributions: design of a distributed workflow management system, design of solutions to provide transactional consistency and security within distributed workflow executions.

- Sept. 2001 - Sept. 2004. **Engineering diploma in Telecommunications**, Institut National des Telecommunications, Evry, France. joint programme with Institut Eurecom.
  Relevant courses: Network processing, Network Security, Web technologies, Project Management.

- Oct. 2003 - Sept. 2004. **M.Sc in Networks and Distributed Systems**, Université de Nice Sophia-Antipolis, France. joint programme with Institut Eurecom.

Research Experience

- Oct. 2004 - Oct. 2007. **Research Associate at SAP Labs France SAS** Sophia-Antipolis, France.
  I was part of the Security and Trust Research program at SAP Research, as a PhD student funded by the French CIFRE programme. My research activities focused on topics related to the execution of decentralized workflows including architecture design, security and transactional consistency. The research results were implemented using Web services technologies.

- Mar. 2004 - Aug. 2004. **Master's Student, Intern at BMW Forschung und Technik, GmbH** Munich, Germany.
  I wrote my Master's thesis during my six-month stay within the Wireless Technology group at BMW on the topic "Predictive model for seamless handover". The project I was involved in focused on wireless communications between cars and a wireless LAN infrastructure. In the scope of this thesis I designed a signal strength prediction algorithm for wireless LAN connections based on fuzzy logic theory in order to provide an estimation of a connection lifetime. The result of this work was implemented using Matlab.

Research Projects

- From 2007. **R4eGov**: *Towards e-Administration in the large*. EU IP Project involving twenty institutions. This project consists in designing a secure collaborative framework supporting the execution of business processes involving different European institutions. This project capitalizes on the results of my thesis with respect to transactional coordination and security solutions.
  http://www.r4egov.info/

- Oct. 2004 - Dec. 2006. **MOSQUITO**: *Mobile Workers' Secure Business Applications in Ubiquitous Environments*. EU STREP Project involving seven institutions. This project focused on the integration of contextual information within business applications executed in the pervasive setting. I was leading the workpackage dealing with the design of a Web services based middleware. My contributions were based on the results of my thesis.
  http://www.mosquito-online.org

Academic teaching and supervision

- Supervision of some labs on IPsec protocol at Institut Eurecom

- Supervision of two master's students at SAP Labs France

Software development activities

- 2004-2007. In the scope of my PhD thesis, I implemented a prototype presenting my research results. This prototype mainly consists of four modules: a decentralized workflow engine, a framework supporting the execution of transactional workflows, a set of security protocols for the execution of distributed workflows and a SVG based workflow monitoring tool. These development tasks used the following technologies: Java / Tomcat, Axis Web Services framework (synchronous and asynchronous WS calls), BPEL, ActiveBPEL workflow engine, WS-Coordination, OWL-S, WS-Discovery, Identity based Encryption Java Libraries, SVG, Ajax.

- 2004. In the scope of my master's thesis, I implemented a simulation framework based on Matlab Simulink for the algorithm I designed.

- 2001-2004. Student projects including a C++/GTK graphical interface for a domino game and a Java course scheduler for teachers and students.

Work experiences

- Oct. 2004 - Oct. 2007. **Research Associate at SAP Labs France SAS** Sophia-Antipolis, France.
  I was mainly involved in two European IST Research projects namely MOSQUITO and R4eGov to which I contributed with my PhD thesis results. My work also involved presentations to SAP internal and external events as well as identifying funding opportunities (I took part in the proposal writing process of the project NEUROLOG funded by the French government).

- Mar. 2004 - Aug. 2004. **Master's Student, Intern at BMW Forschung und Technik, GmbH** Munich, Germany.
  Six-month internship in the Wireless Technology group at BMW.

- Summer 2002. **Intern at Cap Laser** Castres, France.
  Set-up, Maintenance and Administration of computer parks.

Conference Papers

"**A secure public sector workflow management system**", S. Crosta, F. Montagut, J.C. Pazzaglia, Y. Reznichenko, M. Rits, A. Schaad, ACSA 2005, 21st Annual Computer Security Applications Conference - Case Study session, December 5-9, 2005, Tucson, USA.

"**Enabling pervasive execution of workflows**", F. Montagut and R. Molva, in the proceedings of CollaborateCom 2005, 1st IEEE International Conference on Collaborative Computing:Networking, Applications and Worksharing, December 19-21, 2005, San Jose, USA.

"**Augmenting Web services composition with transactional requirements**", F. Montagut and R. Molva, in the proceedings of ICWS 2006, IEEE International Conference on Web Services, September 18-22, 2006, Chicago, USA. **This paper received the conference runners-up award**.

"**Towards transactional pervasive workflows**", F. Montagut and R. Molva, in the proceedings of EDOC 2006, 10th IEEE International EDOC Conference "The Enterprise Computing Conference", 16-20 October 2006, Hong-Kong.

"**Enforcing Integrity of Execution in Distributed Workflow Management Systems**", F. Montagut and R. Molva, in the proceedings of SCC 2007, 2007 IEEE International Conference on Services Computing, 9-13 July 2007, Salt Lake City , USA.

"**Traceability and Integrity of Execution in Distributed Workflow Management Systems**", F. Montagut and R. Molva, in the proceedings of ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007.

Journal Papers

"**The Pervasive Workflow: A Decentralized Workflow System Supporting Long Running Transactions**", F. Montagut, R. Molva and S. Golega, to appear in IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews.

"**Automating the composition of transactional Web services**", F. Montagut, R. Molva and S. Golega, To appear in International Journal on Web Services Research, Idea Publishing.

Book chapters

"**Utilisation des informations contextuelles pour assurer la sécurité d'un processus collaboratif distribué : un exemple dans l'e-Santé**", J.-C. Pazzaglia, K. Wrona, A. Laube, F. Montagut, L. Gomez, Y. Roudier, and S. Trabelsi. OFTA, Paris, France, 2007.

"**Automating the composition of transactional Web services**", F. Montagut, R. Molva and S. Golega, in "Managing Web Services Quality: Measuring Outcomes and Effectiveness", IGI Global, to appear in 2008.

Patents (Filed with INPI by SAP)

- 05.04.2007 Enforcing Integrity of Execution in Distributed Workflow Management Systems
- 30.03.2007 Architecture for seamless display of application event
- 30.01.2007 Method and Apparatus for Sealing Electronic Documents
- 13.09.2006 Towards Transactional Pervasive Workflows
- 05.04.2006 Augmenting Web Services Composition with Transactional Requirements
- 14.12.2005 Automatic Robots Assembly
- 29.09.2005 Support for execution of workflow in pervasive environments

Professional activities

- PC member of the 11th IEEE International EDOC Conference (EDOC 2007)
- Editorial Advisory Board of the book *"Managing Web Services Quality: Measuring Outcomes and Effectiveness"*, IGI Global, USA

Languages

Native French, Fluent English, basic knowledge of German.

Computer Skills

- Unix and Windows 9x
- C/C++, Java, Php, HTML, SQL
- Eclipse, Aptana, Visual C++
- Ms Office, Open Office, Latex, Matlab

Honors and Awards

- International Conference on Web Services 2006 (ICWS'06) "runners-up" award
- 2007 SAP Research Award for exceptional contributions to SAP

Extracurricular activities

- Sports: Ski (French "chamois de bronze"), Chess (Elo 1500), Golf.
- Music (Original Soundtracks), Reading, Cinema, Internet

Bibliography

[2PC] Open system interconnection- distributed transaction processing (osi-tp) model. iso is 100261.

[Aa05] M. Abbott and al. Business transaction protocol, 2005.

[AAA+95] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Gunthor, and M. Kamath. Exotica/fmqm: A persistent message-based architecture for distributed workflow managemen. In *Proceedings of the IFIP WG8.1 Working Conference on Information Systems Development for Decentralized Organizations. Trondheim, Norway, August 1995.*, 1995.

[AAA+96] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *Proc. 12th International Conference on Data Engineering, New Orleans*, February 1996.

[AAAM97] G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. *submitted to IEEE Expert*, 1997.

[ACJT00] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. *Lecture Notes in Computer Science*, 1880:255–271, 2000.

[ACKM03] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web services: Concepts, Architectures, and Applications*. Springer Verlag, 2003.

[ACM01] V. Atluri, S. A. Chun, and P. Mazzoleni. A chinese wall security model for decentralized workflow systems. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 48–57, 2001.

[ADK+05] V. Agarwal, K. Dasgupta, N. Karnik, A. Kumar, A. Kundu, S. Mittal, and B. Srivastava. A service creation environment based on end to end composition of web services. In *Proceedings of the WWW conference*, 2005.

[AEwe07] Active Endpoints www.active endpoints.com. ACTIVE BPEL Engine, 2007.

[AHMS06] P. Allen, S. Higgins, P. McRae, and H. Schlamann. *Service Orientation: Winning Strategies and Best Practices*. Cambridge University Press, Cambridge, UK, 2006.

[Aja] Ajax: Asynchronous javascript and xml.

[AkH96] Vijayalakshmi Atluri and Wei kuang Huang. An authorization model for workflows. In *Proceedings of the Fourth European Symposium on Research in Computer Security*, pages 44–64, 1996.

[AM97] G. Alonso and C. Mohan. Workflow management systems: The next generation of distributed procesing tools. *Advanced Transaction Models and Architectures (Jajodia and L. Kerschberg, eds.)*, pages 35–62, 1997.

[ASF06] Apache Software Fundation. XMLBeans, July 2006.

[ASKP00] G.-J. Ahn, R. Sandhu, M. H. Kang, and J. S. Park. Injecting RBAC to secure a web-based workflow system. In *Proceedings of 5th ACM Workshop on Role-Based Access Control*, pages 1–10, 2000.

[AXI] Axis - apache software foundation. http://ws.apache.org/axis/.

[Ba00] J. J. Barton and al. W3c - soap messages with attachments, 2000. http://www.w3.org/TR/SOAP-attachments.

[Ba05] J. Beatty and al. Web services dynamic discovery (ws-discovery), April 2005.

[BCCT05] M. Brambilla, S. Ceri, S. Comai, and C. Tziviskou. Exception handling in workflow-driven web applications. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 170–179, New York, NY, USA, 2005. ACM Press.

[BCP06] E. Bertino, J. Crampton, and F. Paci. Access control and authorization constraints for ws-bpel. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 275–284, Washington, DC, USA, 2006. IEEE Computer Society.

[BDS05] B. Benatallah, M. Dumas, and Q. Z. Sheng. Facilitating the rapid development and scalable orchestration of composite web services. *Distributed and Parallel Databases*, 17(1):5–37, 2005.

[BF01] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, CA, USA*, pages 213–229, 2001.

[BFA99] E. Bertino, E. Ferrari, and V. Atluri. The specification and enforcement of authorization constraints in workflow management systems. *ACM Trans. Inf. Syst. Secur.*, 2(1):65–104, 1999.

[BGP05] S. Bhiri, C. Godart, and O. Perrin. Reliable web services composition using a transactional approach. In *Proc. of the IEEE International Conference on e-Technology, e-Commerce, and e-Services (EEE), Hong Kong, China*, 2005.

[biz05] Microsoft Biztalk - http://www.microsoft.com/biztalk/, 2005.

[BM04] D. J. Bradley and D. P. Maher. The nemo p2p service orchestration framework. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004.

[BM05] W. Bagga and R. Molva. Policy-based cryptography and applications. In *FC' 2005, 9th International Conference on Financial Cryptography and Data Security, Roseau, The Commonwealth of Dominica*, Mar 2005.

[BMNR03] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath. Web services on mobile devices - implementation and experience. In *Fifth IEEE Workshop on Mobile Computing Systems and Applications*, 2003.

[BMR94] D. Barbara, S. Mehrotra, and M. Rusinkiewicz. Incas: A computation model for dynamic workflows in autonomous distributed environments. Technical report, Department of Computer Science, University of Houston, May 1994.

[BMR96] D. Barbara, S. Mehrotra, and M. Rusinkiewicz. Incas: Managing dynamic workflows in distributed environments. *Journal of Database Management*, 7(1), 1996.

[BN89] D. F. C. Brewer and M. J. Nash. The chinese wall security policy. In *IEEE Symposium on Security and Privacy*, pages 206 –214, 1989.

[Bou07] Bouncy Castle - http://www.bouncycastle.org/, 2007.

[BPE] Business process execution language for web sevices. http://www.ibm.com/developerworks/library/ws-bpel/.

[BPG05] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In *Proc. of the 14th international conference on World Wide Web*, 2005.

[BS06] J. Bloomberg and R. Schmelzer. *Service Orient or Be Doomed!: How Service Orientation Will Change Your Business*. WILEY, Hoboken, New Yersey, USA, 2006.

[Ca04] L. Clement and al. Uddi version 3.0.2, October 2004.

[CAA04] S. Ae Chun, V. Atluri, and N. R. Adam. Policy-based web service composition. In *RIDE '04: Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04)*, pages 85–92, Washington, DC, USA, 2004. IEEE Computer Society.

[CCKM05] G. Chafle, S. Chandra, P. Kankar, and V. Mann. Handling faults in decentralized orchestration of composite web services. In *Lecture Notes in Computer Science, Volume 3826*, 2005.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1, http://www.w3.org/TR/wsdl 2001.

[CKM+03] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Commun. ACM*, 46(10), 2003.

[CL04] D. Chakraborty and H. Lei. Pervasive enablement of business processes. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications 2004. PerCom 2004*, pages 87– 97, March 2004.

[CLW05] S. Chou, A. Liu, and C. Wu. Preventing information leakage within workflows that execute among competing organizations. *J. Syst. Softw.*, 75(1-2):109–123, 2005.

[Coa98] Workflow Management Coalition. Workflow security considerations - white paper. Technical Report WFMC-TC-1019, Workflow Management Coalition, 1998.

[CR04] A. Cichocki and M. Rusinkiewicz. Providing transactional properties for migrating workflows. *Mob. Netw. Appl.*, 9(5):473–480, 2004.

[CYT04] H. Chu, C. You, and C. Teng. Challenges: Wireless web services. In *ICPADS '04: Proceedings of the Parallel and Distributed Systems, Tenth International Conference on (ICPADS'04)*, page 657, Washington, DC, USA, 2004. IEEE Computer Society.

[Den01] P. J. Denning. *Invisible Future: The Seamless Integration of Technology Into Everyday Life*. McGraw-Hill, 2001.

[DGA04] P. Doshi, R. Goodwin, and R. Akkiraju. Parameterized semantic matching for workflow composition. Technical Report RC23133, IBM, 2004.

[DOM05] Document object model - http://www.w3.org/dom/, 2005.

[DSBW+06] J. Davis, D. Sow, D. Bourges-Waldegg, C. Jie Guo, C. Hoertnag, M. Stolze, B. White Eagle, and Y. Yin. Supporting mobile business workflow with commune. *wmcsa*, 0:10–18, 2006.

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

[ERS99] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth. *Management of heterogeneous and autonomous database systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[FDDB05] M. C. Fauvet, H. Duarte, M. Dumas, and B. Benatallah. Handling transactional properties in web service composition. In *Web Information Systems Engineering - WISE 2005, 6th International Conference on Web Information Systems Engineering, New York, NY, USA*, pages 273–289, 2005.

[FHF05] C. K. Fung, P. C. K. Hung, and D. H. Folger. Achieving survivability in business process execution language for web services (bpel) with exception-flows. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 68–74, Washington, DC, USA, 2005. IEEE Computer Society.

[GAC+97] E. Gokkoca, M. Altinel, I. Cingil, N. Tatbul, P. Koksal, and A. Dogac. Design and implementation of a distributed workflow enactment service. In *COOPIS '97: Proceedings of the Second IFCIS International Conference on Cooperative Information Systems*, pages 89–98, Washington, DC, USA, 1997. IEEE Computer Society.

[GC02] N. Gioldasis and S. Christodoulakis. Utml: Unified transaction modeling language. In *WISE '02: Proceedings of the 3rd International Conference on Web Information Systems Engineering*, pages 115–126, Washington, DC, USA, 2002. IEEE Computer Society.

[GFJK03] P. Greenfield, A. Fekete, J. Jang, and D. Kuo. Compensation is not enough. In *Proc. of the 7th International Enterprise Distributed Object Computing Conference (EDOC'03)*, 2003.

[GHJV07] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2007.

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

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

[Gud04] M. Gudgin. Secure, reliable, transacted; innovation in web services architecture. In *Proc. of the ACM International Conference on Management of Data, Paris, France*, 2004.

[HK03] P. C. K. Hung and K. Karlapalem. A secure workflow model. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers*, pages 33–41, 2003.

[HMR07] J. El Haddad, M. Manouvrier, and M. Rukoz. A Hierarchical Model for Transactional Web Service Composition in P2P Networks. In *ICWS 2007, IEEE International Conference on Web Services, July 9-13, 2006, Salt Lake City, USA*, July 2007.

[Hol95] D. Hollingsworth. The workflow reference model. Technical Report WFMC-TC-1003, The Workflow Management Coalition, 1995.

[Hun04] P. C. K. Hung. From conflict of interest to separation of duties in ws-policy for web services matchmaking process. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 3*, page 30066.2, Washington, DC, USA, 2004. IEEE Computer Society.

[Jav] J2ee 1.5 (java 2 enterprise edition).

[JSP05] Java server pages - http://java.sun.com/products/jsp/index.jsp, 2005.

[KH03] J. Kong and X. Hong. Anodr: anonymous on demand routing with untraceable routes for mobile ad-hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 291–302, 2003.

[KH05] S. Karch and L. Heilig. *SAP NetWeaver Roadmap*. Galileo Press, 2005.

[Kha08] K. Khan. *Managing Web Services Quality: Measuring Outcomes and Effectiveness*. IGI Global, Hershey, PA, USA, 2008.

[KM03] H. Koshutanski and F. Massacci. An access control framework for business processes for web services. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 15–24, New York, NY, USA, 2003. ACM Press.

[KPF01] M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 66–74, 2001.

[KR01] K. Knorr and S. Rohrig. Security requirements of e-business processes. In *Proceedings of the IFIP Conference on Towards The E-Society: E-Commerce, E-Business, E-Government*, pages 73–86, 2001.

[KSY02] L. Korba, R. Song, and G. Yee. Anonymous communications for mobile agents. In *MATA '02: Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications*, pages 171–181, London, UK, 2002. Springer-Verlag.

[LA90] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Morgan Kaufmann, 1990.

[La05a] D. Langworthy and al. Ws-atomictransaction, 2005.

[La05b] D. Langworthy and al. Ws-businessactivity, 2005.

[La05c] D. Langworthy and al. Ws-coordination, 2005.

[LCY+04] R. Liu, F. Chen, H. Yang, W. C. Chu, and Y. Lai. Agent-based web services evolution for pervasive computing. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)*, pages 726–731, Washington, DC, USA, 2004. IEEE Computer Society.

[LH03] M. Laukkanen and H. Helin. Composing workflows of semantic web services. In *Workshop on Web Services and Agent-based Engineering*, 2003.

[Lit03] M. Little. Transactions and web services. *Commun. ACM*, 46(10):49–54, 2003.

[Liu05] W. Liu. Trustworthy service selection and composition - reducing the entropy of service-oriented web. In *3rd IEEE International Conference on Industrial Informatics, 2005. INDIN '05*, pages 104–109, Perth, Australia, 2005. IEEE Computer Society.

[LMP01] S. Loureiro, R. Molva, and A. Pannetrat. Secure data collection with updates. *Electronic Commerce Research, Volume 1 Nř1-2, February/March 2001*, 2001.

[LP03] C. Li and C. Pahl. Security in the web services framework. In *ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies*, pages 481–486. Trinity College Dublin, 2003.

[LW05] Z. Leo Liang and R. K. Wong. A lightweight mobile platform for business services networks. In *BSN '05: Proceedings of the IEEE EEE05 international workshop on Business services networks*, pages 12–12, Piscataway, NJ, USA, 2005. IEEE Press.

[LZ04] B. Limthanmaphon and Y. Zhang. Web service composition transaction management. In *ADC '04: Proceedings of the 15th Australasian database conference*, pages 171–179, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[MBB+03] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal*, 12(1):59–85, 2003.

[MM03] D. J. Mandell and S. A. McIlraith. Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In *Proceedings of International Semantic Web Conference*, October 2003.

[MM04] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[MM05] F. Montagut and R. Molva. Enabling pervasive execution of workflows. In *Proceedings of the 1st IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom*, 2005.

[MM06a] F. Montagut and R. Molva. Augmenting Web services composition with transactional requirements. In *ICWS 2006, IEEE International Conference on Web Services, September 18-22, 2006, Chicago, USA*, Sep 2006.

[MM06b] F. Montagut and R. Molva. Towards transactional pervasive workflows. In *EDOC 2006, 10th IEEE International EDOC Conference "The Enterprise Computing Conference", 16-20 October 2006, Hong-Kong*, Oct 2006.

[MM07a] F. Montagut and R. Molva. Enforcing integrity of execution in distributed workflow management systems. In *SCC 2007, 2007 International Conference on Services Computing, July 9-13 2007, Salt Lake City, USA*, July 2007.

[MM07b] F. Montagut and R. Molva. Traceability and integrity of execution in distributed workflow management systems. In *ESORICS 2007, European Symposium On Research In Computer Security,Dresden, Germany, September 24 - 26, 2007*, Sep 2007.

[MMG08a] F. Montagut, R. Molva, and S. T. Golega. Automating the composition of transactional web services. *To appear in International Journal of Web Services Research, Idea Publishing*, 2008.

[MMG08b] F. Montagut, R. Molva, and S. T. Golega. The pervasive workflow: A decentralized workflow system supporting long running transactions. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews. Special Issue on Enterprise Service Computing and Industrial Applications (to appear)*, 2008.

[mos] Eu ist project mosquito. http://www.mosquito-online.org.

[MRSK92] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A transaction model for multidatabase systems. In *Proc. of the 12th IEEE International Conference on Distributed Computing Systems (ICDCS92)*, 1992.

[MTR02] T. Mikalsen, S. Tai, and I. Rouvellou. Transactional attitudes: Reliable composition of autonomous web services. In *Workshop on Dependable Middleware-based Systems*, 2002.

[MWW+98] P. Muth, D. Wodtke, J. Weisenfels, A. Kotz Dittrich, and G. Weikum. From centralized workflow specification to distributed workflow execution. *Journal of Intelligent Information Systems*, 10(2):159–184, 1998.

[NCS03] M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing composite web services. In *Proceedings of Workshop on Compilers for Parallel Computing*, January 2003.

[NK03] M. G. Nanda and N. Karnik. Synchronization analysis for decentralizing composite web services. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 407–414, 2003.

[NWG96] IETF Network Working Group. Multipurpose Internet Mail Extensions RFC 2045, November 1996.

[ODD04] L. Owens, A. Duffy, and T. Dowling. An identity based encryption system. In *PPPJ '04: Proceedings of the 3rd international symposium on Principles and practice of programming in Java*, pages 154–159. Trinity College Dublin, 2004.

[OWL03] OWL-S specifications, http://www.daml.org/services 2003.

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

[Pat02] K. Paterson. Id-based signatures from pairings on elliptic curves. *Electronics Letters*, 38(18):1025–1026, 2002.

[PH03] J. S. Park and J. Hwang. Role-based access control for collaborative enterprise in peer-to-peer computing environments. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 93–99, New York, NY, USA, 2003. ACM Press.

[PWL+07] J.-C. Pazzaglia, K. Wrona, A. Laube, F. Montagut, L. Gomez, Y. Roudier, and S. Trabelsi. *Utilisation des informations contextuelles pour assurer la sécurité d'un processus collaboratif distribué : un exemple dans l'e-Santé*, chapter 14, pages 345–370. OFTA, Paris, France, 2007.

[RM04] A. Ranganathan and S. McFaddin. Using workflows to coordinate web services in pervasive computing environments. In *Proceedings of the IEEE International Conference on Web Services 2004*, pages 288–295, July 2004.

[RS95] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In *Modern database systems: the object model, interoperability, and beyond*, 1995.

[RS04] J. Rao and X. Su. A survey of automated web service composition methods. In *First International Workshop on Semantic Web Services and Web Process Composition*, pages 43–54, 2004.

[SABS02] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and isolation for transactional processes. *Database Systems*, 27(1):63–116, 2002.

[SAS99] H. Schuldt, G. Alonso, and H. Schek. Concurrency control and recovery in transactional process management. In *Proc. of the 18th ACM symposium on Principles of Database Systems*, 1999.

[SCFY96] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[SGR97] P. F. Syverson, D. M. Goldschlag, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, USA, 1997.

[SGS+04] J. Sairamesh, S. Goh, I. Stanoi, S. Padmanabhan, and C. S. Li. Disconnected processes, mechanisms and architecture for mobile e-business. *Mob. Netw. Appl.*, 9(6):651–662, 2004.

[SK02] T. Strandenæs and R. Karlsen. Transaction compensation in web services. In *The Norwegian Computer Science Conference*, 2002.

[SLM04] Z. Song, Y. Labrou, and R. Masuoka. Dynamic service discovery and management in task computing. *Mobiquitous*, 00:310–318, 2004.

[SO01] K. A. Schulz and M. E. Orlowska. Architectural issues for cross-organisational b2b interactions. In *21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*, pages 79–87, 2001.

[SOA] Soap - simple object access protocol. http://www.w3.org/TR/SOAP/.

[SVG07] W3c - scalable vector graphics (svg), 2007.

[SWSS03] C. Schuler, R. Weber, H. Schuldt, and H. Schek. Peer-to-peer process execution with osiris. In *First International Conference on Service-Oriented Computing ICSOC (2003)*, 2003.

[SZ97] R. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.

[TAK03] A. R. Tripathi, T. Ahmed, and R. Kumar. Specification of secure distributed collaboration systems. In *ISADS '03: Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*, page 149, 2003.

[TI05] F. Tartanoglu and V. Issarny. Specifying web service recovery support with conversations. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS'05) - Track 7*, page 167.2, Washington, DC, USA, 2005. IEEE Computer Society.

[TIRL03] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Coordinated forward error recovery for composite web services. In *22nd Symposium on Reliable Distributed Systems (SRDS)*, 2003.

[TKM04] S. Tai, R. Khalaf, and T. Mikalsen. Composition of coordinated web services. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 294–310, New York, NY, USA, 2004.

[TLC05] W. T. Tsai, X. Liu, and Y. Chen. Distributed policy specification and enforcement in service-oriented business systems. In *ICEBE '05: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 10–17, Washington, DC, USA, 2005. IEEE Computer Society.

[TLJ03] S. Tang, C. Liebetruth, and M. C. Jaeger. The owl-s matcher software, http://flp.cs.tu-berlin.de/, 2003.

[Tom] Apache software foundation - tomcat application server. http://tomcat.apache.org/.

[TPR06] S. Trabelsi, J.-C. Pazzaglia, and Y. Roudier. Enabling secure discovery in a pervasive environment. In *SPC 2006, 3rd International Conference on Security in Pervasive Computing, April 18 - 21, 2006, York, UK - also published in LNCS Volume 3934*, Apr 2006.

[UML] Object management group - unified modeling language specification.

[UTF03] Ietf - utf-8, a transformation format of iso 10646 - rfc 3629, 2003.

[VBS04] J. Vidal, P. Buhler, and C. Stahl. Multiagent systems with workflows. *Internet Computing, 8(1)*, 8(1):76–82, 2004.

[Wan00] A. I. Wang. Using software agents to support evolution of distributed workflow models. In *Proc. International ICSC Symposium on Interactive and Collaborative Computing (ICC'2000)*, 2000.

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–110, September 1991.

[WKRL06] M. Wimmer, A. Kemper, M. Rits, and V. Lotz. Consolidating the access control of composite applications and workflows. In *DBSec*, pages 44–59, 2006.

[WS92] G. Weikum and H.-J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In *Database Transaction Models for Advanced Applications*, pages 515–553. Morgan Kaufmann Publishers Inc, 1992.

[WSa] W3c - web services architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[WSC] W3c - web services choreography description language. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/.

[WSR05] Web Services Reliable Messaging Protocol - http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf, 2005.

[WSS] Ws-security - web services security. http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

[WT04] R. Wonohoesodo and Z. Tari. A role based access control for web services. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 49–56, Washington, DC, USA, 2004. IEEE Computer Society.

[WTK02] Web services tool kit for mobile devices, http://www.alphaworks.ibm.com/tech/wstkmd 2002.

[XAC05] OASIS eXtensible Access Control Markup Language (XACML), 2005.

[XML] Xml - extensible markup language. www.w3.org/XML/.

[XML04] XML schema - http://www.w3.org/XML/Schema, 2004.

[YHLC05] S. J. H. Yang, J. S. F. Hsieh, B. C. W. Lan, and J.-Y. Chung. Composition and evaluation of trustworthy web services. In *BSN '05: Proceedings of the IEEE EEE05 international workshop on Business services networks*, pages 5–5, Piscataway, NJ, USA, 2005. IEEE Press.