N° d'ordre: 3858

THÈSE

présentée

devant l'Université de Rennes 1

pour obtenir

le grade de : Docteur de l'Université de Rennes 1
Mention Informatique

par

Debmalya Biswas

Équipe d'accueil : DISTRIBCOM - IRISA
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Visibilité en Systèmes Hiérarchiques

Visibility in Hierarchical Systems

À soutenir le 19 Janvier 2009 devant la commission d'examen

MMe. :	Françoise	André	Président
MM. :	Jean-Bernard	Stefani	Rapporteurs
	Franck	Cassez	
MM. :	Jean-Christophe	Pazzaglia	Examinateurs
	Achour	Mostefaoui	
M. :	Albert	Benveniste	Directeur de Thèse
M. :	Blaise	Genest	Encadrant de Thèse

Remerciements

Je remercie Françoise André, i me fait l'honneur de présider ce jury.

Je remercie Jean-Bernard Stefani, 1, et Franck Cassez, 2, d'avoir bien voulu accepter la charge de rapporteur.

Je remercie Jean-Christophe Pazzaglia, 1, et Achour Mostefaoui, 2, d'avoir bien voulu juger ce travail.

Je remercie enfin , q, u, qui a dirigé ma thèse.

Contents

Introduction en Français

Les systèmes distribués ont beaucoup évolué durant les trois dernières décennies, et plus encore ces dix dernières années, la popularité croissante de l'Internet en faisant un élément central dans le développement de logiciels modernes. Les systèmes distribués présentent de nombreux avantages, notamment en termes de performance, d'autonomie et d'économie.

Les deux paradigmes à la base du développement croissant des systèmes distribués sont : Les services Web et les systèmes Pair-à-Pair (P2P). Les services Web permettent de présenter de façon uniforme les services proposés par une organisation, afin que les clients puissent les utiliser de manières automatiques. Les serveurs hébergeant les services ainsi que les clients peuvent être implémentés à l'aide de différentes plateformes logicielles, conduisant ainsi à un haut niveau d'interopérabilité. Les systèmes P2P, eux, n'utilisent pas le principe de clients/serveurs, mais celui de noeuds « pairs » égaux, pouvant aussi bien être clients que serveurs d'autres pairs du réseau. Les réseaux P2P permettent à un pair de se connecter via des connections fortement ad-hoc, autorisant ainsi un haut niveau d'évolutivité. L'évolutivité et l'interopérabilité sont les fondations des systèmes distribués, et permettent le développement et le déploiement de systèmes distribués larges et complexes, mettant en jeux des centaines d'entités.

Une fois de tels systèmes distribués construits, vient alors la tâche de les exploiter correctement afin qu'ils puissent tenir leurs promesses. Bien que les services Web et les systèmes P2P permettent à une entité de joindre, de partager, d'offrir et d'utiliser des ressources avec facilité, cette même facilité les rend, entre autres, vulnérables aux risques d'être exploité par des entités malveillantes, ainsi que de subir de multiples échecs arbitraires. Un effort de recherche considérable à été fournis afin d'obtenir une sécurité efficace ainsi que des mécanismes de récupérations à partir d'échecs. Cependant, la plupart des approches proposées ne considèrent que l'un ou l'autre de ces aspects. Le problème, lorsque l'on souhaite conjuguer les deux, est que leurs exigences sont d'une certaine manière contradictoires. Tandis que la sécurité souhaite qu'une entité ne fournisse qu'un accès limité à ses ressources (de préférence le minimum nécessaire pour accomplir son but / sa fonction), un mécanisme de récupération d'erreurs a besoin d'accéder à l'intégralité du cycle d'exécution des autres entités, ainsi qu'à leurs spécifications fonctionnelles, etc... Dans le cadre de cette thèse, nous proposons la notion conceptuelle générique de « visibilité » pour représenter/saisir de tels besoins et restrictions d'accès pour des systèmes distribués spécifiques, appelés systèmes hiérarchiques.

Visibilité en systèmes hiérarchiques

Les systèmes hiérarchiques fournissent un élégant mécanisme pour analyser la fonctionnalité d'un système dans des différents niveaux d'abstraction. Ils sont habituellement construits dans un mode "top-down" ou "bottom-up", niveau par niveau. Dans la construction "top-down", une entité complexe est décomposé en éléments plus simples (enfants), alors que dans la construction "bottom-up", des simples entités sont fusionnées dans une entité complexe (parent). En conséquence, par construction, la visibilité des entités dans un système hiérarchique est limitée aux niveaux adjacentes (parent-enfants).

Cette visibilité n'est souvent pas suffisante pour les scénarios de la vie réelle. Par exemple, dans le cadre d'un système pour la gestion de la chaîne logistique, [LTS98] déclare le besoin de visibilité à travers les niveaux comme : "Les informations requises par les entités en aval sont principalement du matériel et des renseignements sur la disponibilité de la capacité de leurs fournisseurs. Les informations acquises par une entité en amont sont des informations sur la demande des clients et leurs ordres. La profondeur de pénétration de l'information peut être spécifiée à des degrés divers, par exemple : isolé, un niveau vers le haut, vers le haut à deux niveaux, un niveau vers le bas, vers le bas deux niveaux, etcetera". Les aspects non-fonctionnelles, tels que les transactions, le suivi, l'interaction-utilisateur, etc. font également appel à des entités ayant plus de visibilité sur ces ancêtres, descendants, frères.

Permettre l'interaction arbitraire entre les entités hiérarchiques, sans aucune restriction, n'est pas non plus une solution acceptable. Dans un environnement hétérogène et dynamique, les questions de sécurité telles que la confiance, la vie privée, autonomie, etc. forcent une entité à être sélective dans les interactions qu'elle a avec les autres. Par exemple, [BKK05] considère la modélisation hiérarchique de hypertexte dans le "World Wide Web" (WWW). Le WWW peut être modélisée comme un grand graphe avec des pages Web comme des noeuds et des hyperliens comme des arrêts. En ce qui concerne un site S, son graphe correspondant se décompose en pages fournies par les départements $d_1, d_2,\ cdots, d_n$ et en liens entre eux. Compte tenu de cela, le graphe peut être hiérarchiquement décomposé comme : (i) Par département : Les pages (et les hyperliens correspondants) appartenant à un département d sont regroupées. (ii) Par la structure de la page : Ici, on considère une page comme une abstraction de son propre chef. Cela peut être utile pour un auteur qui veut modifier une page tout en ayant une vue d'ensemble de sa structure. Avec la représentation ci-dessus, on peut imaginer la nécessité des certains noeuds d'être cachés ou visibles à l'égard d'un groupe donné. La nécessité d'une telle abstraction a été identifié dans [BKK05] comme : "Un modèle de données d'un graphe hiérarchique doit permettre la représentation de ce type d'informations (hiérarchie, visibilité), et de faire respecter les possibles contraintes qui en découlent (par exemple , interdire les bords au noeuds cachés)."

Dans cette thèse, nous nous intéressons à deux sous-problèmes de la visibilité des systèmes hiérarchiques. Dans la partie I, nous considérons le problème de définir un modèle de visibilité, compte tenu des besoins de visibilité et des restrictions des différentes entités dans une hiérarchie. La partie II traite le problème orthogonal de la détermina-

tion des besoins de visibilité de la hiérarchie étant donné une propriété spécifique. Le chapitre 1 présente les définitions préliminaires et les concepts nécessaires pour le reste de cette thèse.

Partie I

La nécessité d'un modèle de visibilité ainsi que les attentes de celui-ci sont analogues à celle de toute abstraction ou modèle d'un système complexe. Le modèle de visibilité proposé devrait être en mesure de capturer toutes les caractéristiques de visibilité du système sous-jacent, ce qui permet en même temps une représentation compacte. En s'appuyant sur le modèle, il devrait être également possible d'identifier, de spécifier, et de vérifier la visibilité des propriétés et des relations entre les entités de la hiérarchie. Enfin, un modèle de visibilité devrait être en mesure de co-exister avec d'autres systèmes fonctionnels / non-fonctionnelles, ou il devrait être possible d'interposer ces aspects dans le modèle de visibilité.

Étant donné la diversité des systèmes hiérarchiques, il serait irréaliste de tenter de concevoir un modèle de visibilité unique, qui serait suffisamment générique pour tenir compte de tous les paramètres, et qui ne serait pas trop lourd à mettre en place. Nous étudions ainsi des modèles de visibilité pour deux types de structures dans les chapitres 2 et 3, les Communautés P2P et les Web services compositionels.

Chapitre 2

Les Communautés P2P sont la variante "collaborative" des systèmes P2P, où les communautés/groupes des pairs partagent des intérêts communs (par exemple, Yahoo Groups citeYG) ou travaillent ensemble pour accomplir une tâche spécifique [Gri, AXM]. Dans ce travail nous nous concentrons sur les Communautés P2P basées sur des intérêts partagés.

Les communautés P2P sont souvent organisées hiérarchiquement, ceci correspond à une ontologie sémantique hiérarchique (Fig. 1). La nécessité d'un modèle de visibilité apparaît pour des raisons de sécurité comme la confiance, la confidentialité, l'anonymat, etc. Ces questions forcent un pair (ou une communauté) à être restrictive dans la visibilité qu'il permet aux autres. D'autre part, les communautés ont besoin de grandir et, pour grandir, ils ont besoin de visibilité par rapport aux autres communautés (et par rapport aux pairs dans leur communautés). De même, un pair aimerait également avoir plus de visibilité par rapport aux pairs dans les communautés avec des intérêts connexes. Ainsi, la visibilité dans les Communautés P2P est intermédiaire entre la visibilité individuel par rapport a un pair/communauté et la visibilité totale par rapport aux autres pairs/communautés [IMZI04]. Nous avons besoin d'un modèle adéquat pour capturer et représenter ces situations.

Le principal objectif des Communautés P2P est bien sûr l'efficacité des requêtes et des mécanismes (structurels) de mise à jour. En l'absence d'un coordinateur central, une requête doit être propagée à de multiples pairs/communautés avant qu'une réponse satisfaisante soit trouvée. L'autre caractéristique des Communautés P2P, et ce qui les

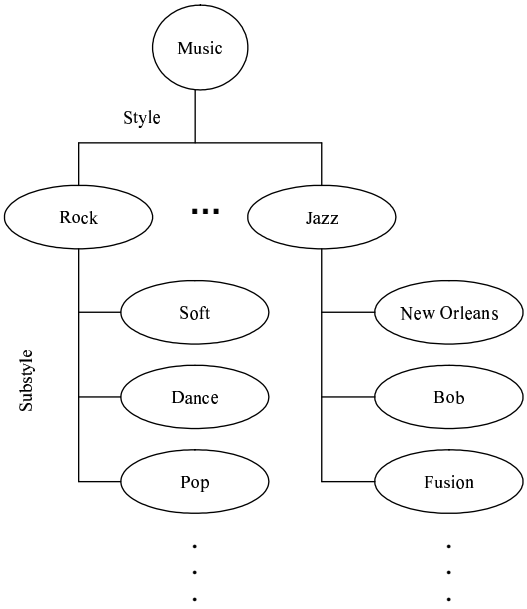

FIGURE 1 – Un exemple d'ontologie hiérarchique.

rend particulièrement intéressant pour nous en tant que modèle, est sa grande dynamique : un pair peut être ajouté ou supprimé d'une communauté, des communautés peuvent être ajoutées ou supprimées, des communautés peuvent être fusionnées ou divisées, et des sous-communautés peuvent devenir parent l'un de l'autre. Dans ce contexte, le modèle de visibilité proposé devrait également faciliter les requêtes et les mises à jour sur les Communautés P2P d'une manière efficace.

À cette fin, nous proposons dans le Chapitre 2 un modèle de graphe avec une visibilité à plusieurs niveaux afin de capturer la visibilité qu'un pair/communauté a par rapport aux autres pairs et communautés. Nous donnons les algorithmes pour effectuer l'évaluation des requêtes et leur mise à jour structurelle d'une façon décentralisée et évolutive. Nous montrons aussi comment d'autres aspects non-fonctionnels, comme un cadre de sécurité fondée sur la confiance, peuvent être mises en oeuvre au dessus du modèle de visibilité.

Ce chapitre de la thèse est paru dans "Debmalya Biswas and Krishnamurthy Vidyasankar. *A Highly Flexible Data Structure for Multi-level Visibility of P2P Communities.* In proceedings of the 9th International Conference on Distributed Computing and Networking (*ICDCN*), Kolkata, India, Lecture Notes in Computer Science vol. 4904 (Springer Verlag), pp. 363-374, 2008".

Chapitre 3

Les services Web [ACKM04] sont devenus la norme de-facto pour la fourniture des services sur le Web. L'aspect intéressant et stimulant des services Web est bien en-

tendu sa promesse de composition automatique. La composition se réfère au processus de combinaison de services existants en une composition plus complexe des services. Le service composite peut agir en tant que composante dans un niveau supérieur de composition, conduisant à une composition hiérarchique. Une composition hiérarchique peut également être formée dans un mode "top-down" quand un service complexe décide de décomposer sa fonctionnalité en services simples, qui peut être décomposé d'avantage à son tour. Nous représentons une composition hiérarchique par un arbre, avec une paire de noeuds "parent-enfant" X et Y respectivement dans la hiérarchie, qui correspond à l'invocation de Y par X.

Par rapport aux Communautés P2P, les compositions des services Web sont plus statiques car elles ne changent pas beaucoup. Cependant, la visibilité de leurs caractéristiques est également beaucoup plus complexe. Pour une composition hiérarchique des services Web, la nécessité de restreindre la visibilité résulte du fait que les services de la composition peuvent être fournis par différents organismes, ce qui conduit à des questions de sécurité telles que la confiance, la confidentialité, l'anonymat, etc. Dans le même temps, l'objectif principal d'une composition est clairement que les services travaillent ensembles vers un objectif commun. Pour ce faire, un service a besoin de visibilité sur les autres services dans la composition, sur d'autres services que son père ou ses enfants, sur d'autres attributs que l'entrée/sortie des valeurs. Ces exigences de visibilité peuvent être poussées par deux types d'objectifs, fonctionnels et non fonctionnels, de la composition hiérarchique. Nous exposons quelques scénarios réels ci-dessous :

- Pour des raisons de sécurité, de nombreux acheteurs en ligne préfèrent donner leur numéro de carte de crédit ou de compte bancaire directement à l'institution financière pour le traitement du paiement. En tant que telle, l'institution financière, qui aurait pu être invoquée quelque part en bas dans la hiérarchie par le vendeur, a besoin de visibilité sur les détails du paiement de son ancêtre (l'acheteur). De même, une compagnie maritime qui fournit des biens directement à l'acheteur est également un exemple classique où la visibilité de l'ancêtre pourrait être nécessaire.
- Les compositions des services Web assurent généralement le fonctionnement des applications d'affaires avec un long temps d'exécution, comme les transactions monétaires. A ce titre, elles doivent être sécurisés, leur progrès doit être suivi, elles devraient être résistantes aux échecs, entre autres. Le fait qu'un service Y ait des restrictions par rapport à un autre service X, étant donnée que X a une visibilité sur Y pour des raisons de sécurité, peut être influencé par un troisième service Z qui contrôle la visibilité de X sur Y. Par exemple, un vendeur Z peut avoir des restrictions contractuelles à l'égard de son fournisseur de services de transport Y qui travaille pour un autre vendeur X. La surveillance et la notification des progrès [BV05, LAP03] dans un ordre hiérarchique, nécessite souvent la mise en visibilité sur les détails d'exécution des descendants. La résistance aux échecs d'un logiciel commercial est généralement fournie en utilisant l'abstraction transactionnelle [WV01], en particulier l'atomicité, qui garantit qu'un groupe des services s'exécute totalement ou pas du tout. Fournir des garanties transactionnelles exige également

d'avoir la visibilité sur les services, autres que les parents et les enfants. [CD96] propose d'exposer les résultats intermédiaires de sous-transactions aux ancêtres pour obtenir directement une plus grande concurrence. Dans le cas d'un échec, l'atomicité est préservée avec l'aide de la compensation [Bis04], qui a besoin de visibilité sur les détails d'exécution des descendants [Boc04, AH00, VV04].

La principale contribution de ce chapitre est de proposer un modèle conceptuel pour exprimer la visibilité dans les compositions hiérarchiques des services Web. Nous introduisons des notions complémentaires de la Sphère de Visibilité (SoV) et de la Sphère de Noticeabilité (SoN). Pour un service X, SoV de X reflète la visibilité de X sur les autres, tandis que la SoN de X reflète la visibilité des autres sur X. Les SoVs et SoNs de différents services peuvent être tout à fait arbitraires. Nous identifions deux propriétés de visibilité intuitives et duals : cohérence et corrélation, qui rattachent les notions de visibilité et de noticeabilité, et également conduisent à des politiques intéressantes et utiles de visibilité. Aussi, nous étudions certaines variantes de la cohérence et de la corrélation. Pour une composition hiérarchique donnée, nous donnons des algorithmes pour assurer ces propriétés. En général, pour représenter les exigences et restrictions de visibilité d'une hiérarchie ayant n services, nous avons besoin de n^2 graphes (chaque graphe représentant la visibilité d'un service sur un autre). Cela conduit à un problème qui concerne l'espace physique nécessaire pour représenter la visibilité des graphes pour des grandes compositions. Nous montrons comment les propriétés de cohérence et corrélation permettent une représentation compacte de la visibilité d'une hiérarchie qui a n services, avec n graphes, et même avec un seul graphe.

Ce chapitre de la thèse apparaît en partie dans "Debmalya Biswas and Krishnamurthy Vidyasankar. *Modeling Visibility in Hierarchical Systems*. In proceedings of the 25th International Conference on Conceptual Modeling (*ER*), Tucson, Arizona, USA, Lecture Notes in Computer Science vol. 4215 (Springer Verlag), pp. 155-167, 2006". Les variantes des propriétés de visibilité : cohérence et corrélation, et les conditions pour la représentation d'un seul graphe compacte, sont des nouvelles contributions dans ce chapitre.

Partie II

Dans la partie I, nous avons présenté des modèles permettant de représenter les exigences et les restrictions de visibilité. Dans cette partie, nous discutons de la façon de déterminer les exigences de visibilité. Pour être plus précis, étant donné un système, nous étudions la visibilité minimale requise pour maintenir une propriété spécifique. Il s'agit d'un problème bien connu dans le domaine des systèmes à évènements discrets (DES), et quelques-unes des propriétés pour lesquelles les chercheurs ont tenté de déterminer la visibilité requise minimale sont : l'observabilité [LW88], la normalité [KG94], la diagnosticabilité [SSL+95] et la testabilité [BC94, Lin94].

Dans notre cas, le système donné correspond toujours à une composition hiérarchique de services Web. Cependant, pour chaque service composite, nous devons également capturer l'ordre dans lequel il invoque/appelle ses enfants. Cette information est

généralement donnée par le schéma de composition du service (composite), qui spécifie l'ordre des appels, ainsi que toutes les possibilités concernant les choix des enfants à appeler (invoquer). Dans cette partie, nous sommes donc passés à un modèle plus descriptif (en utilisant des machines à états finis), qui représente chaque service composite de la hiérarchie comme un ordre partiel d'actions, chaque action correspondant à l'appel d'un enfant ou d'une activité locale.

Précédemment, nous avions brièvement mentionné que la propriété de non - fonctionnalité des transactions atomiques exige une visibilité sur les descendants. Ici, nous donnons des précisions sur ces exigences et étudions la visibilité minimale requise sur les services d'une composition hiérarchique, de telle sorte que l'atomicité des transactions soit garantie. Les transactions [WV01] sont une abstraction utile permettant de donner aux systemes distribués de la tolérance aux pannes, de la fiabilité et de la robustesse. Une transaction peut être considérée comme un groupe d'actions, encapsulés par l'action « Begin » et par l'action « Commit » ou « Abort », et ayant les propriétés suivante (ACID) :

- Atomicité : Soit toutes les actions sont exécutées, soit aucune. Dans le cas d'une erreur (abort), les effets de toutes les actions de la transaction sont annulés (retour en arrière). - Consistance : Chaque transaction fait passer le système d'un état consistant à un autre état consistant. - Indépendance : Pour améliorer les performances, de nombreuses transactions sont exécutées en même temps. Les transaction sont dites indépendantes si les effets de telles exécutions simultanées sont les mêmes que ceux d'une exécution séquentielles. - Durabilité : Dès qu'un transaction à été exécutée (commit), ses effets sont durables, c'est-à-dire qu'ils ne doivent pas être détruits par des échecs systèmes ou logiciels.

Dans ce travail, nous nous concentrons sur l'aspect atomicité. En cas d'échec, l'atomicité est préservée par la compensation [Bis04, GMS87]. Pour une action A, la compensation consiste à exécuter une autre action, appelée action compensatrice, capable d'annuler les effets de A. Par exemple, « Déposer de l'argent » et « Annuler la réservation d'un billet » sont respectivement les actions compensatrices de « Retirer de l'argent » et de « Réserver un billet ». Dans le même esprit, pour une transaction, la compensation consiste à exécuter les actions compensatrices correspondantes à chacune des actions exécutées dans la transaction, en ordre inverse. De nombreux modèles transactionnels avancés ont également été proposés, comme la « compensation sémantique » [WDSS93] qui ne requière aucune connaissance sur la séquence d'exécutions. Cependant, leurs applications à des systèmes plus autonomes tels que les services Web ont été limités, le mécanismes de compensations d'erreurs du plus utilisé des standards d'implémentations de services Web composites, Bussiness Process Execution Language for Web Services (BPEL) [BPE], se contente « d'exécuter les actions accomplies en ordre inverse ». Par conséquent, pour que la compensation soit possible, nous devons être capable de reconstruire chaque action exécutée, ou l'historique complet de chaque exécution. Pour ce faire, la méthode classique est de maintenir un journal des actions exécutées. Cependant, la journalisation n'est pas seulement couteuse en temps et en espace, mais la totalité du journal peut ne pas être visible. Dans une composition hiérarchique, le journal est distribués parmi les sites fournisseurs des services composants.

De plus, des contraintes de sécurité peuvent empêcher un site fournisseur de montrer des parties des journaux de ces sites, correspondantes à des actions dites confidentielles de ses services. Par ailleurs, l'hétérogénéité peut conduire à ce que les journaux de différents fournisseurs soient maintenus dans des formats différents, rendant certains d'entre eux incompréhensibles. Actuellement, les spécifications des services Web pour garantir les transactions, telles que « WS-Coordination », « WS-AtomicTransaction » et « WS-BussinessActivity » [WST], sont simplement des protocoles d'accords distribués basés sur l'hypothèse que « tout les états de transitions peuvent être enregistrés avec fiabilité » et peuvent être compensés. Notre approche est orientée vers des environnements plus hétérogènes et autonomes, où toutes les actions ne sont pas obligatoirement visibles. Nous souhaitons donc déterminer l'ensemble minimal d'actions de service devant être visible pour garantir la compensabilité. Nous étudions la visibilité minimale requise suivant deux points de vue. Premièrement, nous discutons de la visibilité minimale requise des actions de services d'un point de vue global, lorsqu'un coordinateur global est responsable de la récupération à partir d'échecs de l'ensemble de la composition. Ensuite, nous considérons un cadre de travail plus sécurisé, sur les lignes du Chapitre 3, où il est suffisant pour obtenir la compensabilité, que dans chacune des paires de services parent/enfant dans la hiérarchie, que chacun ait une visibilité sur un petit nombre de propriétés de l'autre.

Malheureusement, déterminer l'ensemble minimal des actions qui doivent être visibles, de sorte que toute exécution d'une composition donnée soit compensable, est un problème NP-complet. Nous montrons également que ce problème reste NP-complet même lorsque le graphe correspondant au schéma de composition des services est borné par des degrés entrants et sortants inférieurs à 3. Nous utilisons une double approche pour surmonter cette NP-dureté : Nous étudions des algorithmes de type Diviser pour Régner afin d'obtenir l'ensemble minimum absolu dans le chapitre 4, et des algorithmes d'approximations pour le problème dans le chapitre 5.

Chapitre 4

Un algorithme de type diviser pour régner fonctionne en découpant de façon récursive un problème donné en deux ou plusieurs sous-problèmes de même nature (ou similaire), jusqu'à ce qu'ils soient suffisamment simples pour être résolus. Les solutions de ces sous-problèmes sont alors combinées pour obtenir la solution du problème original. Dans notre cas, nous disposons déjà d'une décomposition hiérarchique, et nous en utilisons la structure pour obtenir un algorithme de type diviser pour régner. Malheureusement, une simple union des ensembles minimums compensables des différents composants n'est pas un ensemble minimum compensable de la compositions complète. Néanmoins, nous montrons que pour obtenir l'ensemble minimum compensable, il suffit d'exécuter l'algorithme proposé avec des paramètres légèrement différents sur chaque composant. Nous présentons une analyse théorique de la complexité de notre méthode, qui en illustre l'efficacité (jusqu'à deux exponentielles de mieux lorsque nous utilisons la représentation hiérarchique complète, et une exponentielle de mieux lorsque nous utilisons la représentation hiérarchique même si les composants ne sont utilisés qu'une

seule fois, comparé à la hiérarchie mise à plat), qui est aussi vérifiée expérimentalement. Nous expliquons comment travailler avec des compositions hiérarchiques composées à la fois de manière ascendantes et descendantes, et aussi comment acquérir la structure hiérarchique d'un service composite (si elle n'est pas donnée explicitement comme partie de la définition du service). Ce chapitre de la thèse apparait en partie dans "Debmalya Biswas and Blaise Genest. *Minimal Observability for Transactional Hierarchical Services.* In proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (*SEKE*), CA, USA, pp. 531-536, 2008". La discussion concernant la visibilité minimal en l'absence de coordinateur global, ainsi que celle concernant la mise en place de la récupération lorsque que le partage de visibilité se fait uniquement entre pairs de services parent/enfant dans la hiérarchie, sont les principales nouvelles contributions de ce chapitre.

Chapitre 5

Dans ce chapitre, nous étudions des algorithmes approchés pour le problème de la compensabilité minimale. La stratégie de diviser pour régner proposée dans le chapitre précédent n'est pas toujours applicable, les services complexes ne pouvant pas toujours être décomposés de manière hiérarchique. Par ailleurs, lorsque le coût de la journalisation n'est pas prohibitif, ou lorsque la sécurité n'est pas indûment restrictive, il est généralement acceptable (/ préférable) de journaliser un peu plus de transactions, comparé aux coûts de calculs des minimums exacts. Nous proposons dans ce chapitre deux heuristiques, et analysons de façon expérimentale leurs complexité et leurs proximités avec les minimums absolus.

Ce chapitre de la thèse apparait dans " Debmalya Biswas, Thomas Gazagnaire and Blaise Genest. *Small Logs for Transactional Services : Distinction is much more accurate than (Positive) Discrimination.* In proceedings of the 11th IEEE High Assurance Systems Engineering Symposium (*HASE*), Nanjing, China, 2008 (To be published by IEEE CS)".

Introduction

Distributed systems have been evolving over the last 2-3 decades, especially in the last decade, the popularity of the internet has made it central for the development of modern software. Some of the many advantages of developing and deploying distributed systems are better performance, economics, autonomy, etc.

Two paradigms which have been very successful in pushing the cause of distributed systems ahead in recent years are : Web services and Peer to Peer (P2P) systems. Web services provide a uniform way of exposing services offered by an organization, which can then be consumed by clients in an automated fashion. The server hosting the services and the clients may actually be implemented on different software platforms, leading to a high level of *interoperability*. P2P systems, on the other hand, do not have the notion of clients and servers, but only equal "peer" nodes that function simultaneously as both clients and servers to the other nodes in the network. P2P networks allow a peer to connect via largely ad-hoc connections, leading to high *scalability*. Scalability and interoperability are clearly the backbone of any distributed system, and allow the successful development and deployment of large and complex distributed systems involving hundreds of entities.

Once such large distributed systems have been built, then comes the task of running them properly so that they can deliver their promise. While Web services and P2P systems make it convenient for entities to join, share, offer and consume resources in a collaborative fashion, the same convenience also leaves them open to the threat of being exploited by malicious entities, multiple and arbitrary failures, among others. There has been considerable research to provide effective security and failure recovery mechanisms for distributed systems. However, most of the approaches consider either of the two aspects independently. The problem with trying to provide them in conjunction is that their requirements are in some ways contradictory. While security dictates an entity to allow limited accessibility over its resources (preferably only as much as required to satisfy its goals), a failure recovery mechanism would prefer accessibility over the entire execution cycle of the other entities, over their functional specifications, execution states, logs, etc. In this work, we propose the generic conceptual notion of *visibility* to capture such accessibility restrictions and requirements of specific distributed systems, namely hierarchical systems.

Visibility in Hierarchical Systems

Hierarchical systems provide an elegant mechanism to analyze system functionality at different levels of abstraction. They are usually constructed in a top-down or bottom-up fashion level by level. In top-down construction, a complex entity is decomposed into simpler children entities, while in bottom-up construction, simple entities are merged into a more complex parent entity. As a result, by construction, visibility of entities in a hierarchical system is restricted to adjacent (parent-child) levels.

Such restricted visibility is often not sufficient for real-life scenarios. For example, in a supply chain management system, [LTS98] states the visibility requirement across levels as follows : "The information required by downstream entities are mainly material and capacity availability information from their suppliers. The information acquired by an upstream entity is information about customer demand and orders. The depth of information penetration can be specified in various degrees, e.g., isolated, upward one tier, upward two tiers, downward one tier, downward two tiers, and so forth". Non-functional aspects such as transactions, monitoring, user-interaction, etc. also call for entities having visibility over their ancestors, descendents, siblings, and so on.

Allowing arbitrary interaction among the hierarchical entities, without any restrictions, is not an acceptable solution either. In a dynamic and heterogeneous environment, security issues such as trust, privacy, autonomy etc., force an entity to be selective in the interactions it has with others. For example, [BKK05] considers hierarchical modeling of hypertexts in the World Wide Web (WWW). The WWW can be modeled as a large graph with Web pages as nodes and hyperlinks as edges. With respect to a site S, its corresponding graph consists of the pages provided by its departments $d_1, d_2, \cdots, d_n$ and the hyperlinks among them. Given this, the graph can be hierarchically decomposed as follows : (i) By department : The pages (and the corresponding hyperlinks) belonging to a department d are grouped together. (ii) By page structure : Here, we consider a page as an abstraction on its own. This can be useful for an author who wants to change a page while having an overview of its structure. With the above representation, one can imagine the need for specific nodes to be hidden or visible with respect to a given grouping. The need for such an abstraction has been identified in [BKK05] as follows : "A hierarchical graph data model must allow the representation of this kind of information (hierarchy, visibility), and enforce possible constraints that derive from it (for example, forbidden edges to hidden nodes)."

In this thesis, we address two sub-problems of the visibility issue in hierarchical systems. In Part I, we consider the problem of defining a visibility model, given the visibility requirements and restrictions of the different entities in a hierarchy. Part II deals with the orthogonal problem of determining the visibility requirements of the given hierarchy such that a specific property holds. Chapter 1 introduces preliminary definitions and concepts needed for the rest of this thesis.

Part I

The need for a visibility model as well as expectations from it are analogous to that of any abstraction or model of a complex system. The proposed visibility model should be able to capture all visibility characteristics of the underlying system, allowing for a compact representation at the same time. Based on the model, it should also be possible to identify, specify and verify visibility properties and relationships among entities of the hierarchy. Finally, a visibility model should be able to co-exist with other system functional/non-functional aspects, or it should be possible to interpose those aspects on the visibility model.

Given the diversity of prevalent hierarchical systems, it would be impractical to try and design an all-purpose visibility model, which is generic enough to accommodate all the inherent diversity, and not cumbersome at the same time. We study visibility models for P2P Communities and Web services compositions in Chapters 2 and 3.

Chapter 2

P2P Communities is the "collaborative" variant of P2P systems, where communities/groups of peers share some common interests (e.g., Yahoo Groups [YG]) or work together to perform a specific task [Gri, AXM]. We focus on P2P Communities based on common shared interests in this work.

P2P communities are often organized hierarchically corresponding to a hierarchical semantics ontology (Fig. 2). The need for a visibility model arises from security issues like trust, privacy, anonymity, etc. These issues force a peer (or community) to be restrictive in the visibility it allows to others. On the other hand, communities need to grow, and to do this, they need visibility over other communities (and the peers in them). Similarly, a peer would also like to have visibility over peers in communities catering to "related" interests. Thus, visibility in P2P Communities is intermediate between individual visibility over a peer/community and full visibility over all other peers/communities [IMZI04]. And, we need an adequate model to capture and represent this.

The main goals of P2P Communities are of course efficient query and (structural) update mechanisms. In the absence of a central co-ordinator, a query may have to be propagated to multiple peers/communities before a satisfactory answer is found. The other defining characteristic of P2P Communities, and which makes it particularly interesting for us as a candidate system, is its *high dynamicity* : a peer may be added to or deleted from a community, communities may be added or deleted, communities may be merged into bigger ones or split into smaller ones, and sub-communities may become parent-level communities and vice versa. Given this, the proposed visibility model should also facilitate queries and updates over the underlying P2P Communities system in an efficient manner.

Towards this end, we propose a multi-level visibility graph model in Chapter 2 to capture the visibility a peer/community has over the other peers and communities. We give algorithms to perform query evaluation and structural updates in a decentralized

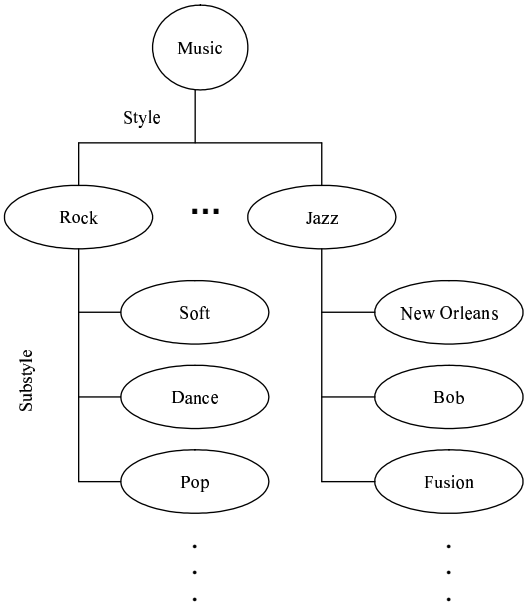

FIGURE 2 – A sample hierarchical ontology.

and scalable fashion. We also show how other non-functional aspects, such as a trust based security framework, can be implemented on top of the visibility model.

This chapter of the thesis appears in "Debmalya Biswas and Krishnamurthy Vidyasankar. *A Highly Flexible Data Structure for Multi-level Visibility of P2P Communities.* In proceedings of the 9th International Conference on Distributed Computing and Networking (*ICDCN*), Kolkata, India, Lecture Notes in Computer Science vol. 4904 (Springer Verlag), pp. 363-374, 2008".

Chapter 3

Web services [ACKM04] have become the de-facto standard for providing services on the Web. The interesting and challenging aspect of Web services is of course its promise of automated composition. Composition refers to the process of combining existing services into a more complex composite service. The composite services may themselves act as components for a higher level composition, leading to a hierarchical composition. A hierarchical composition may also be formed in a top-down fashion when a complex service decides to decompose its functionality into simpler services, which in turn can be decomposed further. We represent a hierarchical composition as a tree, with a pair of parent-child nodes X and Y respectively in the hierarchy, corresponding to the invocation of Y by X.

In comparison to P2P Communities, Web services compositions are more static as they do not change so much. However, their visibility characteristics are also much more complex. For a hierarchical Web services composition, the need for restricting visibility

arises from the fact that the services in the composition may be provided by different organizations, leading to security issues such as trust, privacy, anonymity, etc. At the same time, the main objective of a composition is clearly for the services to work together towards a common goal. To achieve this, a service needs visibility over other services in the composition, over services other than its parent and children, over attributes other than input/output values. Such visibility requirements may be prompted by both functional and non-functional objectives of the hierarchical composition. We outline some real-life scenarios below :

- Due to security reasons, many online shoppers prefer to give their credit card or bank account information directly to the financial institution handling the payment. As such, the financial institution which might have been invoked somewhere down the hierarchy by the seller needs visibility over the payment details of its ancestor (the shopper). Similarly, a shipping company delivering goods directly to the buyer is also a classic example where ancestor visibility might be required.
- Web services compositions usually cater to critical long running business applications which involve monetary transactions. As such, they need to be secure, their progress needs to be monitored, they need to be failure resilient, among others. In addition to a service Y having restrictions with respect to another service X having visibility over it due to security reasons, a third service Z might also have a say in X having visibility over Y. For example, a seller Z might have contractual restrictions with respect to its shipping service provider Y working for another seller X. Monitoring and reporting progress [BV05, LAP03] in a hierarchical setting often requires visibility over the execution details of descendents. Failure resilience for business software is usually provided using the transactional abstraction [WV01], especially atomicity, which guarantees that a group of services either execute completely or none at all. Providing transactional guarantees also requires visibility over services, other than the parent and children. [CD96] proposes exposing intermediate results of subtransactions to ancestors directly to achieve higher concurrency. In the event of a failure, atomicity is preserved with the help of compensation [Bis04], which requires visibility over the execution details of descendents [Boc04, AH00, VV04] (for details, see Section 1.6 and Part II).

The main contribution of this chapter is to propose a conceptual model to express visibility in hierarchical Web services compositions. We introduce the complementary notions of *Sphere of Visibility (SoV)* and *Sphere of Noticeability (SoN)*. For a service X, SoV of X reflects X's visibility over others, while SoN of X reflects the visibility others have over X. The SoVs, and similarly SoNs, of different services may be quite arbitrary. We identify two intuitive and dual visibility properties : *coherence* and *correlation*, that relate the visibility and noticeability notions, and also lead to interesting and useful visibility assignments/policies. We study some variants of coherence and correlation as well. For a given hierarchical composition, we give algorithms to ensure these properties. In general, to represent the visibility requirements and restrictions of a hierarchy having

n services, we need n^2 graphs (one graph each to denote the visibility of a service X over Y). This leads to a problem with respect to the physical space needed to represent the visibility graphs for large compositions. We show how the coherence and correlation properties allow compact visibility representation of the given hierarchy having n services, in n graphs, and even *one* graph.

This chapter of the thesis appears partly in "Debmalya Biswas and Krishnamurthy Vidyasankar. *Modeling Visibility in Hierarchical Systems.* In proceedings of the 25th International Conference on Conceptual Modeling (*ER*), Tucson, Arizona, USA, Lecture Notes in Computer Science vol. 4215 (Springer Verlag), pp. 155-167, 2006". The variants of coherence and correlation visibility properties and conditions for compact single graph representations, are novel contributions of this chapter.

Part II

In Part I, we presented models to represent visibility requirements and restrictions. In this part, we discuss how to determine those visibility requirements. To be more precise, given a system, we study the minimal visibility required for a specific property to hold. This is a well researched problem in the area of Discrete Event Systems (DES), and some of the properties for which researchers have tried to determine the minimal visibility required are : observability [LW88], normality [KG94], diagnosability [SSL+95], testability [BC94, Lin94].

In our case, the given system still corresponds to a hierarchical Web services composition. However, for each composite service, here we also need to capture the order in which it invokes its children. This is usually given by the composition schema of a composite service, which specifies the invocation order as well as any choices which might exist with respect to choosing between a pair of children (to invoke). We thus switch to a more descriptive model in this part (we use Finite State Machines), which represents each composite service in the hierarchy as a partial order of actions, with each action corresponding to the invocation of a child or some local activity.

Earlier, we briefly mentioned that the non-functional property of transactional atomicity requires visibility over descendents. Here, we elaborate on that requirement and study the minimal visibility required over services of a hierarchical composition such that transactional atomicity is guaranteed. Transactions [WV01] are a useful abstraction to provide fault-tolerance, reliability and robustness for distributed systems. A transaction can be considered as a group of actions encapsulated by the actions Begin and Commit/Abort having the following properties (ACID) :

- Atomicity : Either all the actions are executed or none of them are executed. In case of failure (abort), the effects of all actions belonging to the transaction are canceled (rollback).
- Consistency : Each transaction moves the system from one consistent state to another.
- Isolation : To improve performance, often several transactions are executed concurrently. Isolation necessitates that the effects of such concurrent execution are equi-

valent to that of a serial execution.
- Durability : Once a transaction commits, its effects are durable, that is, they should not be destroyed by any system or software failure.

In this work, we focus on the atomicity aspect. In the event of a failure, atomicity is preserved by compensation [Bis04, GMS87]. For an action a, compensation consists of executing another action, referred to as its compensating action, capable of undoing the effects of a. For example, "Deposit Money" and "Cancel Ticket" are the compensating actions of "Withdraw money" and "Book Ticket" respectively. On the same lines, for a transaction, compensation consists of executing the compensating actions, corresponding to each executed action of the transaction, in reverse order. Many advanced transactional models have also been proposed, e.g. "semantic compensation" [WDSS93], which do not require any knowledge of the execution sequence. However, their application to more autonomous systems such as Web services has been limited, where the default compensation mechanism of the widely used industrial standard to implement Web services compositions, Business Process Execution Language for Web Services (BPEL) [BPE], is to "execute the completed actions in reverse order". Thus, for compensation to be feasible, we need to be able to reconstruct each executed action or the complete history of any execution. The usual way of achieving this is to maintain a log of the executed actions. However, logging is not only expensive in terms of time and space, but the whole log may not be visible. For a hierarchical composition, its log is distributed across the provider sites of the invoked component services. Given this, security constraints may prevent a provider from exposing (part of) the log at its site, corresponding to the execution of some say confidential actions of its service. Also, heterogeneity may lead to the logs being maintained in different formats by the different providers, rendering some of them incomprehensible. Current Web services specifications to provide transactional guarantees, such as WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity [WST], are basically distributed agreement protocols which are based on the assumption that "all state transitions can be reliably recorded" and can be compensated. Our approach is targeted towards a more heterogeneous and autonomous environment where all actions of a service may not be visible. *We are thus interested in determining a minimal set of service actions which need to be visible to provide compensability.* We study the minimal required visibility from two perspectives. First, we discuss the minimal visibility required over service actions from a global perspective, by say a global co-ordinator responsible for handling failure recovery of the whole composition. Later, we consider a more secure framework on the lines of Chapter 3, where it is sufficient for compensability, for a pair of parent-child services in the hierarchy to have visibility over some properties of each other.

Unfortunately, the problem of determining the minimal set of actions which need to be visible, such that any execution of the given composition is compensable, is NP-complete. We show that the problem is NP-complete even if the graph corresponding to the composition schema of a service, is bounded by indegree and outdegree less than 3. We take a two pronged approach to overcome the NP-hardness : We study divide and conquer algorithms to get the absolute minimal set in Chapter 4, and approximation

algorithms for the problem in Chapter 5.

Chapter 4

A divide and conquer algorithm works by recursively breaking down the given problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem. Here, we already have a hierarchical decomposition, and use this structure to get a divide and conquer algorithm. Unfortunately, a simple union of the minimal compensable sets of the different components is not a minimal compensable set of the whole composition. Nevertheless, we show that it suffices to run the proposed algorithm with slightly different parameters on each component. We thus obtain a divide and conquer algorithm. We present a theoretical complexity analysis which illustrates the benefit of our method (up to two exponentials better when using the full hierarchical representation and one exponential better by using the hierarchical representation even if components are used only once, compared to flattening the hierarchy), that is also verified experimentally. We explain how to deal with hierarchical compositions composed both in a top-down and bottom-up fashion, and also how to acquire the hierarchical structure of a composite service (if it is not given explicitly as part of the service description).

This chapter of the thesis appears partly in "Debmalya Biswas and Blaise Genest. *Minimal Observability for Transactional Hierarchical Services.* In proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (*SEKE*), CA, USA, pp. 531-536, 2008". The minimal visibility discussion in the absence of a global co-ordinator, on how to perform recovery with only visibility sharing between parent-child pairs of services in the hierarchy, is the main novel contribution of this chapter.

Chapter 5

In this chapter, we study approximation algorithms to solve the minimal compensability problem. The divide and conquer strategy proposed in the previous chapter is not always applicable as all complex services cannot be decomposed hierarchically. Also, in cases where the cost of logging is not prohibitive or security is not unduly restrictive, it is generally acceptable to log a few more transitions as compared to the computationally expensive process of computing the exact minimum. We propose two heuristics in this chapter, and experimentally analyze their complexity and closeness to the absolute minimum.

This chapter of the thesis appears in "Debmalya Biswas, Thomas Gazagnaire and Blaise Genest. *Small Logs for Transactional Services : Distinction is much more accurate than (Positive) Discrimination.* In proceedings of the 11th IEEE High Assurance Systems Engineering Symposium (*HASE*), Nanjing, China, 2008 (To be published by IEEE CS)".

```
<?xml version = "1.0" encoding = "UTF-8"?>
<ATPList date = "18042005">
   <player rank = 1>
      <name>
         <firstname>Roger</firstname> <lastname>Federer</lastname>
      </name>
      <citizenship>Swiss</citizenship>
      <points>475</points>
      <axml:sc mode = "merge" serviceNameSpace = "getGrandSlamsWon"
serviceURL = "..." methodName = "getGrandSlamsWon">
         <axml:params>
            <axml:param name = "name">
            <axml:value>Roger Federer</axml:value>
            <axml:param name = "year">
            <axml:value>$year</axml:value>
         </axml:params>
         <grandslamswon year = "2003">A, W</grandslamswon>
      </axml:sc>
   </player>
...
</ATPList>
```

FIGURE 3 – Sample AXML document with embedded service call "getGrandSlamsWon".

Related Contributions

This section describes some related work, not directly related to visibility in hierarchical systems, but on which I also worked during the course of my PhD. The section can be skipped without any loss of continuity to the remaining part of the thesis.

Active XML Transactions and Security

Active XML (AXML) systems [AXM] provide an elegant way to combine the power of XML, Web services and P2P paradigms by allowing (active) Web service calls to be embedded within XML documents. An AXML system consists of the following main components :

- AXML documents : XML documents with embedded AXML service calls (defined below). For example, the AXML snippet in Fig. 3 is an AXML document with the embedded service call "getGrandSlamsWon".
- AXML services : Web services defined as queries/updates over AXML documents.
- AXML peers : Peers where both the AXML documents and services are hosted. AXML peers also provide a user interface to query/update the stored AXML documents locally.

An embedded service call may need to be *materialized* : 1) in response to a query on the AXML document (the materialization results are required to evaluate the query) or 2) periodically as specified by the "frequency" attribute of the AXML service call tag $< axml : sc >$. The process of materialization is illustrated with the following example : Let the AXML document D corresponding to Fig. 3 be hosted on peer AP_1 and the service $getGrandSlamsWon$ hosted on another peer AP_2. Now, let us assume that the

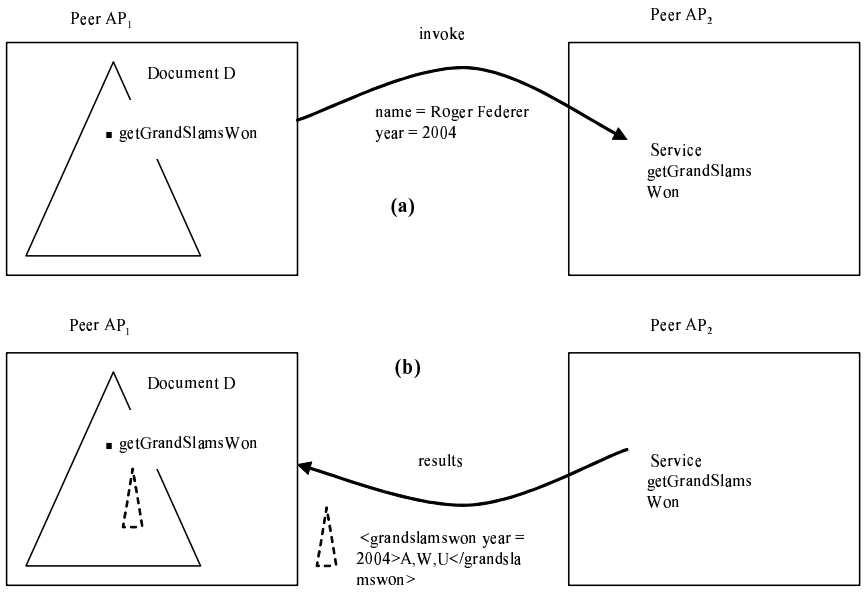

FIGURE 4 – Materialization of the embedded service call *getGrandSlamsWon*.

embedded service call *getGrandSlamsWon* needs to be materialized, then the following sequence of steps takes place :

1. Fig. 4(a) : AP_1 invokes the service *getGrandSlamsWon* of AP_2 with the parameter value children nodes of the service call *getGrandSlamsWon* node of D. A service call's parameters may themselves be defined as service calls, in which case, AP_1 first materializes the parameter service calls and then invokes the service *getGrandSlamsWon*.
2. Fig. 4(b) : On receiving the invocation results (an XML subtree), AP_1 adds the results as a child of the *getGrandSlamsWon* service call node (updating document D). The resulting document D, after a materialization of *getGrandSlamsWon* with parameter value $\$year = 2004$, is shown in Fig. 5. Analogous to parameter inputs, the invocation results also may not be static XML. If the invocation results contain service calls, then AP_1 needs to materialize them first before inserting the results in D.

We studied both transactional and security aspects of AXML systems.

Some interesting and novel issues encountered from a transactional perspective are as follows :

– *Concurrency control :* Researchers have separately considered optimized locking for nested data [JCW02, HH04] and nested transactions [Mos81]. With AXML systems, both the data (XML) and transactional structure are nested. The need

```
<?xml version = "1.0" encoding = "UTF-8"?>
<ATPList date = "18042005">
   <player rank = 1>
      <name>
         <firstname>Roger</firstname> <lastname>Federer</lastname>
      </name>
      <citizenship>Swiss</citizenship>
      <points>475</points>
      <axml:sc mode = "merge" serviceNameSpace = "getGrandSlamsWon"
serviceURL = "..." methodName = "getGrandSlamsWon">
         <axml:params>
            <axml:param name = "name">
            <axml:value>Roger Federer</axml:value>
            <axml:param name = "year">
            <axml:value>$year</axml:value>
         </axml:params>
         <grandslamswon year = "2003">A, W</grandslamswon>
         <grandslamswon year = "2004">A, W,U</grandslamswon>
      </axml:sc>
   </player>
...
</ATPList>
```

FIGURE 5 – Sample AXML document (after an invocation of the embedded service call "getGrandSlamsWon").

for nested transactions arises from the nested invocation of services : a) Local nesting : As a result of the possibility of service call parameters and invocation results themselves containing service calls. b) Distributed nesting : Invocation of a service S_X of peer AP_2, by peer AP_1, may require the peer AP_2 to invoke another service S_Y of peer AP_3, leading to a nested invocation of services across multiple peers. We propose an integrated locking protocol which combines the benefits of both nested data and transactions.

- *Undo operations :* Current industry standards, e.g., BPEL, only allow static definition of the compensating operations, that is, the compensation handlers need to be defined at design time on the lines of exception handlers. However, static compensating operation definition is neither feasible nor sufficient for AXML operations, not even for AXML query operations. Traditionally, query operations do not need to be compensated as they do not modify data. However, AXML query evaluation, due to the possibility of embedded service call materializations, is capable of modifying the AXML document (insertion of the invocation result nodes). To overcome this limitation, we show how the compensating operations (corresponding to AXML operations) can be constructed dynamically at run-time.
- *Undo order :* In general, the compensating operations are executed sequentially in reverse order of their original execution order. We contend that the compensation of operations executed in parallel originally, can also be executed in parallel to optimize performance. However, the presence of nesting leads to additional synchronization issues. We present an algorithm to compute the subtransactions which need to be aborted in the event of a failure, and their optimum compensation order.

- *Peer disconnection detection and recovery :* Peer disconnection is an inherent trait of P2P systems, including AXML systems, which has not been considered in the transaction literature (to the best of our knowledge). Without any knowledge of when a disconnected peer is going to reconnect (if ever), it is very difficult to define a recovery protocol (retry till ?) or even characterize the recovery as "Success/Failure" (the system state on reconnection ?). We outline an innovative solution based on maintaining a list of the active peers, referred to as "chaining", for early detection and recovery from peer disconnection without increasing the communication overhead.
- *Replication :* AXML documents (or fragments of the documents) and services may be replicated on multiple peers [ABC+03]. We consider replication from a recovery perspective, and study the effect of peer disconnection on replication. Given peer disconnection, for both eager [JPPMKA02] and lazy [BKR+99, DS04] replication strategies, we discuss the following : (a) the replication guarantees that can be provided and (b) recovery procedures for peer disconnection and reconnection.

The replication part appears in "Debmalya Biswas. *Active XML Replication and Recovery.* In proceedings of the 2nd International Conference on Complex, Intelligent and Software Intensive Systems (*CISIS*), Barcelona, Spain, IEEE Computer Society Press, pp. 263-269, 2008". The rest of the transactional part first appeared in "Debmalya Biswas and Il-gon Kim. *Atomicity for P2P based XML Repositories.* In proceedings of the 2nd ICDE Workshop on Services Engineering (*SEIW*), Istanbul, Turkey, IEEE Computer Society Press, pp. 369-376, 2007". An extended version of the same has been accepted for publication as a Book Chapter in "Debmalya Biswas and Il-gon Kim. *Active XML Transactions.* In Service and Business Computing Solutions with XML : Applications for Quality Management and Best Processes (To be published by IGI Global), 2008".

From an AXML security perspective, it is necessary to protect : 1) peers from malicious documents and 2) documents from malicious peers. To solve the above security problems, document-level security with embedded service calls as well as XML Encryption and XML Signature have been studied [ABCM04].

Over the last decade, a lot of attention has been paid to the question of developing formal methods for analyzing security protocols. While some methods have been successfully applied to verify security properties of traditional message-based security protocols, they have not yet been applied to analyze security problems specific to AXML document-based systems. For example, AXML documents support query delegation by invoking embedded service calls, which are not considered in Web services message security [WS-]. In addition, it is worth noting that the formal specification and verification issues related with AXML documents include new types of security aspects not considered in traditional message-based protocols. For example, an AXML document is basically an XML document and service calls. As such, it is necessary to develop an abstract model by analyzing XML tagging and embedded service calls. AXML documents invoke embedded security-related service calls in order to obtain a key and generate encrypted or signed documents. This means that an abstract model could be

extracted from two viewpoints : 1) before invoking a security service call and 2) after invocation of a security service call. Besides, it also needs to reflect the fact that there would be more security threats in addition to a traditional one, such as overhearing and modifying transmitted messages. For example, an intruder could embed enormous amount of false data or additional service calls in the returned AXML document to the intended recipient after intercepting the original document.

In this work, we show how existing model checking techniques, generally used for analyzing message-based protocols, can be adapted to verify new vulnerabilities of AXML systems. To do this, we have chosen formal analysis techniques based on Casper (Compiler for the Analysis of Security Protocols) [Low97] and FDR (Failure-Divergence Refinement) [FDR99] because it has already been proven to be very successful in verifying traditional message-based protocols. This work appears in "Il-gon Kim and Debmalya Biswas. *Application of Model Checking to AXML System's Security.* In proceedings of the 3rd BPM Workshop on Web Services and Formal Methods (*WS-FM*), Vienna, Austria, Lecture Notes in Computer Science vol. 4184 (Springer Verlag), pp. 242-256, 2006".

Web services Discovery and Constraints Composition

In this work, we focus on the discovery aspect of Web services composition. We generalize the characteristics of a service which need to be considered for successful execution of the service, as constraints. We present a predicate logic model to specify the corresponding constraints. Further, composite services are also published in a registry and available for discovery (hierarchical composition). The main contribution of this work is to show how the constraints of a composite service can be derived from the constraints of its component services in a consistent manner. This aspect has been mostly overlooked till now, as according to most specifications, the description of a composite service resembles that of a primitive service externally (or at an abstract level). However, determining the description of a complex composite service, by itself, is non-trivial. Given their inherent non-determinism (allowed by the "choice" operators within a composition schema), it is impossible to statically determine the subset of component services which would be invoked at run-time. The above implies the difficulty in selecting the component services, whose constraints should be considered, while defining the constraints of the composite service. Basically, the constraints of a composite service should be consistent with the constraints of its component services. In this work, we take the bottom-up approach and discuss how the constraints of a composite service can be consistently derived from the constraints of its component services. We propose four approaches : optimistic, pessimistic, probabilistic and relative. Finally, we discuss the actual selection (matchmaking) process based on the constraints model. Current matchmaking algorithms focus on "exact" matches (or the most optimum match). They do not consider the scenario where a match does not exist. We try to overcome the above by allowing inconsistencies during the matchmaking process (does not have to be an exact match) up to a "bounded" limit.

This work appears in "Debmalya Biswas. *Web Services Discovery and Constraints*

Composition. In proceedings of the 1st International Conference on Web Reasoning and Rule Systems (*RR*), Innsbruck, Austria, Lecture Notes in Computer Science vol. 4524 (Springer Verlag), pp. 73-87, 2007".

Chapter 1

Preliminaries

In this chapter, we provide the necessary background needed to understand concepts used predominantly in this thesis. First, we introduce graphs which help in graphical representation and understanding of both control flows and structural relationships in systems.

1.1 Graphs

A graph or undirected graph G is an ordered pair $G = (V, E)$ that is subject to the following conditions :

- V is a set, whose elements are called vertices or nodes,
- E is a multiset of unordered pairs of nodes (not necessarily distinct), called edges.

For each edge $e = (u, v)$ the nodes u and v are said to be adjacent to one another. The edge e is also said to be incident to both u and v. For any graph G, the set of nodes in G is denoted $V(G)$, and the set of edges $E(G)$. For example, with reference to the undirected graph G in Fig. 1.1, $V(G) = \{s_1, s_2, s_3\}$ and $E(G) = \{e_1, e_2, e_3\}$.

A directed graph or digraph D is an ordered pair $G = (V, A)$ where

- V is a set, whose elements are called vertices or nodes,
- A is a set of ordered pairs of nodes, called directed edges, or arcs.

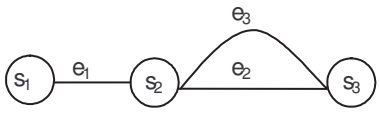

FIGURE 1.1 – An undirected graph G.

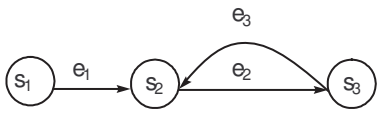

FIGURE 1.2 – A directed graph D.

Here, an arc $e = (x, y)$ is considered to be directed from x to y. For example, Fig. 1.2 shows a directed graph D. We use figures 1.1 and 1.2 to illustrate the the notions in the remaining part of this section.

The degree of a node is the number of edges incident to that node. In directed graphs, indegree is the number of incoming edges at the node and outdegree is the number of outgoing edges from the node. For example, the degree of node s_2 of graph G is 3, while the indegree and outdegree of node s_2 of D are 2 and 1 respectively.

A path in a graph is a sequence of edges $(s_1, s_2)(s_2, s_3) \cdots (s_{n-1}, s_n)$, with s_1 and s_n as the start and end nodes, respectively. A cycle is a path such that the start and end nodes are the same. A directed cycle is a directed version of the above with all the edges being oriented in the same direction. For example, the path e_2e_3 is a directed cycle in D. A directed acyclic graph, also called a DAG, is a directed graph with no directed cycles.

Two nodes u and v of a graph G are called connected if there exists a path from u to v. Otherwise, they are called disconnected. A graph is called connected if every pair of distinct nodes in the graph is connected (directly or indirectly). A subgraph of a graph G is a graph whose node and edge sets are subsets of those of G. In the other direction, a supergraph of a graph G is a graph that contains G as a subgraph. A connected component is a maximal connected subgraph of G. Each node belongs to exactly one connected component, as does each edge. For directed graphs, a directed graph D is called strongly connected if there is a path from each node in the graph to every other node. The strongly connected components of a directed graph are its maximal strongly connected subgraphs. For example, the directed graph $D' = (\{s_2, s_3\}, \{e_2, e_3\})$ is a strongly connected component of D.

We turn now to define a structure which will be useful to express hierarchy.

1.2 Trees

The commonly used graphical form to represent hierarchical structures is a tree. A tree is a connected undirected acyclic graph. A tree is called a rooted tree if one vertex has been designated the root, in which case the edges have a natural orientation, towards or away from the root. The tree-order is the partial ordering on the nodes of a tree with $u \leq v$ if and only if the unique path from the root to v passes through u. A spanning tree of a connected graph G is a maximal set of edges of G that contains no cycle, or a minimal set of edges that connects all nodes.

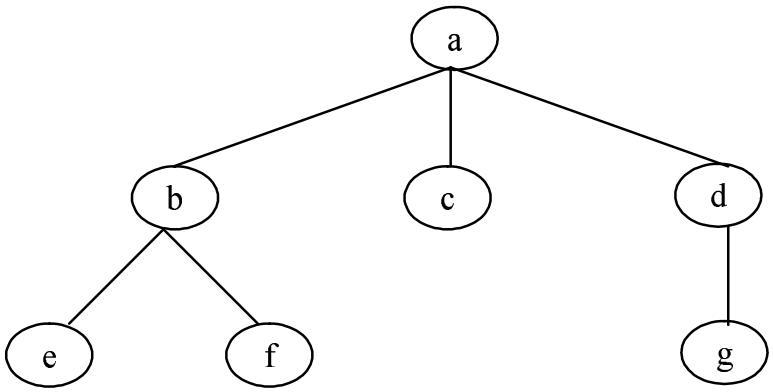

FIGURE 1.3 – A tree (hierarchy) H.

For nodes u and v in a tree (hierarchy) H, $H[u, v]$ denotes the nodes and edges in the (unique) path from u to v. The names of relationships between nodes are modeled after family relations.

- A node u is a "parent" of another node v if u is closer to the root, that is, one step higher in the hierarchy. The definition of a "child" follows analogously.
- Nodes without children are called "end-nodes" or "leaves".
- "Sibling" nodes share the same parent node.
- A node u is an ancestor of another node v if (i) there exists at least one node w in the path from v to u, and (ii) u is the root or we will reach the root on continuing along the path from v via u. Again, the definition of a "descendent" follows analogously.

We use the tree in Fig. 1.3 to illustrate the above relationships.

- a is the parent of b while f is a child of b.
- e, f, c and g are leaves.
- e and f are siblings.
- a is an ancestor of f. On the same lines, g is a descendent of a.

1.3 Graph Traversal

Traversal algorithms allow us to explore all nodes of a graph, determine the distance between a pair of nodes, etc. The two commonly used graph traversal algorithms are : Breadth First Search (BFS) and Depth First Search (DFS). BFS begins at the start node (given as input) and explores its adjacent nodes. Then, for each of those adjacent nodes, it explores their unexplored adjacent nodes, and so on.

BFS(n)

1. Insert n into the queue.
2. Pull a node from the beginning of the queue, mark it, and insert all its unmarked adjacent nodes into (the end of) the queue.
3. If the queue is empty, all nodes have been examined, else repeat Step 2.

Replacing the queue with a stack in the above algorithm gives us DFS. Basically, in DFS, one starts at the specified node, and explores as far as possible along each path before backtracking. A recursive version of the DFS algorithm is given below :

DFS(n)

```
mark n ;
while there is an unmarked adjacent node m of n do
  DFS(m) ;
endwhile
```

For example, BFS and DFS traversal of the tree in Fig. 1.3 with the root node a specified as the start node, would mark the nodes in the following order :

BFS : $abcdefg$
DFS : $abefcdg$

Next, we present Finite State Machines which are a popular formal model to represent flow of control in systems.

1.4 Finite State Machines

We define a Finite State Machine (FSM) as a 4-tuple $M = (Q, s_0, s_f, \mathcal{T})$, where $(Q, \mathcal{T})$ is a graph ($q \in Q$ is called a state and $t \in \mathcal{T}$ a transition) and $s_0 \in Q$ and $s_f \in Q$ are the initial and final states respectively. Note that unlike the usual definition of FSMs, we ignore the use of alphabets to label transitions in the FSM definition as we do not need to read words. We also assume that there are no outgoing transitions from s_f and no incoming transitions to s_0 (We could deal with FSMs without these requirements, but the proofs would be more technical.) The converse does not hold, that is, there can be a non-initial (final) state with no incoming (outgoing) transitions. Our FSMs are thus graphs with a unique input and output point, also known as the source and sink nodes respectively. A sequence of transitions $\rho = \tau_1 \cdots \tau_n \in \mathcal{T}^*$ is a path of M if there exists $q_0, \cdots, q_n \in Q^{n+1}$ with $\tau_i = (q_{i-1}, q_i)$ for all $1 \leq i \leq n$. A path is called initial if furthermore $q_0 = s_0$. We denote by $\mathcal{P}(M)$ the set of initial paths in M. We denote by $|M|$ the size of M, that is, its number of transitions.

In a distributed setting, we may have more than one FSM modeling different systems deployed at different sites. These FSMs can interact with each other to produce

a new distributed FSM. The behavior of such a distributed FSM is defined as a product of its constituent FSMs. Fig. 1.4 illustrates a sample distributed FSM $M \times N$. The distributed FSM can be thought of as a machine that runs its constituent FSMs simultaneously. Interaction or synchronization among the constituent FSMs occurs via shared transitions. With alphabets in the FSM definition, shared transitions are specified as transitions labeled by the same alphabet (e.g., transition a in Fig. 1.4). However, as we do not use alphabets, we assume that any shared transitions for a pair of FSMs M, N are specified explicitly by the set $Sync_{M,N}$. To ease illustration, we still refer to transitions by alphabets, and shared transitions by the same alphabet, in figures and examples.

Definition 1.1 (Distributed FSM) *Given FSMs* $M = (Q_1, s_1, s_2, \mathcal{T}_1)$ *and* $N = (Q_2, s_3, s_4, \mathcal{T}_2)$, *let* $Sync_{M,N} \subseteq \mathcal{T}_1 \times \mathcal{T}_2$ *denote the set of shared transitions. Then,* $\mathcal{T}_1^S = \{t \in \mathcal{T}_1 \setminus \exists t' \in \mathcal{T}_2, (t, t') \in Sync_{M,N}\}$ *and* $\mathcal{T}_2^S = \{t \in \mathcal{T}_2 \setminus \exists t' \in \mathcal{T}_1, (t', t) \in Sync_{M,N}\}$. *Given this, we define the distributed FSM* $M \times N = (Q_1 \times Q_2, (s_1, s_3), (s_2, s_4), \mathcal{T}')$, *where* $((s_1', s_2'), (s_3', s_4')) \in \mathcal{T}'$ *if*

- $(s_1', s_3') \in \mathcal{T}_1 \setminus \mathcal{T}_1^S$ *and* $s_2' = s_4'$, *or*
- $(s_2', s_4') \in \mathcal{T}_2 \setminus \mathcal{T}_2^S$ *and* $s_1' = s_3'$, *or*
- $((s_1', s_3'), (s_2', s_4')) \in Sync_{M,N}$.

An alternate way of extending the functionality of a system M is by allowing some transitions (supertransitions) of M to correspond to FSMs themselves. Of course, an FSM corresponding to a supertransition (of a higher level FSM), may itself have supertransitions, leading to a hierarchical FSM. We consider hierarchical FSMs where two transitions (supertransitions) can be further refined into another FSM.

Definition 1.2 (Hierarchical FSM) *A hierarchical FSM* H *is a finite sequence* $\langle M_i \rangle_{i=1 \cdots n}$, *where* $M^i = (Q^i, s_0^i, s_f^i, \mathcal{T}^i, (\tau_1^i, k_1^i), (\tau_2^i, k_2^i))$ *is defined as follows :*

- $(Q^i, \mathcal{T}^i)$ *is a finite graph,*
- s_0^i *and* s_f^i *are the initial and final states respectively, and*
- $\tau_1^i, \tau_2^i \in \mathcal{T}^i \cup \{\epsilon\}$ *are two supertransitions representing FSMs* $M^{k_1^i}, M^{k_2^i}$ *respectively, with* $k_1^i, k_2^i > i$.

With each hierarchical FSM H, we associate an ordinary FSM $\mathcal{H}$ obtained by taking M^i, and recursively substituting each supertransition τ_1^i by the FSM $M^{k_1^i}$ it represents. Let $\tau_1^i = (s_1, s_2)$ and $M^{k_1^i} = \{Q, s_1', s_2', \mathcal{T}\}$, then on substituting τ_1^i of M^i by $M^{k_1^i}$, we have $M^i = \{Q', s_0', s_f', \mathcal{T}'\}$ where

- $Q' = Q^i \setminus \{s_1, s_2\} \cup Q$
- if $s_0^i = s_1$, then $s_0' = s_1'$, else $s_0' = s_0^i$
- if $s_f^i = s_2$, then $s_f' = s_2'$, else $s_f' = s_f^i$
- $\mathcal{T}' = \mathcal{T}^i \setminus \{(s_1, s_2)\} \cup \mathcal{T} \cup \mathcal{I}$, where $\mathcal{I} = \{(q, s_y') \mid (q, s_y) \in \mathcal{T}^i \wedge y \in \{1, 2\}\} \cup \{(s_y', q) \mid (s_y, q) \in \mathcal{T}^i \wedge y \in \{1, 2\}\}$.

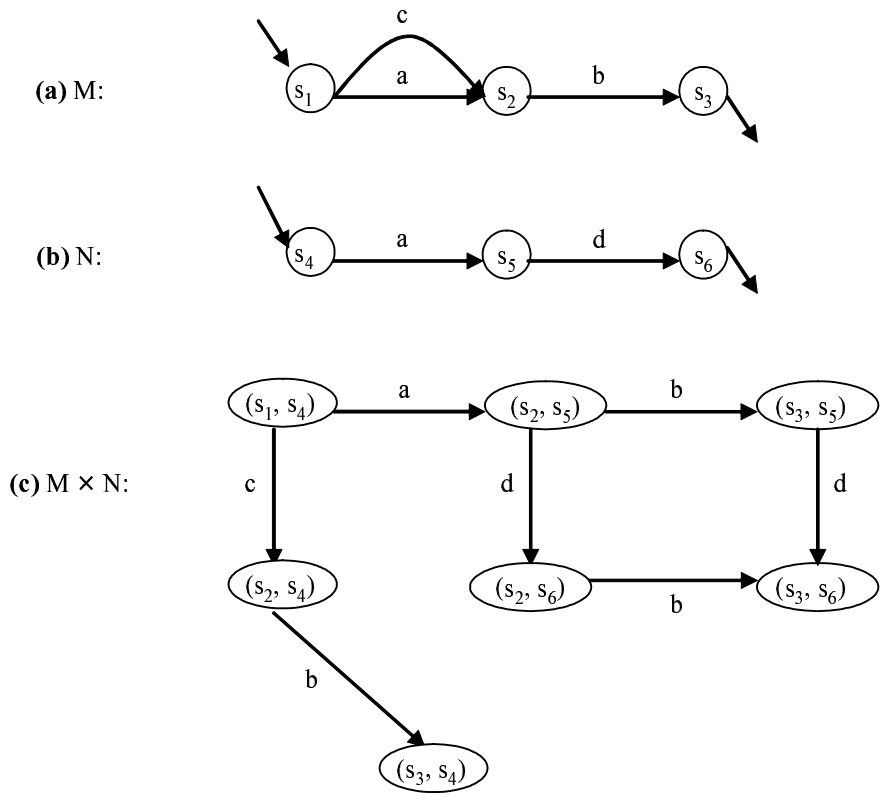

FIGURE 1.4 – (a) FSM M, (b) FSM N and (c) FSM $M \times N$.

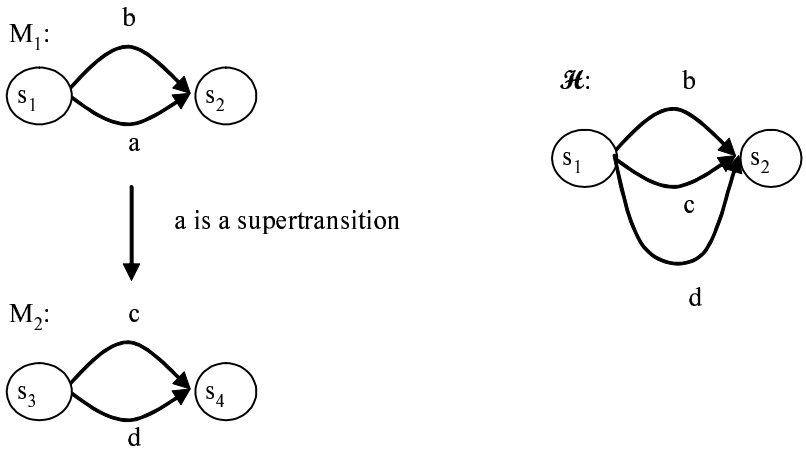

FIGURE 1.5 – A hierarchical FSM H and its corresponding $\mathcal{H}$.

Fig. 1.5 depicts a hierarchical FSM H and its associated $\mathcal{H}$, where the supertransition a of FSM M_1 corresponds to the FSM M_2. Given a hierarchical FSM $\langle H_n \rangle$, $\mathcal{H}_j$ is a component of $\mathcal{H}_i$, if there is a supertransition (t, j) in H_i. We define the size $|H|$ of a hierarchical FSM H as the sum of the number of transitions of its components M^i. Its diameter $||H||$ is the number of transitions of $\mathcal{H}$. Note that the diameter $||H||$ of H can be exponential in the size of H, because components can be reused several times (for instance, a supertransition of H_3 and two supertransitions of H_4 can represent H_{10}, in which case one does not need to redefine H_{10} three times).

1.5 Web Services

Web services [ACKM04] are recognized as the next generation of distributed computing by both academicians and industrial organizations. The World Wide Web Consortium (W3C) defines Web Services as "a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols". Web services, also known in a broader context as Service Oriented Architectures (SOA), are based on the assumption that the functionality provided by an enterprise/provider is exposed as a service. The common Web services usage scenario can be summarized as follows (the steps below correspond to the steps in Fig. 1.6) :

1. Publish : The service provider prepares a Web Service Description Language (WSDL) [WSD] document describing the services it provides. The provider publishes (registers) the WSDL document with an Universal Description, Discovery, and Integration (UDDI) [UDD] registry.

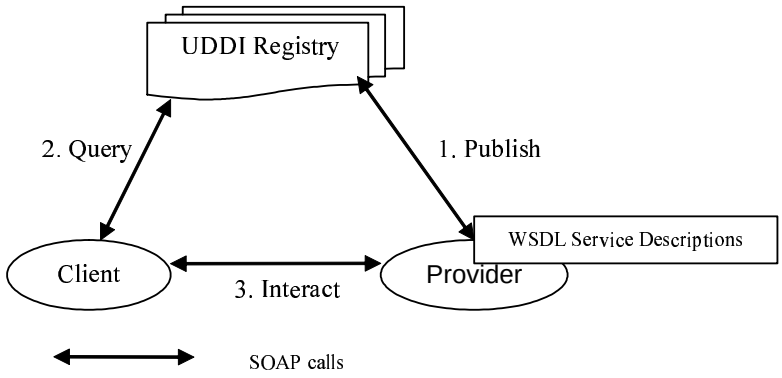

FIGURE 1.6 – Web services usage scenario.

2. Discover : The client (whenever it needs to get some work done) queries the UDDI registry. The registry returns not only descriptive information about the service provider but also information regarding where (endpoint URI) and how (protocol) the service can be invoked.
3. Interact : The client uses the above information to interact with the provider and get the work done.

The above scenario describes how individual services can be published and used. However, the real power of Web services comes from its ability to form new services out of existing ones. This phenomenon is called Web services composition, and consists of assembling autonomous components so as to deliver a new service out of the components' primitive services. Fig. 1.7 shows a typical Web services composition scenario. From the client's perspective, a composite Web service implemented by invoking other primitive Web services is the same as a primitive Web service and can be described, discovered and invoked the same way. Thus, a composite Web service can act as a component Web service for further compositions leading to a hierarchical composition. In this work, we focus on hierarchical compositions of Web services. A hierarchical composition is represented as a rooted tree where nodes denote services and the edges represent the parent-child relationship of a service invoking another service. For example, in the hierarchical composition of Fig. 1.8, the composite Travel and Shipping service invokes the Travel Booking service, which itself is a composite service composed of the Flight Booking and Hotel Booking services.

1.5.1 Web Services Description

To be published, and eventually discovered, it is essential to provide a detailed description of the service's capabilities. As mentioned earlier, the basic description mecha-

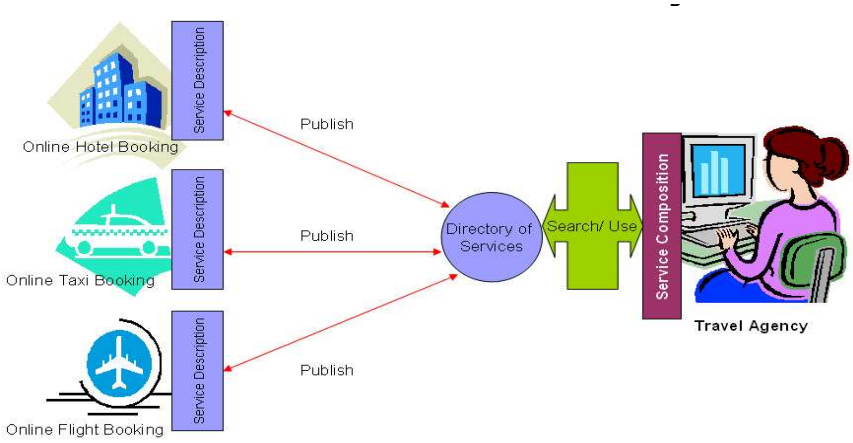

FIGURE 1.7 – Web services composition scenario.

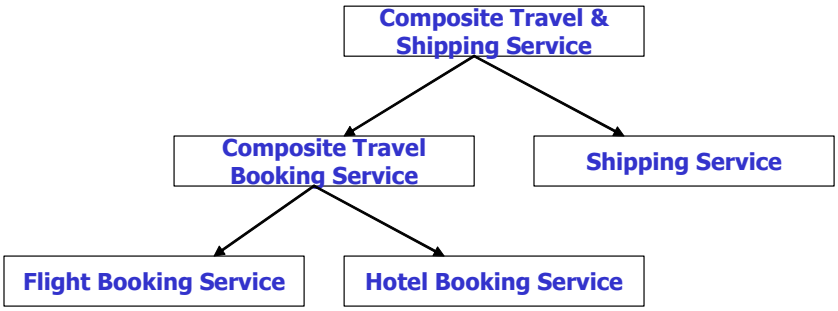

FIGURE 1.8 – Hierarchical Web services composition scenario.

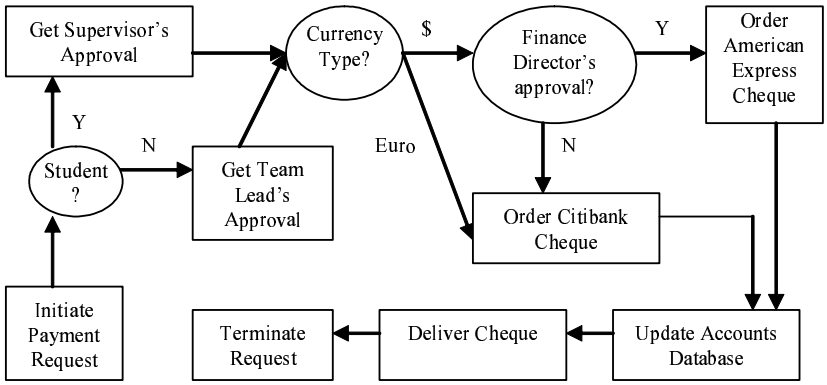

FIGURE 1.9 – Travel Funds service.

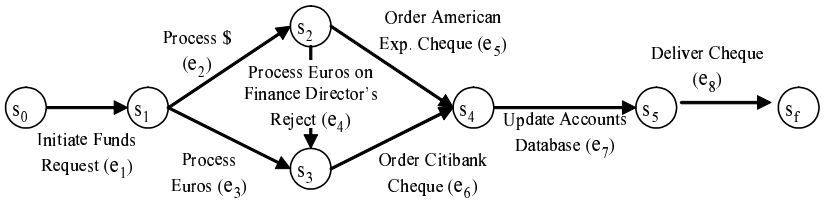

FIGURE 1.10 – FSM representation of the Travel Funds service in Fig. 1.9.

nism is via WSDL, which consists of the input/output signatures and functional/non-functional properties of the service. However, sometimes an abstract WSDL service description is not sufficient, and a more detailed internal representation of the service is needed. Especially for a composite service, we would like to know :

- its component services,
- the state after execution of each component service, and
- if the component services need to be executed in any particular order, e.g., a payment receipt cannot be generated before an order has been received.

Basically, it is essential to consider the behavioral aspects while describing a Web service. Such descriptions are usually provided with the help of a state based formalism. In this work, we give the internal or behavioral description of a Web service as a 4-tuple Finite State machine (FSM) $M = (Q, s_0, s_f, \mathcal{T})$. For example, Fig. 1.9 shows a Travel Funds service, inspired by the workflow in [SO00]. It involves different departments across organizations. It can be modeled as an FSM as shown in Fig. 1.10.

In the practical world, composite services are usually designed and implemented using BPEL [BPE]. In BPEL terminology, the behavioral description of a composite service corresponds to its composition schema. We do not tackle here the modelization of a service as an FSM, which should be handled with care to yield an FSM of reasonable size (see [WFN04]).

We are interested in hierarchical compositions, and hence would like to extend the above internal description to hierarchical Web services compositions as follows. Basically, there are two approaches to hierarchically composing a service :

- Top-down : We say that it is a top-down composition if an existing service would like to add some new functionality or outsource part of its functionality to another service. For example, the Travel Funds service in Fig. 1.9 may decide to extend its "Deliver Cheque" part with a Courier service to process outstation cheques. The extended Travel Funds service is shown in Fig. 1.11. We consider hierarchical services where two transitions (supertransitions) can be further refined into another service, and specify them as hierarchical FSMs (see Definition 1.2). For instance, the hierarchical service in Fig. 1.12 can be described by a hierarchical FSM $\langle M_1, M_2 \rangle$, where M_2 is made of an initial and final state, and two transitions e_8, e_9 from the initial to the final state. The service M_1 is very similar to Fig. 1.10, except that there is a unique transition e_8 between s_5 and s_f instead of two. This is a supertransition (τ_1^1, k_1^1), with $\tau_1^1 = e_8$ and $k_1^1 = 2$, meaning that e_8 represents M_2. Fig. 1.12 actually represents the $\mathcal{H}$ corresponding to the hierarchical Travel Funds service H depicted in Fig. 1.11.
- Bottom-up : In bottom-up approach, given a goal, the client or a provider composes two or more existing services to form a *new* composite service which satisfies the goal. Here, the assumption is that none of the existing services can satisfy the given goal individually, but they can in combination with some others. The newly formed composite service is defined as a product of its components' functionalities (see Definition 1.1). For example, let us consider the FSMs $M = (Q_1, s_1, s_5, \mathcal{T}_1)$ (Fig. 1.13a) and $N = (Q_2, s_6, s_9, \mathcal{T}_2)$ (Fig. 1.13b) representing services which allow searching and listening to songs online [BCLM03]. The services allow different modes of payment and searching for song files by singer/track. The composite service formed by their product $M \times N$ is shown in Fig. 1.13c.

1.6 Transactions

As mentioned earlier, a transaction [PABG87] can be considered as a group of operations encapsulated by the operations Begin and Commit/Abort having A (Atomicity), C (Consistency), I (Isolation), D (Durability) properties. For example, let us consider the classic bank transaction t_b which involves transferring money from an account A to another account B. The transaction consists of two operations : the first operation withdraws money from account A and the second deposits it into account B. Needless to say, any partial execution of the transaction would result in an inconsistent state. The

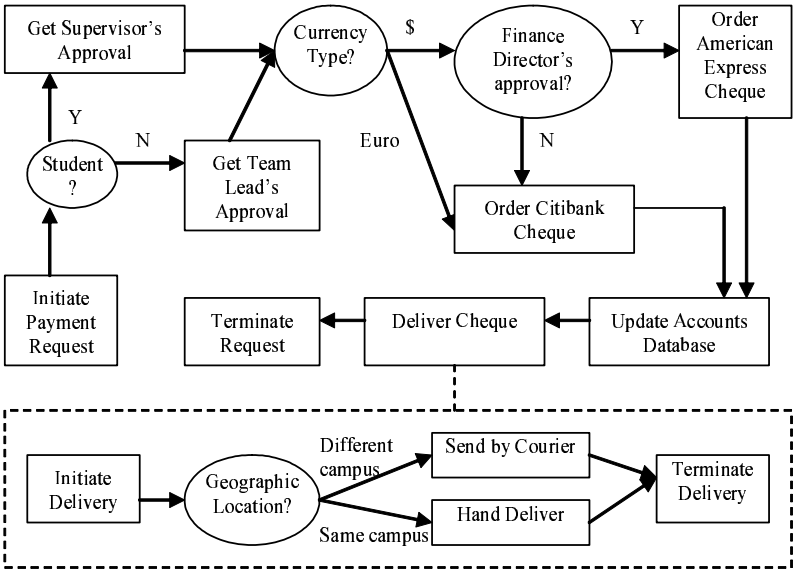

FIGURE 1.11 – Hierarchical Travel Funds service.

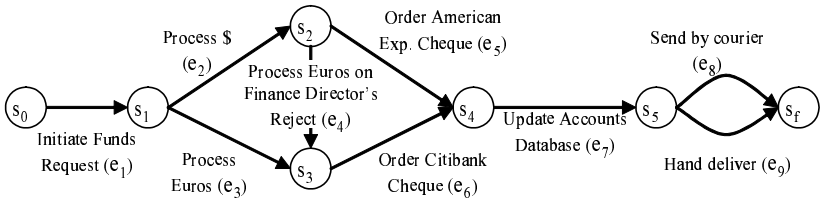

FIGURE 1.12 – Hierarchical FSM representation of the Travel Funds service in Fig. 1.11.

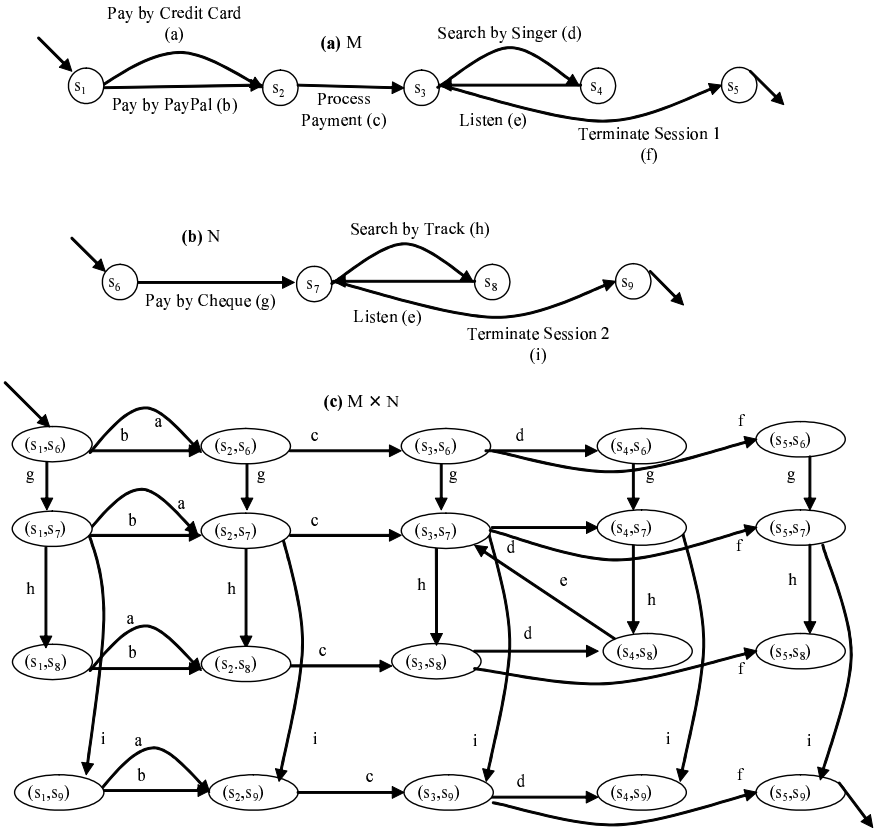

FIGURE 1.13 – Bottom-up composition scenario (product of FSMs).

atomicity property ensures that either both withdraw and deposit operations succeed or both fail. The isolation property ensures that the changes to both accounts A and B are not visible to other transactions until t_b commits. The atomicity and isolation properties together ensure the consistency of the system (accounts A and B).

The software responsible for implementing transactions is often referred to as a Transaction Manager (TM). We can divide the functionality of a TM into the following parts :

- Concurrency Control Manager (CCM) : The CCM ensures the isolation property. Concurrency control mechanisms allow sharing the resources between concurrent transactions in a controlled manner.
- Recovery Manager (RM) : The RM is responsible for providing the atomicity and durability properties in the event of a failure or software/hardware crash.
- Log Manager (LM) : The LM is responsible for writing the execution details to stable storage. The log not only plays an important role in recovery, but is also used to analyze and improve the system performance and efficiency.

While transactions have been very successful in providing fault-tolerance and reliability to stand-alone systems, new challenges arise when we try to apply them in a distributed setting. Distributed transactions consist of operations which are executed at different sites connected by a communication network. Distributed transactions originate at a site (also known as the root site) gradually involving other sites where operations belonging to the transaction are forwarded for execution. The main differences between transaction processing in a centralized and distributed setting are as follows :

1. Decision making : The decision to commit/abort a transaction is not restricted to a single TM. Rather, a collective decision needs to be taken based on the decisions of the TMs of all the involved sites.
2. Multiple points of failure : With centralized systems, the system is either working or not working. However, in a distributed system we can have partial failures in the sense that some of the involved sites fail while others are still working.

As such, we need a protocol which ensures that the same decision (commit/abort) is consistently carried out at all the involved sites irrespective of partial failures. Two phase commit (2PC) protocol (commercially standardized as the XA interface specification) is probably the most widely accepted solution for the above problems. The TM at the home site acts as the coordinator while the TMs at all other involved sites assume the role of participants. As suggested by the name, 2PC protocol consists of two phases. In the first phase, the coordinator TM sends a PREPARE message to all the participant TMs. Each participant TM votes Yes/No depending on whether it wants to commit/abort. If the coordinator TM receives "Yes" from all of its participant TMs, then it begins the second phase of the protocol by sending COMMIT messages to all of them. However,

if it receives "No" from at least one of the participant TMs, then it initiates the second phase by sending ABORT messages to all the participant TMs. Finally, the coordinator TM waits for acknowledgment from the participant TMs to complete the second phase.

While the above protocol works well for tightly coupled distributed applications, its applicability to long running, loosely coupled and cross-organizational applications is limited. To ensure ACID properties (in a centralized scenario), locks need to be held until the transactions commit. With distributed transactions, the locks would have to be held until all the involved (coordinator and participant) sites are ready to commit. The above scenario can be easily extended to a hierarchy of TMs as shown in Fig. 1.14. Analogous to hierarchical Web services compositions, such a hierarchy arises when a site, invoked to process part of the execution, invokes another site to process part of its own execution in a recursive fashion. Given this, all the TMs in the hierarchy are responsible for executing operations associated with the global transaction initiated by the topmost (also known as the root coordinator) TM. All the non-root TMs except the leaves (also known as subordinate coordinators) are responsible for coordinating operations executed by the corresponding subtree of participating TMs. Given such a setting, locks at each site would have to be held until all the TMs in the hierarchy are ready to commit. Obviously, this is not a desirable situation performance wise, especially for long running transactions. An elegant solution to the above limitation is the concept of Nested Transactions [Mos81]. Nested Transactions allow the TMs at the involved sites to release their locks as soon as the transaction completes locally by externalizing intermediate results in a controlled manner. Basically, the global transaction (submitted to the root TM) is divided into a number of subtransactions that may be executed concurrently. While ACID properties are guaranteed for the global transaction, subtransactions are not fully isolated as their results are exposed to their parents. Even the durability of subtransactions is not guaranteed as its effects might need to be canceled (after it has been committed) if its parent aborts.

While Nested Transactions resolve the performance issue to some extent, the effects of a subtransaction are still exposed at its commit and only to its parent, blocking any dependent (sub)transactions at each site. For example, in an online travel agency, a hotel booking would not be reflected in the hotel booking database till the confirmation actually reaches the customer. This delay may have an adverse effect on the transactions which need to be performed following a hotel booking, e.g., "Prepare the Room", "Schedule Staff", etc. In a similar online shopping scenario, the delay might affect the follow-up transaction of an order, that is, the "Restock" transaction which is triggered if the stock falls below a threshold limit. From an implementation perspective, this approach of exposing the effects of a (sub)transaction at its commit is implemented using the "deferred updates" (UD) strategy [WV01]. According to this strategy, each (sub)transaction t has a corresponding private workspace W_t. For each write operation of t, the corresponding data item is copied to W_t and the update applied on the local copy. Upon commit of t, the current values of the data items in W_t are reflected atomically to the actual database. As obvious, abortion is very simple and can be achieved by simply purging W_t.

To overcome the unwarranted delays caused by UD, "updates in place" (UIP) stra-

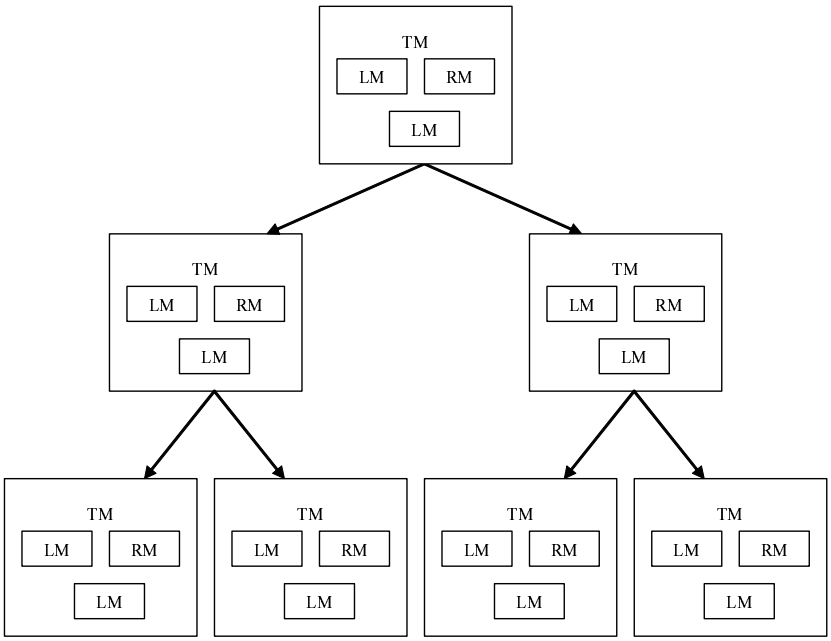

FIGURE 1.14 – Nested transaction processing infrastructure.

tegy [WV01] has been proposed where transactional updates are applied to the database as and when they occur. While UIP clearly improves performance in a failure-free environment, its abortion mechanism is more complicated. If (sub)transaction t is aborted, not only do we have to undo the effects of t but also any (sub)transactions which depend upon t. The latter aspect, of aborting transactions which depend upon an aborting transaction, is known as cascading aborts or the dynamo effect. For example, "Prepare the Room" transaction should also be aborted if the corresponding "Book Hotel" transaction is aborted. Given this, abortion is achieved with the help of a compensating transaction [GMS87, WDSS93] which is responsible for semantically undoing the effects of the original transaction. For example, the compensation of "Withdraw $x from account y" is "Deposit $x into account y", and that of "Book Hotel" is "Cancel Hotel Booking". Here, it helps to recall that compensation is not equivalent to the tradition database "undo". Rather, it is another forward activity (or transaction) which moves the system to an acceptable state (and as such, avoids cascading aborts). For example, a compensating "Deposit $x into account y" does not need to trigger the compensation of any subsequent updates to the accounts table after "Withdraw $x from account y", which would have been required if we had restored (roll-back) the accounts table to its state before "Withdraw $x from account y". On the same lines, "Cancel Hotel Booking" does not need to trigger the compensation of "Prepare the Room" as well based on the semantics that the prepared room can be used for the next booking. Thus, for each

```
<invoke partnerLink="Seller"
  portType="SP:Purchasing"
  operation="Purchase"
  inputVariable="sendPO"
  outputVariable="getResponse">
  <correlations>
    <correlation set="PurchaseOrder" initiate="yes"
      pattern="request" />
  </correlations>
  <compensationHandler>
    <invoke partnerLink="Seller"
      portType="SP:Purchasing"
      operation="CancelPurchase"
      inputVariable="getResponse"
      outputVariable="getConfirmation">
      <correlations>
        <correlation set="PurchaseOrder" pattern="request" />
      </correlations>
    </invoke>
  </compensationHandler>
</invoke>
```

FIGURE 1.15 – BPEL compensation handler usage.

operation, we assume the existence of a compensating operation, capable of semantically undoing its effects. In case of failure, atomicity is guaranteed by executing the compensating (operations) transactions in the reverse order of the original execution sequence of their respective (operations) transactions.

In BPEL, for instance, we can specify a compensating operation for each invoke operation. We can even specify which compensating operation to invoke depending on the type of failure (on the lines of exception/fault handlers). Fig. 1.15 shows an example of a compensating operation specified in BPEL. In this example, the original <invoke> activity makes a purchase, and in case that purchase needs to be compensated, the <compensationHandler> invokes a cancellation operation on the same partner link, using the response to the purchase request as the input. The BPEL specification provides a feature called Long Running Transactions (LRT) which keeps track of the execution order, and then dictates the order in which the compensating operations need to be invoked.

Part I

Modeling Visibility in Hierarchical Systems

Introduction

In his part, we present visibility models for two specific hierarchical systems: (i) P2P Communities (Chapter 2) and (ii) Web services compositions (Chapter 3). The first one is simple, but well suited for highly evolving systems like P2P Communities. We give algorithms to accommodate any structural changes associated with this visibility model in a scalable fashion. The second model is more general, better suited to model more static but complex visibility requirements, like Web services compositions. We study the important properties of visibility, e.g., coherence and correlation. We also analyze the property(ies) needed to get a compact representation of visibility, that is, represent the visibility model which in general takes n^2 graphs for a n node hierarchy, in n graphs, and even one graph.

Chapter 2

Visibility Model for P2P Communities

P2P communities are an abstraction for a group of peers which share some common interests. And, the defining characteristic of P2P Communities is their high dynamicity : a peer may be added to or deleted from a community, communities may be added or deleted, communities may be merged into bigger ones or split into smaller ones, and sub-communities may become parent-level communities and vice versa. The overall objective of this chapter is to give algorithms to perform queries and structural updates in an efficient manner.

Each peer by default has knowledge of (visibility over) the rest of the peers in its own community. For a community to grow, it needs visibility over other communities and their peers. Similarly, a peer would like to have information about communities catering to "related" interests. Current P2P systems either function as independent entities (peers/communities) or assume that each peer is aware of all the other peers and communities ([IMZI04]). Allowing each peer to have full information, about all the other communities and their peers, is not a practical solution. Trust, privacy and anonymity issues in P2P Systems may force a peer (or community) to be restrictive in the visibility it allows to others. Thus, we first need an adequate model to represent visibility in P2P Communities, which should also be adaptive to the inherent structural dynamicity.

Towards this end, we propose a multi-level visibility graph model (Section 2.1) to capture the visibility a peer (or community) has over the other peers and communities. And, show how the visibility model facilitates query evaluation (Section 2.2) and structural updates (Section 2.3) in a decentralized and scalable fashion. Some related works are discussed in Section 2.4.

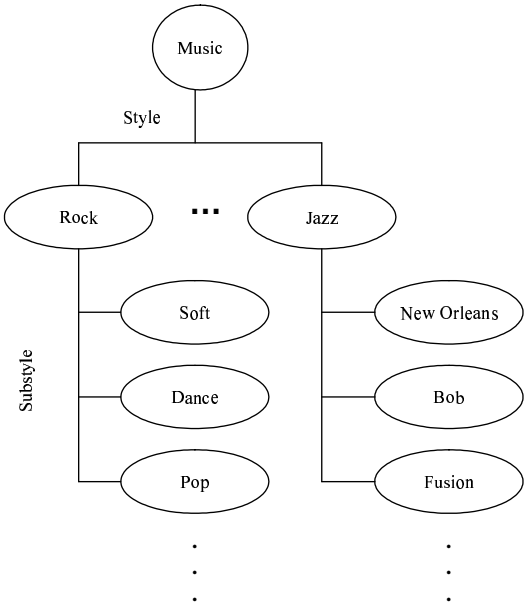

FIGURE 2.1 – A sample hierarchical ontology.

2.1 Visibility Graph

2.1.1 Multi-level Visibility

We present a graph model to represent the visibilities of the peers and communities in a P2P system. Let $\mathcal{P}$ be the set of peers and $\mathcal{C}$ be the set of communities in a P2P system S. Peers are represented by nodes. We use the same notation to refer to the nodes as well as the peers they represent. An edge between peers P_1 and P_2 indicates that P_1 is visible to P_2, and vice versa. The visibility is assumed to be symmetric. Peers are grouped into communities, which may further be grouped into higher level communities, and so on. The communities are usually arranged according to one or more hierarchical semantics ontologies. Fig. 2.1 shows a sample hierarchy of music related interests (as given in [CGM02]). Fig. 2.2 illustrates a hierarchical organization. At the bottom level, there are several communities. We call them level-1 communities. We use single black edges for the intra-community edges. The level-1 communities are grouped into higher level communities, called level-2 communities. The connections at this level are shown by blue dashed edges. These groups of communities belong to an even higher level (level-3) community, connections shown by red dashed-dotted edges. For example, Fig. 2.2 may correspond partly to Fig. 2.1, with level-1 communities of Soft, Dance, Pop, etc., level-2 communities of Rock, Jazz, etc., and level-3 community of Music.

In our model, we represent a parent-level community through certain peers of its children communities called *seers*. For example, the seer S_1 of level-2 community b in Fig. 2.2 maybe P_1. The seers are connected by edges identifying the community. Thus,

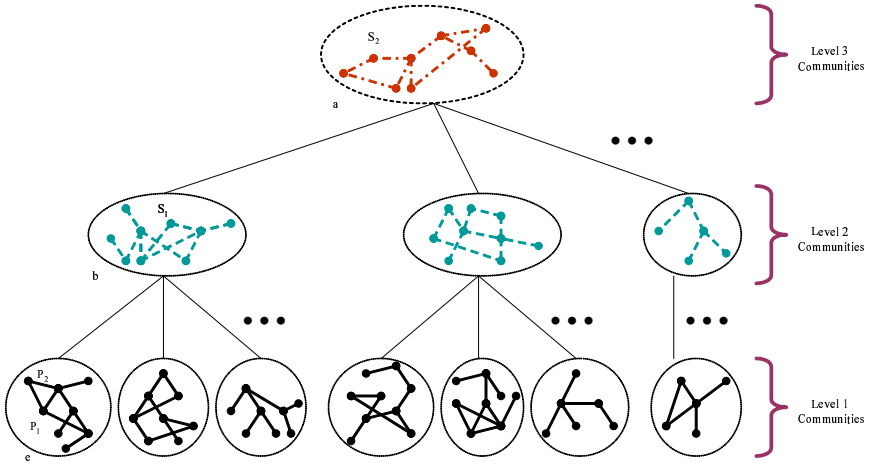

FIGURE 2.2 – Hierarchical visibility.

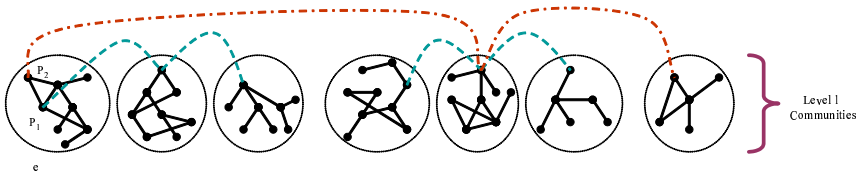

FIGURE 2.3 – Actual connections of the hierarchy in Fig. 2.2.

in Fig. 2.2, the blue dashed edges actually connect seers of level-1 communities, and similarly, the red dashed-dotted edges connect seers of level-2 communities. Thus, the actual connections will be as shown in Fig. 2.3. We note that each node will be incident to single black edges (as long as its community has at least two peers). Then, some nodes may be incident to other (blue dashed, red dashed-dotted, etc.) edges too. In Fig. 2.3, there are nodes incident to (i) single black and blue dashed edges, (ii) single black, blue dashed, and red dashed-dotted edges, and also (iii) single black and red dashed-dotted edges. That is, a peer may be a seer of a higher level (e.g., level-3) community, and not of a lower level (e.g., level-2) community. For example, in Fig. 2.3, P_2 is a seer of the level-3 community a, but not that of the level-2 community b (P_1 is the seer for b). Thus, the edges of the visibility graph are basically of different level communities in the hierarchy.

2.1.2 Model

Let $\mathcal{H}$ be a set of hierarchies that may exist in a P2P system S. We confine our initial discussion and the graph model to one hierarchy $H \in \mathcal{H}$. Fig. 2.4 shows a possible

community wise representation (intra community connections are not shown for clarity) of the hierarchy used in our previous example. Each node in the figure represents a community at some level of the hierarchy. We assume that each peer forms its own *unit community*. We call this level-0 community. Thus, Fig. 2.4 should be expanded to Fig. 2.5 to show the complete community structure in the hierarchy. We use Fig. 2.5 to illustrate some notations and concepts. The nodes, and the corresponding communities, are named locally (within that level) and globally with the sequence of local labels of nodes in the path from the root to that node. For example, e is the local name and $a/b/e$ is the global name of a node in the figure. We use the global name to indicate the corresponding path also.

We note that only the leaf nodes in H (Fig. 2.5) correspond to peers in S. They have labels as in the figure. (Labels of only some nodes are shown for easy readability.) All non-leaf nodes are virtual. Essentially, H describes all (lower and higher level) communities in S. Each peer belongs to several, hierarchically related, communities. To be precise, a peer γ with label $\alpha/\beta/\gamma$ is a member of all the communities in the path $\alpha/\beta/\gamma$, in this case, communities α and β.

For each node (with local or global name) α in the hierarchy, we define two communities. The first one is the *α-full-community*, denoted with α in square brackets as $[\alpha]$-community. This consists of all the peers in the subtree rooted at α. For example, $[b]$-community membership is $\{P_1, P_2, P_3, P_4, P_5\}$. The second is *$\alpha$-seer-community*, denoted with α in parentheses as (α)-community. For non-leaf nodes α, this will contain one or more peers from *each* of its children communities, these peers will be *seers* of the respective children communities *for* α. For example, (b)-community membership could be $\{P_2, P_3, P_5\}$, where P_2, P_3 and P_5 are the seers of the children communities e, f and g for b respectively. For leaf nodes α, the local name will be that of the respective peer, say P_i, and both the full and the seer communities will consist of just that peer. Now, a peer can be a seer for several ancestral communities. For a community C, we define a *C-graph* as the graph with node set consisting of members of C and edges, called C-edges, depicting the visibility among the members. When C is a seer community, we refer to the graph as *C-seer-graph* also. In our model, we require that each seer graph is connected.

Definition 2.1 (Global Visibility Graph) For a P2P system S, with peers $\mathcal{P}$, hierarchy H, and related set of (full) communities $\mathcal{C}$, a *global visibility graph* is the union of C-seer-graphs of all communities C in $\mathcal{C}$ such that each seer graph is connected.

Example : A visibility graph for the P2P system in Fig. 2.5 is shown in Fig. 2.6. Here :

- $\mathcal{P}$ is $\{P_1, P_2, ..., P_{11}\}$.
- $\mathcal{C}$ is $\{[a], [b], ..., [k], [P_1], [P_2], ..., [P_{11}]\}$.
- $[P_2]$ is the unit community containing peer P_2, and the corresponding seer community (P_2) also contains P_2.

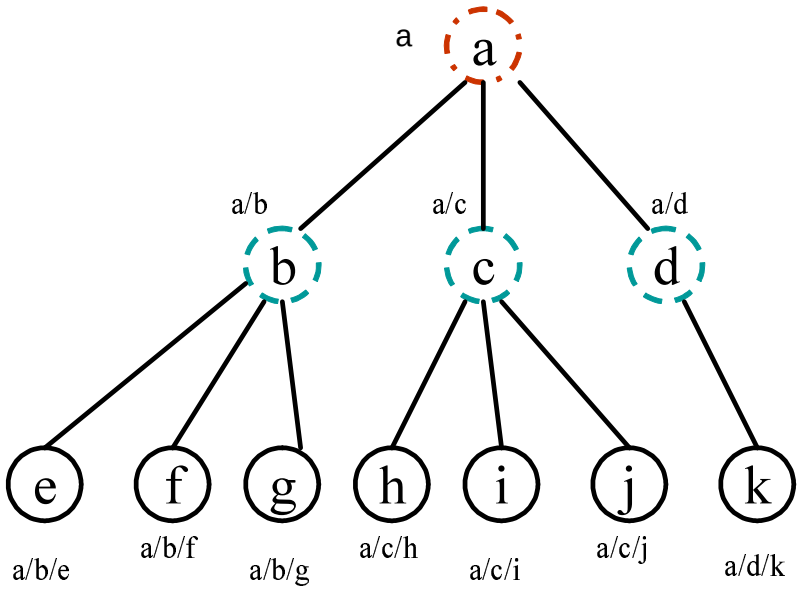

FIGURE 2.4 – Hierarchy of communities.

- Both $[e]$- and (e)-communities have peers P_1 and P_2. Similarly, for each of $\{f, g, ..., k\}$, their full and seer communities have all their children shown in the figure.
- Each of $\{[a]$-,$[b]$-,$[c]$-,$[d]$-$\}$ communities have all the peers in the leaf level of the corresponding sub-trees in the hierarchy. For example, $[b]$-community membership is $\{P_1, ..., P_5\}$.
- The seer community membership is as follows : (b)-community has $\{P_1, P_3, P_5\}$, (c)-community has $\{P_6, P_7, P_8\}$, (d)-community has $\{P_{11}\}$, and (a)-community has $\{P_3, P_9, P_{11}\}$.

In this example, the connected graph of each seer community is a tree, in fact, a path. Note that P_8 is a seer for (c)-community but P_9 is the seer for (a)-community. That is, different peers of a sub-tree may be seers for different ancestral communities.

We point out that the visibility graph does not show the member peers of higher level full communities explicitly. However, the property that for every full community at every level its corresponding seer community must have at least one peer from each of its children communities, guarantees that all the members of that community can be accessed if needed.

2.2 Searching the Visibility Graph

Any search involves searching one or more communities. Search within a community typically involves flooding, that is, searching each peer in the community. Common

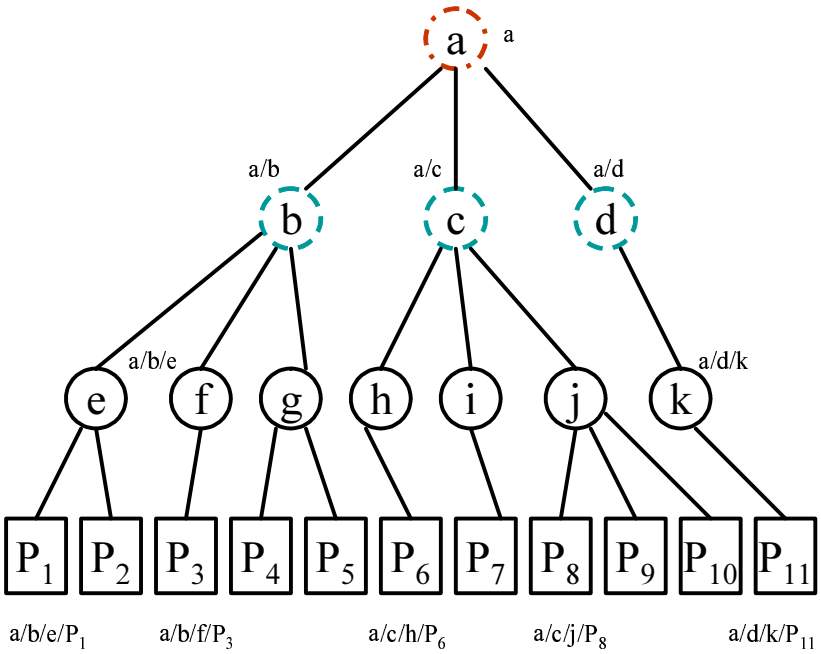

FIGURE 2.5 – Complete hierarchy with peers.

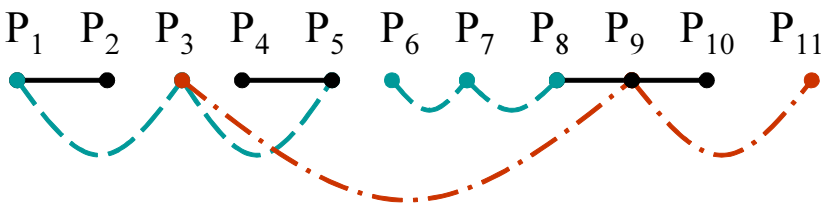

FIGURE 2.6 – Complete visibility graph of Fig. 2.5.

search methods, when the peers within a community are arbitrarily connected, are BFS and DFS, with or without using a spanning tree. A search may be initiated from any node. It may also be initiated, in parallel, from several nodes. In each case, the search results may be forwarded to one or more initiating nodes, to some other nodes, or even to all the nodes. In this work, we will neither be concerned with the actual search method nor with any specific way of forwarding the results. We denote searching a community C as C-search.

We explain a general search procedure with the graph in Fig. 2.6. Any query is initiated at a peer. Suppose a query q is initiated at P_2. Then, first, the (P_2)-search and, if necessary, an (e)-search will be performed. If q cannot be evaluated within (e)-community, then P_1 (which is the seer of $[e]$-community for (b)-community, will initiate a (b)-search. This will involve the graph consisting of P_1, P_3 and P_5. Then, P_5 may suggest that a (g)-search is appropriate, involving peers P_4 and P_5. On the other hand, P_3 may suggest that (the higher level) (a)-search, involving P_3, P_9 and P_{11} may be appropriate. Taking up one or more suggestions and continuing the search can be done either in a centralized or distributed fashion.

Thus, in comparison to traditional query resolution via flooding, our method provides the following benefits : Privacy and security constraints are maintained as queries are only forwarded to visible (trusted) peers. On the other hand, each peer may be a member of different seer communities at different levels. And, based on the information it has on each of these communities, it may be able to direct the search to any of these communities. For example, in the above search process, P_3 will be involved in the searches of (P_3)-, (f)-, (b)- and (a)-communities. Suppose (P_3)-search is the first one. The next search could be (a)-search, then (f)-search, etc. This allows the searches to be mixed rather than being strictly top-down or bottom-up. In (P_3)-search, its role is an individual peer ; in (a)-search, its role is a seer for $[b]$-community, and in that capacity it is supposed to utilize some summary information of the $[b]$-community ; and so on.

2.2.1 A Specific Example

We consider simple hierarchical path queries having the following syntax :

$I :=$ An area of interest in the underlying (hierarchical) ontology O.
$Path := \epsilon | I/Path$
$Query := (n)Path | (Query \wedge Query) | (Query \vee Query)$
where n refers to the *number of nodes* to retrieve satisfying the $Path$ criterion. $I/(sub\text{-})I$ refers to a pair of parent-child interests in O. A sample query is given below :

Query q. $(1)music/classic \wedge ((1)music/jazz/fusion \vee (2)music/rock/soft)$.

Query q can be interpreted as follows : Find a peer interested in $music/classic$ and a peer interested in $music/jazz/fusion$, or a peer interested in $music/classic$ and two peers interested in $music/rock/soft$.

Given this, a search to evaluate query q over the global visibility graph (Fig. 2.6) would be as follows. We assume that q was submitted at peer P_3, and that a, b, c and d correspond to the *music*, *music/rock*, *music/jazz* and *music/classic* communities, respectively.

1. P_3 initiates an (a)-search, that is, evaluates q within the (a)-community.
2. If unsuccessful, P_3 chooses nodes in the (a)-community (peers P_9, P_{11}, and peer P_3 itself) for propagating (the entire query or parts of) q for further evaluation. Peer P_3 makes this decision based on the "similarity" between the path criteria in q and the global names of the nodes in the (a)-community. Here, P_3 would choose itself for the (sub)query $(2)music/rock/soft$, while P_9 would be chosen for $(1)music/jazz/fusion$ and P_{11} for $(1)music/classic$.
3. The nodes P_3, P_9 and P_{11} initiate a (b)-search, (j)-search and (d)-search, respectively. In the course of (j)-search, P_9 might find a need for, and initiate, a (c)-search.
4. This continues until the appropriate peers are found, or all the peers in H have been explored (in which case, there may not exist peers satisfying the query).

2.3 Visibility Structure Changes

As mentioned earlier, P2P systems are highly dynamic in nature, with peers being added to or deleted from communities, communities being added to or deleted from higher level communities, merging and splitting of the communities, etc. All these operations change the visibility graph. Our model facilitates these changes easily. We outline the general procedures in the following, with examples from the hierarchy of Fig. 2.5, and Fig. 2.6.

(a) *Adding a peer P to a community C.* Add the node P and a C-edge between P and some node already in C. Note that the addition of this single edge is sufficient to keep the new C-graph connected. Of course, additional edges can be added too. We restrict our description to the minimum requirements.

(b) *Deleting a peer P from a community C.* This may require several operations :

1. P is deleted from the C-graph.
2. If this disconnects the C-graph, then additional C-edges, between other peers in C, are added to keep the C-graph connected.
3. Suppose P is a seer of a sub-community C' of C for C. If P is the only such seer, then some other peer P' of C'-community should be designated as a seer for C, and added to C.

(c) *Deleting a peer P from the P2P system S.* Then, P must also be deleted from every community it is part of (as a seer).

(d) *Adding a community C to a higher level community C'.* Designate a peer P in C as a seer, and add a C'-edge between P and an already existing peer in C'.

(e) *Deleting a community C from a higher level community C'.* If P is the seer of C in C', then P has to be deleted from C', and the procedure is the same as the case (b) above. For example, let us consider deleting the community j in Fig. 2.5, from c. Recall that the (c)-community membership is $\{P_6, P_7, P_8\}$. This can be done by

1. Deleting the c-edge between P_7 and P_8.
2. Designating P_7 as the seer of c for a, and deleting P_8 and adding P_7 in the (a)-graph.

Note that there is no need to alter the internal structure of j.

(f) *Merging two same level communities.* We consider the following two scenarios with reference to Fig. 2.5.

- Merging communities with the same parent, h and i. The c-edge between P_6 and P_7, the respective seers, is changed to h-edge, assuming that the merged community will be called h. Also, P_7 may be designated as the seer of h for c. Then, P_6 is deleted from the (c)-graph. Again, in general, some c-edges may have to be added to keep it connected.
- Merging communities with different parents, g and h. A simple strategy is :

    1. Delete g from b, and delete h from c, performing all the additional operations required as in the above cases.
    2. Merge g and h, to form a new h, by adding h-edge between P_5 and P_6, the respective seers.
    3. Add h to b or c, as desired.

(g) *Splitting a community to two communities under the same parent community.* Consider splitting community c into c_1 consisting of h and j, and c_2 consisting of i alone. One way of doing this is by

1. Deleting i from c.
2. Renaming c as c_1, and i as c_2.
3. Adding c_2 to a.

Note that deletion of i from c requires adding a c-edge, in fact a c_1-edge, between P_6 and P_8.

Some of the other structural changes are :

- merging two different level communities,
- elevating a sub-community to a parent-level community,
- making a community a sub-community, etc.

These, and other similar structural operations involving combinations of those considered above, can be executed. All these involve essentially some generic operations like adding a node to a graph, deleting a node from the visibility graph but keeping the graph connected after the deletion, etc. Depending on how the graph (C-graph for community C) is maintained (as a tree, arbitrary connected graph, complete graph, etc.), these operations can be implemented efficiently. The key property of our model is that a node may be incident to C-edges of several communities C, and so the node may have to be deleted from some C-graphs and not from some others.

2.4 Discussion and Related Works

As we know, P2P systems are very highly decentralized. Peers are far apart, they are highly autonomous, they may join the system and drop out in an ad hoc manner, their storage and processing capacities may be varied and limited, their availability and accessibility may be different at different times, communication between them may not be reliable, and so on. Therefore, implementation of the algorithms for both searching the peers for query evaluation and for modifying the structure of the visibility graph, outlined in the previous two sections, requires special attention. For instance, in any distributed implementation, the algorithms will be executed only lazily, that is, asynchronously. The algorithms can be fine tuned for "safe" execution. For example, when a community is deleted from another, and added to a different community, the addition part can be done first, and then the deletion. This may result, in the worst case, in a concurrent execution exploring more peers than necessary, but without loss of visibility.

We note that peers can execute their part of the algorithms autonomously. For example (as discussed in Section 2.2), a peer which is a seer for several (higher level) communities could employ its own heuristics to decide on which order to explore the various communities. The selection strategy could be different for a different peer even for the same set of choices.

The core element of the implementation is the manipulation of C-graph for each community C in the hierarchy. Depending on the type of connectivity maintained, the amount of work in the manipulation will be different. Searching could be more systematic and efficient with tree structure. On the other hand, updates on the hierarchical structure may be implemented more efficiently with graphs which are not trees.

With our model, several C-graphs have to be manipulated. However, each can be manipulated independent of the others. So, the complexity of the implementation will

not increase very much with an increase in the number of communities. With proper extensions to the underlying hierarchy, an increase in the number of peers in the P2P system can be accommodated by increasing the number of communities without substantially increasing the size of the corresponding seer-communities. Thus, our data structure and the algorithms are highly *scalable*.

At an abstract level, multi-level visibility seems close to the notion of database views [vie]. For a set of relational tables in a database, a view allows us to summarize their data based on some characteristics. Here, the peers can be considered as data. The information on the various peers in a community can be summarized and kept in the seers of that community. However, since a peer can be a seer of a higher level ancestral community without being a seer of a lower level one, hierarchical relationships of the communities cannot be represented directly. Having said this, there is also sufficient overlap between the two concepts, for a lot of the existing work for database/views to be used here, especially with respect to query optimization and rewriting. Also, we have not considered *concurrent* query and update of the visibility graph which is very relevant, especially, for collaborative P2P systems. Database solutions [AAC+99] appear very attractive for the above as well.

The notion of P2P communities was introduced in [KRD02]. They also represent visibility using intra-community and inter-community edges. They use only one type of inter-community edges. Our formalism uses different edges for (seer communities at) different levels of the hierarchy.

Further works [IMZI04], [AR03] and [KRD03] have extended the P2P community notion as follows : [IMZI04] and [MNBE08] present a Trust Management and Role Based Access Control scheme for P2P communities, respectively. [AR03] discusses efficient discovery for P2P communities based on the "type" of communities, e.g., co-operative, goal oriented, ad hoc communities, etc. [KRD03] presents a gossip based discovery mechanism for P2P communities. We consider visibility graphs as an abstraction to capture the visibility aspect of P2P communities, and other middleware aspects, e.g., security, discovery, monitoring, etc. can build on this abstraction. In the sequel, we briefly show how visibility graphs facilitate P2P security with the help of a Trust Management scheme [IMZI04]. In this scheme, each peer maintains the trust rating of all the other members in its own group (group-mates), based on its own dealings with them. For inter-community accesses, "when a member p requests to acquire resources from a member q of another community, it sends a request to q. q checks with the group-mates of p if p is trustable and what kind of access privileges it has. q then accepts or rejects the transaction with p". The above can be modeled using visibility graphs as follows : The underlying assumption, that the seer graph of each community is connected, allows a peer to monitor the activities of its group-mates (that is, to maintain their trust ratings). Inter-community access between peers $P_1 \in C_1$ and $P_2 \in C_2$ may only occur via an access path from a C_1-seer to a C_2-seer. Further, recall that visibility is symmetric. Thus, given such an inter-community access, P_2 can backtrack along the access path, and retrieve the trust rating of P_1 from the C_1-seer.

Chapter 3

Visibility Model for Hierarchical Web Services Compositions

In the previous chapter, we explored a visibility model which accommodates highly evolving systems. Here, we consider a more static environment, basically hierarchical Web services compositions, which also requires a more complex visibility assignment. We emphasize on properties which lead to interesting practical visibility policies and a compact representation.

For two services X and Y in a hierarchical Web services composition H, whether X can see Y, that is, whether X has visibility over Y, depends on the following :

- X wishes to see Y : X may be interested in Y due to functional or non-functional requirements.
- Y does not have any objection to X seeing it : Security, privacy, confidentiality, etc. issues play an important role in determining the visibility allowed by a service.
- Remaining nodes in H do not have any objections to X seeing Y : Contractual agreements between Y and another service Z may have a bearing on X seeing Y.

In the remaining part, we develop a generic conceptual model to express and restrict visibility. We study two complementary notions : *Sphere of Visibility* (Section 3.2) of a node X that includes all the nodes in the hierarchy that X can see, and *Sphere of Noticeability* (Section 3.3) of X that includes all the nodes that can see X. We identify two visibility properties : *coherence* (Section 3.2.1) and *correlation* (Section 3.2.2). We give algorithms to ensure these properties for a given hierarchical composition (Section 3.4.1). We present conditions under which the visibility of a hierarchy can be represented compactly as a single graph (Section 3.4.2). Finally, some comparisons with related works are discussed in Section 3.5.

3.1 An Informal Introduction to Sphere of Visibility (SoV)

The Sphere of Visibility (SoV) of a node X (SoV_X) consists of services visible to X in the hierarchy. The visibility is with respect to some attribute A such as provider details

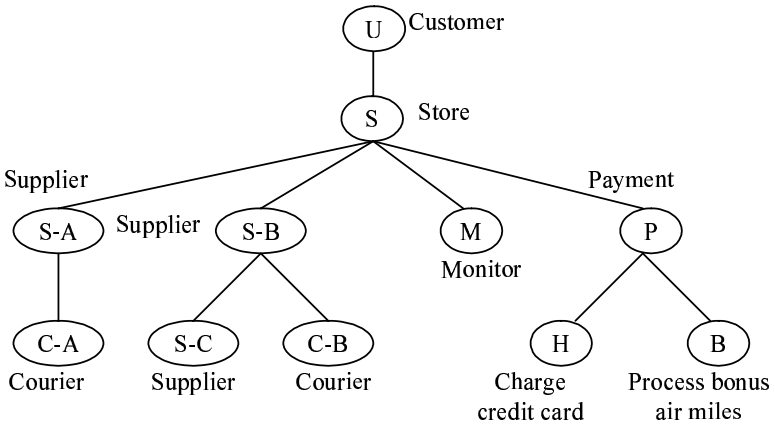

FIGURE 3.1 – A hierarchical composition graph H corresponding to an e-shopping scenario.

(URI, physical address), service details (inputs, outputs, pre-conditions and effects) and execution details (execution state, history). Roughly, provider details are required to invoke a service, service and execution details are required for non-functional aspects such as recovery, monitoring, auditing, and others. We assume that the visibilities of a service with respect to different attributes are independent, that is, a service X might have visibility over Y's provider details only, service details only, execution details only or any combination of the above.

For example, let us consider an e-shopping scenario (Fig. 3.1). A customer U orders a few goods from a store S. S splits the order into two parts and sends them to suppliers $S-A$ and $S-B$. Supplier $S-B$ uses supplier $S-C$ to fulfill part of the order. $S-A$ and $S-B$ use courier companies $C-A$ and $C-B$ respectively to ship the goods to the customer. The store uses a financial service P for processing payment for the goods. This involves charging a credit card by the credit card company H and awarding bonus air miles by another service B. The store also uses a monitor/auditor M to keep track of the service execution.

Taking the attribute A as service details, the SoVs of some services in the hierarchy of Fig. 3.1 may be as follows, shown in Fig. 3.2. In the illustration of visibility of X over Y, X is represented in double ovals, Y is represented in thick oval, and the other nodes, if any, are represented in thin ovals.

- The store S has visibility over its parent and all its children. It does not have visibility over the next level descendents (Fig. 3.2(a)).
- The bonus air miles processing unit B's visibility is limited to the credit card company H and the customer U (Fig. 3.2(b)). It is only concerned with the customer's credit card number and the purchase amount *without any need to know the context*, namely the goods purchased and the store. We call the visibilities of B over H and U as *weak visibilities*, whereas the visibilities of S over U, $S-A$,

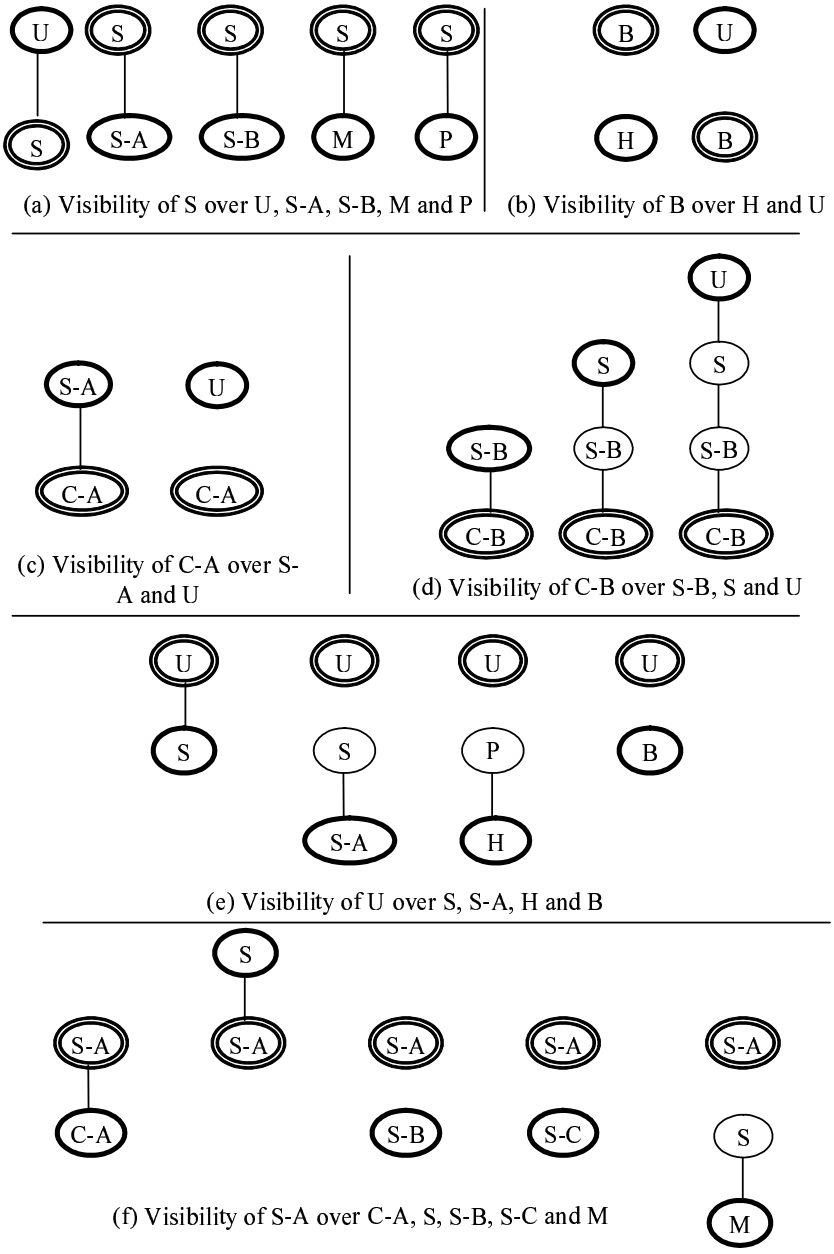

FIGURE 3.2 – SoVs of some of the services in hierarchy H of Fig. 3.1.

$S-B$, M and P, described above, are referred to as *strong visibilities*, meaning that the "structures" of the nodes $S-A$, $S-B$, M and P, relative to S, in the hierarchy are also visible to S. (Formal definitions are given in the next section.)
- In addition to strong visibility over supplier $S-A$, the courier company $C-A$ needs weak visibility over the customer U to get the address details (Fig. 3.2(c)).
- The courier company $C-B$ has strong visibility over $S-B$, S and U (Fig. 3.2(d)).
- The visibilities of U are described in Fig. 3.2(e) : (i) the visibility over S is strong ; (ii) the visibility over B is weak ; and (iii) the visibilities over $S-A$ and H are of "intermediate strength" ; we call this *partially strong visibility*. The partially strong visibility of U over H may be interpreted, for the service details attribute, as : U can get the service details of H directly, or from P. Note that U does *not* have visibility over P. Thus, U can get the service details of H from P, but not the service details of P itself. In addition, P itself may not have visibility over H. In that case, P gets the service details of H only to forward to U and not for its own use.
- Some non-hierarchical (horizontal) visibilities are illustrated in Fig. 3.2(b) (B over H) and Fig. 3.2(f) ($S-A$ over $S-B$, $S-C$, and M).
- Visibilities need not be symmetric. For example, S does not have visibility over $C-B$ (Fig. 3.2(a)), perhaps due to $S-B$ wishing to hide the details of its courier company $C-B$ from S due to competitive reasons, whereas $C-B$ has visibility over S (Fig. 3.2(d)).
- Visibilities of the providers in the hierarchy need not be related. For example, U has weak visibility over B (Fig. 3.2(e)) to ensure that the bonus air miles are credited, but U's child S does not have visibility over B (Fig. 3.2(a)).

3.2 Formal SoV

In the sphere of visibility of X, we identify the services visible to X and their "type" of visibility. First, we introduce some terminology. We consider a hierarchy H as an undirected tree. Throughout this section, we refer to a generic *visibility assignment* $\mathcal{V}$ in H with respect to an attribute A. $\mathcal{V}$ consists of the set of subgraphs $\mathcal{V}[X,Y]$, for all pairs X, Y of nodes in H, defined as follows : $\mathcal{V}[X,Y]$ is either (i) a connected subgraph of $H[X,Y]$ that contains Y, or (ii) the null graph, meaning that X does not have visibility over Y. $\mathcal{V}[X,Y]$ denotes the *type*, also *strength*, of visibility X has over Y. We assume that $\mathcal{V}[X,X]$, for every X, is the graph containing just the node X.

Definition 3.1 (Sphere of Visibility) The *Sphere of Visibility* of a node X in hierarchy H, denoted SoV_X, is the set of non-null subgraphs $\mathcal{V}[X,Y]$, for Y in H.

We say that X has a *weak visibility* to any node Y that is visible to X, that is, when $\mathcal{V}[X,Y]$ is non-null. If $\mathcal{V}[X,Y]$ has some edges then we say that X has a *partially strong visibility* to Y. If $\mathcal{V}[X,Y]$ is $H[X,Y]$, that is, it has all the nodes and edges in the path from X to Y in H, then we say that X has a *strong visibility* to Y. The weak, partially strong and strong visibilities are also denoted as *weak, partially strong* and

strong references, respectively. We use the following notation to represent the strength of visibility. If $\mathcal{V}[X, Y]$ is $H[Y_1, Y]$, for Y_1 in the path from X to Y, then it is denoted as $\langle Y_1, Y\rangle$. Then, strong reference will be $\langle X, Y\rangle$, and weak reference will be $\langle Y, Y\rangle$ or simply $\langle Y\rangle$.

In this chapter, we are not concerned about the semantics of the visibility assignments, that is, whether $\mathcal{V}[X, Y]$, for a pair of nodes X and Y, should be null, weak, partially strong or strong visibility. This is highly application-dependent. We are concerned only with properties and representations of visibility assignments.

As mentioned earlier, the visibilities are defined with respect to some attributes. In this sense, the underlying hierarchical organization of the nodes may be known to all the nodes and may even be used to forward information from one node to another, but knowledge of the values of attributes at the nodes may be specified by the visibility assignments. We illustrate this with two examples.

Application-SD : Here we consider *service details* attribute in hierarchical Web service composition. Service details in this context may include details necessary for say compensating an execution of a service. For example, a hotel reservation made by a travel agency for an user can be canceled directly by the user if the details are known to the user. Otherwise, the user has to contact the travel agency for canceling the reservation.

Here, the visibility $\mathcal{V}[X, Y] = \langle Y_1, Y\rangle$ can be attributed to the following : X can get the service details of Y from any of the nodes in the path $\langle Y_1, Y\rangle$. Thus, in the case of weak visibility, Y_1 is Y meaning that the details can be obtained directly and only from Y. In the case of strong visibility, Y_1 is X meaning that the details are readily available at X itself. Null value indicates that the details at Y are not available to X.

Application-OUTSOURCING : We consider a simplified outsourcing scenario. A plant (for example, a car plant) manufactures some products. It uses, as sub-products, some products manufactured in some other plants, which in turn may get some products from other plants and so on. The manufacturing of each product must adhere to certain (governmental, environmental, etc.) regulations. Some regulations may be local. Some others may be inherited from ancestral entities. This can be depicted in a hierarchy as follows :

- The nodes represent different sites (plants) and the edges represent parent-child outsourcing relationships.
- Each site manufactures some products which may be used as sub-products manufactured in other sites. One site may manufacture products for several sites, and a site may get (sub-)products from several sites. The products manufactured by site S_i for site S_j are denoted P_{ij}.
- The regulations at site S_i that are applicable to *every* product manufactured at that site are denoted R_i.
- In addition, the regulations of one or more additional sites must be adhered to for the products sent to other sites. We denote the regulations for (the products manufactured by S_i for S_j) P_{ij} as R_{ij}.

Here R_{ij} can be modeled as $\mathcal{V}[S_i, S_j]$. If the value is $\langle S_k, S_j \rangle$, for node S_k in the path from S_i to S_j, then it means that R_{ij} is the union of the regulations at all the sites from S_k to S_j. Here, weak visibility means only the local regulations are met. Strong visibility means the regulations of each intermediate node in the path from S_i to S_j are adhered to. Null visibility may mean that no regulations need to be adhered to, perhaps because P_{ij} is empty, that is, S_i does not manufacture anything for S_j.

In general, a visibility assignment may prescribe $\mathcal{V}[X, Y]$ for each pair of nodes X and Y individually. As in the e-shopping scenario of Section 3.1, with such a general assignment, the SoVs of the various nodes can be quite arbitrary and unrelated to each other. However, when the SoVs are related in some way, meaningful visibility assignments can be obtained. The relationship among the SoVs may be influenced by the type and intended use of the attributes over which visibility is sought, whether the hierarchy is intra-organizational (where the visibilities can be very flexible) or inter-organizational (where the visibilities need to be restricted due to trust and security concerns), hierarchical control and authorization policies, etc. Meaningful relationships among the SoVs may also help formulate global visibility policies and the individual assignments based on these policies. In addition to simplifying the logic behind the individual assignments, this will facilitate defining visibility characteristics even for nodes that have not joined the hierarchy yet, in a dynamic environment. In the sequel, we identify two special visibility relationships among the SoVs, *coherence* and *correlation*. These are intuitive and at the same time appear very fundamental for several applications.

3.2.1 Coherence

Definition 3.2 (Coherent Visibility) A visibility assignment $\mathcal{V}$ is *coherent* if for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, $\mathcal{V}[X, Z] \cap H[X, Y]$ is a subgraph of $\mathcal{V}[X, Y]$.

Nodes X, Y and Z are illustrated in Fig. 3.3. We use this figure to comprehend and relate the different properties in this chapter. We will refer to the node W (in the figure) also some times. Informally, coherence means that the strength of visibility of X over Y is *at least* as much as the strength used for visibility of X over Z : $\mathcal{V}[X, Z] \cap H[X, Y]$ refers to the strength of visibility of X over Y "used" for visibility over Z, whereas $\mathcal{V}[X, Y]$ is simply the strength of visibility of X over Y. It can be used to enforce the following visibility policy :

Coherent Visibility Policy : If X can get the service details of Z from W, then (i) X has visibility over all the nodes in the path from W to Z, and (ii) X can get their service details also from W.

For instance, in Application-SD, coherence implies that it is not the case that X could get the service details of Z from W but not the service details of Y from W. However, it is possible that X could get the service details of Y from W but not the service details of Z from W. In Application-OUTSOURCING, coherence implies that

FIGURE 3.3 – Sample Path.

if the regulations at W are applicable to products manufactured at Z for X, then they are applicable to products manufactured at Y also, again for X. Here also, it is possible that the regulations at W are applicable to products manufactured at Y for X, but not for those manufactured at Z for X.

Fig. 3.4 describes several pairs of $\mathcal{V}[X, Z]$ and $\mathcal{V}[X, Y]$ in coherent and non-coherent visibility assignments.

3.2.2 Correlation

Correlation property captures the intuitive notion that, referring to Fig. 3.3 again, the strength of visibility of Y over Z is *at least* as much as the strength of visibility of X over Z restricted to $H[Y, Z]$. In Application-SD, in general, Y can forward the details of Z to X, without itself having visibility over Z, that is, act simply as a propagator. Here, correlation property can be used to impose the following visibility policy :

Correlated Visibility Policy : Y is allowed to forward the service details of Z to X (or any other node) only if Y can get the service details of Z for itself.

However, correlation property does allow the possibility that Y can get the service details of Z for itself, but does not forward them to X. In Application-OUTSOURCING, if the regulations at Y are applied to products manufactured at Z for X, then they must be applied for those manufactured (at Z) for Y also. Again, it is possible that the regulations at Y are applicable to the products manufactured at Z for Y, but not to the products manufactured (at Z) for X.

Definition 3.3 (Correlated Visibility) A visibility assignment $\mathcal{V}$ is *correlated* if for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, $\mathcal{V}[X, Z] \cap H[Y, Z]$ is a subgraph of $\mathcal{V}[Y, Z]$.

Fig. 3.5 illustrates correlation. Informally, along the path from Z to X, the strength of visibility over Z may first be increasing, and then remain the same, and finally be decreasing. At some stage, Z may not be visible and, if so, it remains invisible to all the other nodes along the path.

We note that coherence and correlation are orthogonal properties. Fig. 3.6 shows a visibility assignment which is coherent but not correlational : X_1 is visible to X_4, but

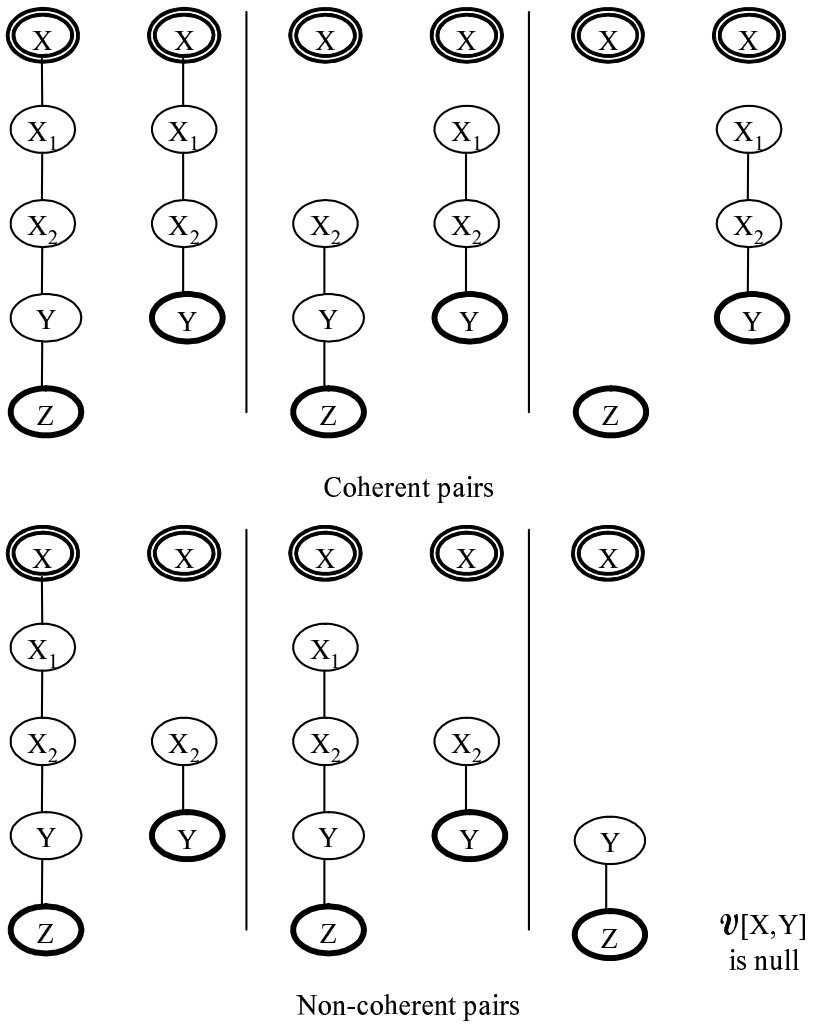

FIGURE 3.4 – Coherent and non-coherent pairs.

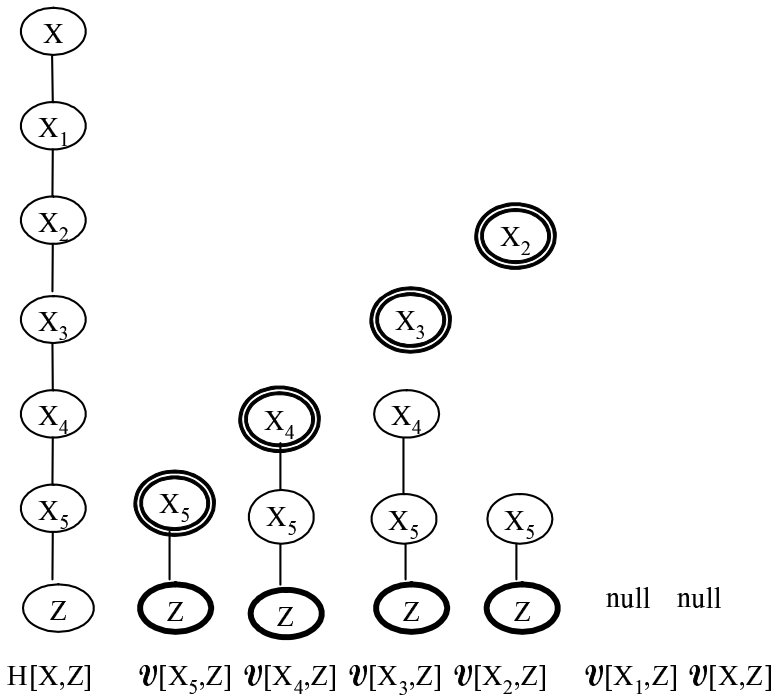

FIGURE 3.5 – Correlation.

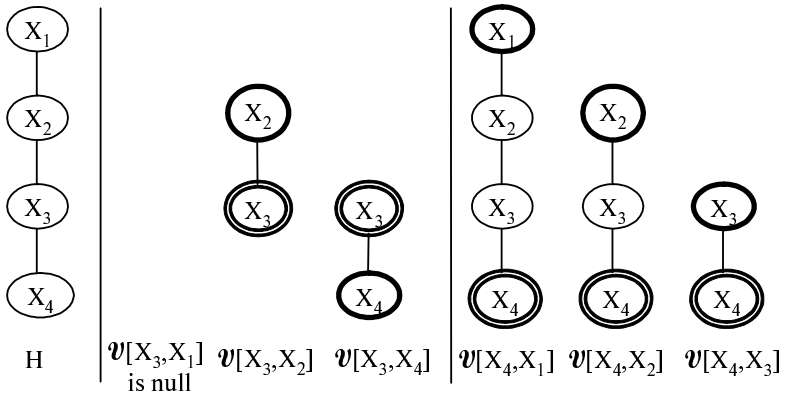

FIGURE 3.6 – Illustration of coherent, but not correlated, SoVs.

not to its parent X_3. Fig. 3.7 shows a visibility assignment which is correlated but not coherent : $\mathcal{V}[X_4, X_1] \cap H[X_4, X_2]$ is not a subgraph of $\mathcal{V}[X_4, X_2]$.

3.2.3 Related Properties

In this section, we define some variants of the coherence and correlation properties that will be useful in some application environments. First we consider coherence. By replacing "subgraph" by "supergraph" in the definition of coherence, we get inverse coherence. For graphs G and G', we call G a *supergraph* of G' if G' is a subgraph of G.

Definition 3.4 (Inversely Coherent Visibility) A visibility assignment $\mathcal{V}$ is *inversely coherent* if for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, $\mathcal{V}[X, Z] \cap H[X, Y]$ is a supergraph of $\mathcal{V}[X, Y]$.

Informally, inverse coherence means that the strength of visibility of X over Y is *at most* as much as the strength used for visibility of X over Z. For instance, in Application-SD, inverse coherence implies that it is possible that X could get the service details of Z from W but not the service details of Y from W. Analogous to coherence, it can be used to enforce the following visibility policy :

Inversely Coherent Visibility Policy : If X can get the service details of Y from W, then (i) X has visibility over all the nodes in the path from Y to Z, and (ii) X can get their service details also from W.

Note the difference in terms of the set of nodes affected by the coherent and inversely coherent visibility assignments. In Application-OUTSOURCING, inverse coherence implies the possibility that the regulations at W are applicable to products manufactured at Z for X, but not to products manufactured at Y, again for X. A visibility assignment

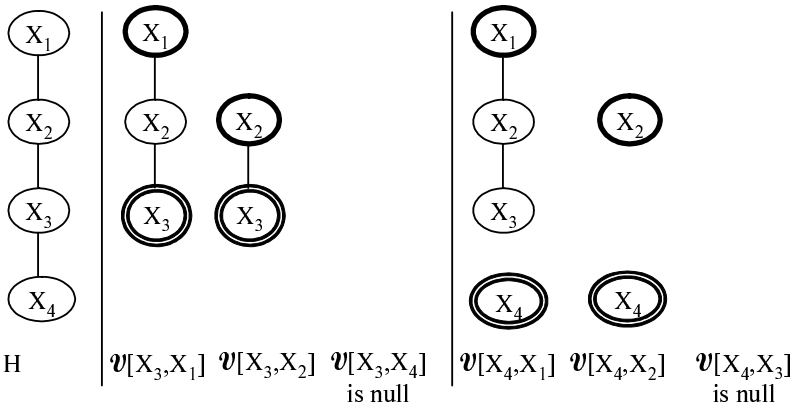

FIGURE 3.7 – Illustration of correlated, but not coherent, SoVs.

where adjacent nodes are weakly visible, while non-adjacent nodes are strongly visible, shown in Fig. 3.8, is inversely coherent.

Another variant of the coherence property is the following. Here, "subgraph" in the definition of coherence is replaced by "equal to".

Definition 3.5 (Uniformly Coherent Visibility) A visibility assignment $\mathcal{V}$ is *uniformly coherent* if for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, $\mathcal{V}[X,Z] \cap H[X,Y]$ is equal to $\mathcal{V}[X,Y]$.

For instance, in Application-SD, uniform coherence implies that for every node W such that X could get the service details of Z from W, it could get the service details of Y also from W, and vice versa. In Application-OUTSOURCING, uniform coherence implies for every node W such that the regulations at W are applicable to products manufactured at Z for X, those regulations are applicable to products manufactured at Y also, again for X, and vice versa. Note that uniform coherence is the same as uniform inverse coherence. Fig. 3.9 illustrates the notion of uniform coherence.

We now consider variants of the correlation property. The definitions are similar to those of coherence.

Definition 3.6 (Inversely Correlated Visibility) A visibility assignment $\mathcal{V}$ is *inversely correlated* if for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, $\mathcal{V}[X,Z] \cap H[Y,Z]$ is a supergraph of $\mathcal{V}[Y,Z]$.

For example, in Application-SD, inverse correlation means that Y can forward the service details of Z to X, but not to W. That is, it can be used to enforce the following visibility policy :

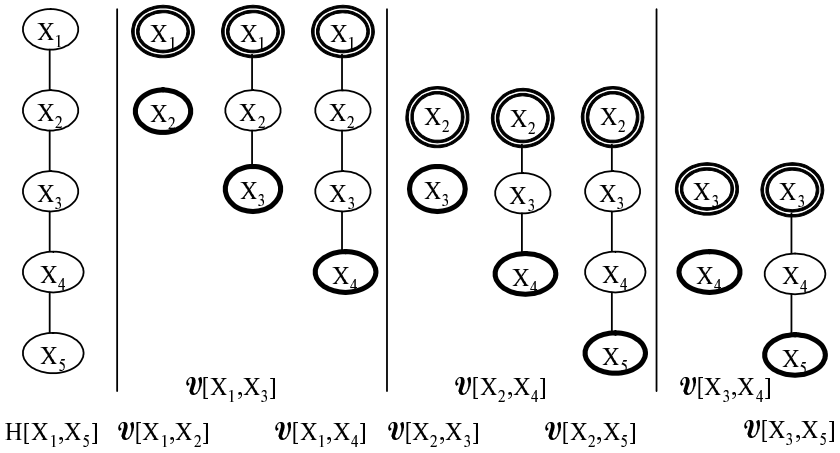

FIGURE 3.8 – Inverse coherence.

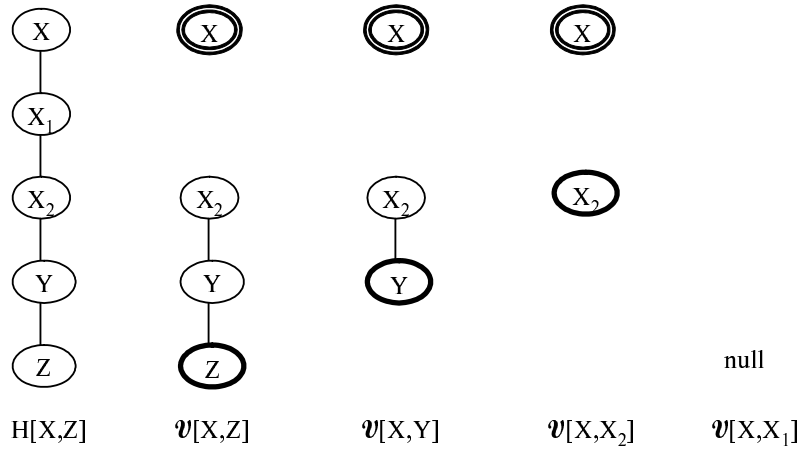

FIGURE 3.9 – Uniform coherence.

Inversely Correlated Visibility Policy : If W can get the service details of Z from Y, then (i) all nodes in the path from X to W have visibility over the service details of Z, and (ii) they can get those service details from Y also.

Again, note the difference in the set of affected nodes in comparison to (inverse) coherence. While coherence imposes conditions on a node's visibility over a set of other nodes, correlation affects the visibility of a set of nodes over a node (which is basically the noticeability of that node, formalized in Section 3.3). In Application-OUTSOURCING, inverse correlation means that it is possible that the regulations at Y are applied to products manufactured at Z for X, but not for those manufactured (at Z) for W.

It is easy to check that the visibility assignment in Fig. 3.8, where adjacent nodes are weakly visible and non-adjacent nodes are strongly visible, is also inversely correlated (in addition to being inversely coherent).

Definition 3.7 (Uniformly Correlated Visibility) A visibility assignment $\mathcal{V}$ is *uniformly correlated* if for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, $\mathcal{V}[X,Z] \cap H[Y,Z]$ is equal to $\mathcal{V}[Y,Z]$.

For example, in Application-SD, uniform correlation means that Y can forward the service details of Z to X if and only if it can get those details for itself. In Application-OUTSOURCING, uniform correlation means that the regulations at Y are applied to products manufactured at Z for X if and only if they are applied for those manufactured (at Z) for Y.

Clearly, it is possible to have global visibility policies which characterize a mix of coherence and correlation properties. One such policy is already illustrated in Fig. 3.8.

Visibility Policy 1 : Weak visibility to adjacent nodes and strong visibility to all other nodes.

As mentioned earlier, this visibility has inverse coherence and inverse correlation properties.

We illustrate the next couple of policies with the path ($X = Y_5, Y_4, Y_3, Y_2, Y_1, Y_0 = Y$).

Visibility Policy 2 : (Fig. 3.10) Strong visibility to nodes at distance 1 or 2, and partially strong visibility consisting of a path with two edges to all other nodes. That is, $\mathcal{V}[X,Y]$ is :

- $\langle X, Y\rangle$ if Y is adjacent to or at distance 2 from X, and
- $\langle Y_2, Y\rangle$ if Y is at distance greater than 2 from X, where Y_2 is the third last node in the path from X to Y.

This visibility assignment is coherent and uniformly correlated.

Example of coherence is,

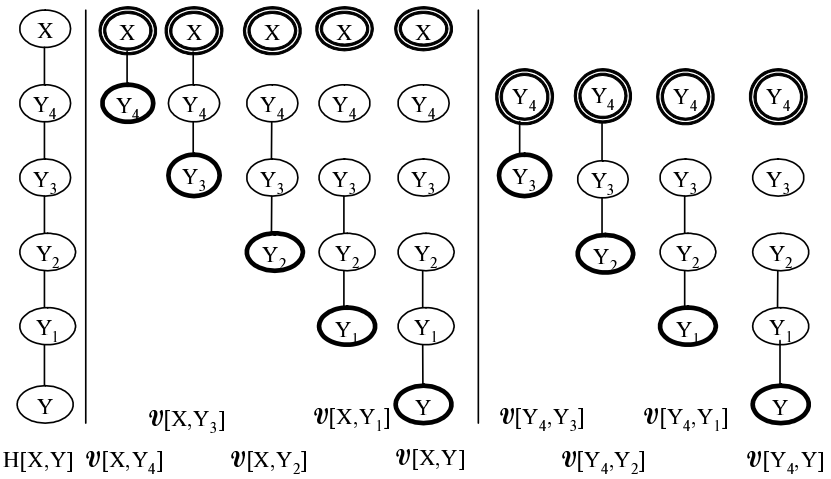

FIGURE 3.10 – Illustration for visibility policy 2.

– $\mathcal{V}[X,Y] \cap H[X,Y_2]$ is $\langle Y_2 \rangle$, and is a subgraph of
– $\mathcal{V}[X,Y_2]$, which is $\langle Y_4, Y_2 \rangle$.

An example of uniform correlation is,

– $\mathcal{V}[X,Y] \cap H[Y_4,Y]$ is $\langle Y_2, Y \rangle$, and is equal to
– $\mathcal{V}[Y_4,Y]$ which is also $\langle Y_2, Y \rangle$.

On the other hand, a policy does not have to satisfy any of the properties we have discussed. For example, the visibility policy below does not satisfy any of the properties.

Visibility Policy 3 : (Fig. 3.11) Strong visibility to nodes at distance 1 and 2, weak visibility to nodes at distance 3 or 4, and strong visibility to all other nodes.

Example of coherence violation is, with X, Y_1 and Y,

– $\mathcal{V}[X,Y]$ is $\langle X, Y \rangle$.
– $\mathcal{V}[X,Y] \cap H[X,Y_1] = \langle X, Y_1 \rangle$, which is not a subgraph of
– $\mathcal{V}[X,Y_1]$ is $\langle Y_1 \rangle$.

Example of inverse coherence violation is, with X, Y_1 and Y_3,

– $\mathcal{V}[X,Y_1]$ is $\langle Y_1 \rangle$.
– $\mathcal{V}[X,Y_1] \cap H[X,Y_3]$ is null, which is not a supergraph of
– $\mathcal{V}[X,Y_3]$ is $\langle X, Y_3 \rangle$.

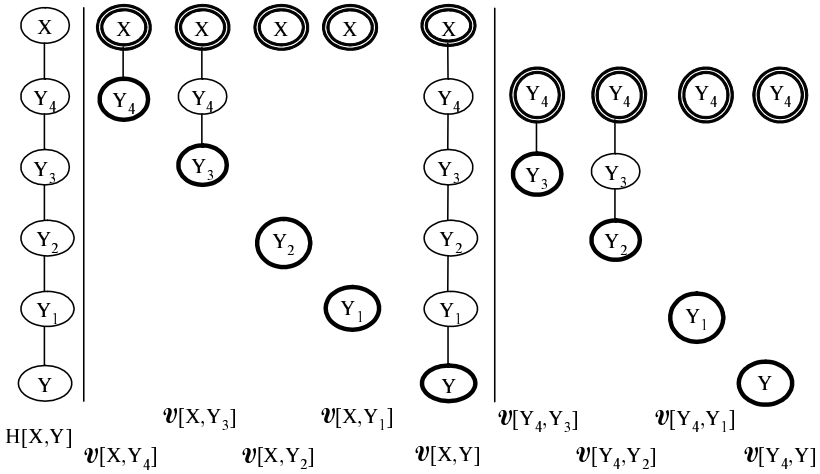

FIGURE 3.11 – Illustration for visibility policy 3.

An example of correlation violation is, with X, Y and Y_3,

- $\mathcal{V}[X, Y]$ is $\langle X, Y \rangle$.
- $\mathcal{V}[X, Y] \cap H[Y_3, Y]$ is $\langle Y_3, Y \rangle$, which is not a subgraph of
- $\mathcal{V}[Y_3, Y]$ is $\langle Y \rangle$.

An example of inverse correlation violation is, with X, Y_2 and Y_4,

- $\mathcal{V}[X, Y_2]$ is $\langle Y_2 \rangle$.
- $\mathcal{V}[X, Y_2] \cap H[Y_4, Y_2]$ is $\langle Y_2 \rangle$ which is not a supergraph of
- $\mathcal{V}[Y_4, Y_2]$ is $\langle Y_4, Y_2 \rangle$.

3.3 Sphere of Noticeability

For a service X, the Sphere of Noticeability notion is intended to capture : (i) which services have visibility over X ; and (ii) what type of visibility they have of X. First we define noticeability independent of, but in a way analogous to the definition of, visibility. We refer to a general *noticeability assignment* $\mathcal{N}$ in H with respect to an attribute A. $\mathcal{N}$ consists of a set of subgraphs $\mathcal{N}[X, Y]$, for all pairs X, Y of nodes in H, defined as follows : $\mathcal{N}[X, Y]$ is either (i) a connected subgraph of $H[X, Y]$ that contains X, or (ii) the null graph. $\mathcal{N}[X, Y]$ denotes the *type*, also *strength*, of noticeability, that is, the type of visibility Y has over X. In the last case, X is not noticed by Y. We assume that $\mathcal{N}[X, X]$, for every X, is the graph containing just the node X. Note that since visibility and noticeability notions are complementary, $\mathcal{V}$ and $\mathcal{N}$ definitions are also

complementary. That is, for $\mathcal{N}$ that "corresponds to" a $\mathcal{V}$, for X and Y, $\mathcal{N}[X,Y]$ is the same as $\mathcal{V}[Y,X]$. We use $\mathcal{N}[X,Y]$ most of the time in the definitions and discussions in this section, though $\mathcal{V}[Y,X]$ could also be used instead. In the illustration of noticeability of X by Y, X is represented in an oval inside a rectangle, Y is represented by a thick oval, and other nodes, if any, are represented in thin ovals.

Definition 3.8 (Sphere of Noticeability) The *Sphere of Noticeability* of a node X in hierarchy H, denoted SoN_X, is the set of non-null subgraphs $\mathcal{N}[X,Y]$, for Y in H.

Note that, for a specific node X, SoV_X is the set of $\mathcal{V}[X,Y]$s for different Ys, whereas SoN_X is the set of $\mathcal{V}[Y,X]$s for different Ys. An obvious application of SoN is for change management. An example is a provider X notifying the providers, who have visibility over X, when there is some change in the provider URI (provider details), metrics used to compute the service (service details), log format (execution details), etc. An interpretation of the relationship between SoV and SoN using the e-shopping scenario introduced earlier (Fig. 3.1) follows. For a node X, SoV_X can be considered as the nodes from which some information (input) is expected, and SoN_X can be considered as the nodes to which some information (output) is to be sent. In both cases, the type of visibility reflects how the information may be received or sent. For example (Figures 3.1 and 3.2), for the air miles provider B, SoV_B conveys that B is expecting the credit charge information from H and the air miles account details from U. On the other hand, SoN_B may contain U, P and H, reflecting that B should send confirmation of the air miles reward to U, P and H.

Coherence and correlation properties for noticeability assignments can be defined analogous to those for visibility assignments.

Definition 3.9 (Coherence Properties of Noticeability) A noticeability assignment $\mathcal{N}$ is *coherent* (respectively, *inversely coherent*, *uniformly coherent*) if for each pair of nodes Z and X, and every node Y in the path from Z to X in H, Y not equal to Z or X, $\mathcal{N}[Z,X] \cap H[Z,Y]$ is a subgraph of (respectively, a supergraph of, equal to) $\mathcal{N}[Z,Y]$.

Definition 3.10 (Correlation Properties of Noticeability) A noticeability assignment $\mathcal{N}$ is *correlated* (respectively, *inversely correlated*, *uniformly correlated*) if for each pair of nodes Z and X, and every node Y in the path from Z to X in H, Y not equal to Z or X, $\mathcal{N}[Z,X] \cap H[Y,X]$ is a subgraph of (respectively, a supergraph of, equal to) $\mathcal{N}[Y,X]$.

We now show that the coherence and correlation properties are indeed related.

Proposition 3.1 In a hierarchy H, a visibility assignment $\mathcal{V}$ is coherent (respectively, inversely coherent, uniformly coherent) if and only if the corresponding noticeability assignment $\mathcal{N}$ is correlated (respectively, inversely correlated, uniformly correlated), and vice versa.

Proof The proof follows from the complementary relationship between visibility and noticeability. Recall that, for $\mathcal{N}$ that "corresponds to" a $\mathcal{V}$, for X and Y, $\mathcal{N}[X, Y]$ is the same as $\mathcal{V}[Y, X]$. Applying the above transformation to the coherent visibility definition, we get the correlated noticeability definition and vice versa. The same applies to the other equivalences. □

3.4 Implementation Issues

Consider a node X of a static hierarchy H. Initially, X has some visibility requirements and noticeability restrictions. Note that the above reflects only X's point of view, that is, visibility and noticeability that X would like. As such, the final assigned SoV_X may be quite different from its initial expectations due to conflicts with the noticeability restrictions of nodes Y over which X would like to have visibility. The same applies for SoN_X also. That is, let us assume that X would like to have visibility over Y, but Y does not want to be visible to X. Given this, there are two possibilities : (i) Y's restriction cannot be overruled and so X is not allowed to have visibility over Y, and (ii) X's requirement has higher priority leading to negotiation with Y, and X gets visibility over Y. In addition, X's visibility (noticeability) over (of) Y may be affected by the requirements and restrictions of the remaining nodes in H, global coherence/correlation properties, etc. Often in a Web services context, it is not the service provider itself but some higher level logical entity or an agent acting on behalf of the provider which is responsible for regulating the visibility of a service. For example, with reference to the e-shopping scenario in Fig. 3.1, let us assume that $S - A$ would like to have visibility over the courier companies (such as, $C - B$) used by other suppliers to find the cheapest option. On the same lines, $S - B$ might like to keep the details of its courier company $C - B$ hidden due to competitive reasons and should be in a position to reject $S - A$'s request for visibility over $C - B$. Resolving such conflicts is application/domain dependent and beyond the scope of this work. Basically, we assume the existence of a non-conflicting set of visibility requirements for H.

3.4.1 Assignment

In this section, we outline a simple scheme for adjusting SoVs (and SoNs) of all the nodes dynamically, each time a node Y is added to the hierarchy. The first step is setting the SoV of the new node Y. Then, by traversing the entire hierarchy, the SoVs of all the nodes are adjusted, which is equivalent to setting SoN_Y. Let us look at the adjustment steps, to be taken at a node X, due to the addition of Y in H, in detail.

Let $Y = Y_0, Y_1, Y_2, \cdots, Y_i, \cdots, Y_j, \cdots, Y_k, \cdots, Y_n, Y_{n+1} = X$, be the nodes in the path from Y to X such that $k > j > i$ (shown diagrammatically in 3.12). Then, information about Y reaches X via Y_n. Node X has to do the following things :

1. decide whether to include Y in X's SoV ;

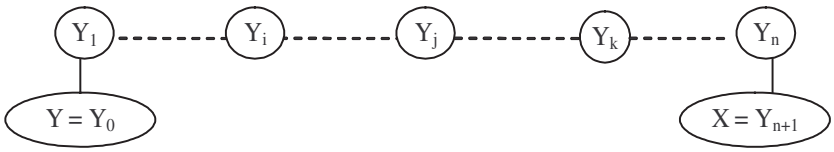

FIGURE 3.12 – Illustration to show visibility assignments.

2. if the answer to (1) is yes, decide how to include Y in SoV_X ; Couple of options are using : (i) a strong reference to Y_n, that is, including the edge from Y_n to X also in SoV_X ; or (ii) a weak reference to Y_i, for some i, or to Y itself.
3. irrespective of the decision, send information about Y to all neighbors of X other than Y_n.

The decisions in the above three steps can be made so as to obtain the desired coherence and correlation properties of the SoN of Y, and hence the related properties of the SoVs of other nodes. Obviously, the above algorithm is a very simple one. Optimizations are possible for special cases. For example, if node X does not include Y in SoV_X, it may not have to send information on Y to the other neighbors. Also, it may send information on Y only to some neighbors, not all. We consider such optimizations for coherence and correlated visibility and noticeability in the rest of this section.

Steps to define $\mathcal{V}[X,Y]$*, that is, determining* j *in* $\mathcal{V}[X,Y] = \langle Y_j, Y\rangle$ *:*

- $\mathcal{V}[X,Y]$ could be null. In the following, we consider non-null options.
- For non-coherent $\mathcal{V}$, j could be any value between 0 and $n+1$.
- For coherent $\mathcal{V}$, if $\mathcal{V}[X,Y_1]$ is null, then $\mathcal{V}[X,Y]$ has to be Y, that is, (weak reference to) just the node Y. Otherwise, $j \leq k$, where $\mathcal{V}[X,Y_1] = \langle Y_k, Y_1\rangle$. This will also give correlated $\mathcal{N}$.
- For inversely coherent $\mathcal{V}$, if $\mathcal{V}[X,Y_1]$ is strong, then $\mathcal{V}[X,Y]$ has to be strong. Otherwise, $j \geq i$, where $\mathcal{V}[X,Y_1] = \langle Y_i, Y_1\rangle$.
- For uniformly coherent $\mathcal{V}$, if $\mathcal{V}[X,Y_1]$ is null (strong), then $\mathcal{V}[X,Y]$ is weak (strong), that is, Y. Otherwise, $j = k$, where $\mathcal{V}[X,Y_1] = \langle Y_k, Y_1\rangle$.
- For correlated $\mathcal{V}$, if $\mathcal{V}[Y_n,Y]$ is null, then $\mathcal{V}[X,Y]$ must be null too. Otherwise, $j \leq k$, where $\mathcal{V}[Y_n,Y] = \langle Y_k, Y\rangle$. This will also give coherent $\mathcal{N}$.
- For inversely correlated $\mathcal{V}$, $j \geq i$, where $\mathcal{V}[Y_n,Y] = \langle Y_i, Y\rangle$.
- For uniformly correlated $\mathcal{V}$, if $\mathcal{V}[Y_n,Y]$ is null, then $\mathcal{V}[X,Y]$ must be null too. Otherwise, $j = k$, where $\mathcal{V}[Y_n,Y] = \langle Y_k, Y\rangle$.

Steps to define $\mathcal{V}[Y,X]$*, that is, determining* j *in* $\mathcal{V}[Y,X] = \langle Y_j, X\rangle$ *:*

- $\mathcal{V}[Y,X]$ could be null. In the following, we consider non-null options.

- For non-coherent $\mathcal{V}$, j could be any value between 0 and $n+1$.
- For coherent $\mathcal{V}$, if $\mathcal{V}[Y, Y_n]$ is null, then $\mathcal{V}[Y, X]$ has to be X, that is, (weak reference to) just the node X. Otherwise, $j \geq i$, where $\mathcal{V}[Y, Y_n] = \langle Y_i, Y_n \rangle$. This will also give correlated $\mathcal{N}$.
- For inversely coherent $\mathcal{V}$, if $\mathcal{V}[Y, Y_n]$ is strong, then $\mathcal{V}[Y, X]$ has to be strong. Otherwise, $j \leq k$, where $\mathcal{V}[Y, Y_n] = \langle Y_k, Y_n \rangle$.
- For uniformly coherent $\mathcal{V}$, if $\mathcal{V}[Y, Y_n]$ is null (strong), then $\mathcal{V}[Y, X]$ is weak (strong), that is, X. Otherwise, $j = k$, where $\mathcal{V}[Y, Y_n] = \langle Y_k, Y_n \rangle$.
- For correlated $\mathcal{V}$, if $\mathcal{V}[Y_1, X]$ is null, then $\mathcal{V}[Y, X]$ must be null too. Otherwise, $j \geq i$, where $\mathcal{V}[Y_1, X] = \langle Y_i, X \rangle$. This will also give coherent $\mathcal{N}$.
- For inversely correlated $\mathcal{V}$, $j \leq k$, where $\mathcal{V}[Y_1, X] = \langle Y_k, X \rangle$.
- For uniformly correlated $\mathcal{V}$, if $\mathcal{V}[Y_1, X]$ is null, then $\mathcal{V}[Y, X]$ must be null too; otherwise, $j = k$, where $\mathcal{V}[Y_1, X] = \langle Y_k, X \rangle$.

Such adjustments need to be made for every existing node X.

3.4.2 Single Graph Representation

We would like to optimize the (visual) space required to represent the visibilities of nodes in a hierarchy. Ideally, we would have a graph $\mathcal{V}[X, Y]$ corresponding to each pair of nodes X and Y in the hierarchy, which may become cumbersome for a large hierarchy. As such, we are interested in conditions under which the visibilities (i) of a node over other nodes, and (ii) of all the nodes in a hierarchy, can be represented in a single graph. First, we tackle the former.

Uniform coherence facilitates representing SoV_X, of a node X, in a single graph. The graph is obtained just by merging all the graphs in SoV_X. We denote this graph as $\mathcal{V}_X$. We show the validity of such representation in the following. The connected component of $\mathcal{V}_X \cap H[X, Y]$ that contains Y is denoted $\mathcal{V}_X[X, Y]$. In this work, we extend the use of "$\subseteq$" to denote the subgraph relationship as well in the context of graphs. By an *extension* of a path $\langle X, Y \rangle$ we mean a path $\langle X, Z \rangle$, for some Z, such that $\langle X, Y \rangle$ is a sub-path of $\langle X, Z \rangle$.

Proposition 3.2 For a uniformly coherent visibility assignment $\mathcal{V}$, $\mathcal{V}_X$ represents SoV_X in a "lossless" fashion, that is, $\mathcal{V}_X[X, Y]$ equals $\mathcal{V}[X, Y]$, for every Y.

Proof For any Y, clearly, $\mathcal{V}[X, Y] \subseteq \mathcal{V}_X[X, Y]$. We show the equality. We need to consider only the case where $\mathcal{V}_X[X, Y]$ is non-null. We first show that $\mathcal{V}[X, Y]$ is non-null. Assume the contrary. Then, for some node Z in the extension of the path $\langle X, Y \rangle$, $\mathcal{V}[X, Z]$ must contain the path $\langle Y, Z \rangle$. (Note that Z could be Y or some other node.) Then, $\mathcal{V}[X, Z] \cap H[X, Y]$ is non-null (containing at least the node Y), and it should be equal to $\mathcal{V}[X, Y]$, by uniform coherence. Thus, $\mathcal{V}[X, Y]$ cannot be null.

If $\mathcal{V}[X, Y]$ is strong, that is, $\langle X, Y \rangle$, then clearly, $\mathcal{V}_X[X, Y]$ is also $\langle X, Y \rangle$. Now suppose $\mathcal{V}[X, Y]$ is a proper subgraph of $H[X, Y]$, say $\langle Y_1, Y \rangle$ for some Y_1 in the path from X to Y. We claim that $\mathcal{V}_X[X, Y]$ does not contain any more edges than in $\langle Y_1, Y \rangle$.

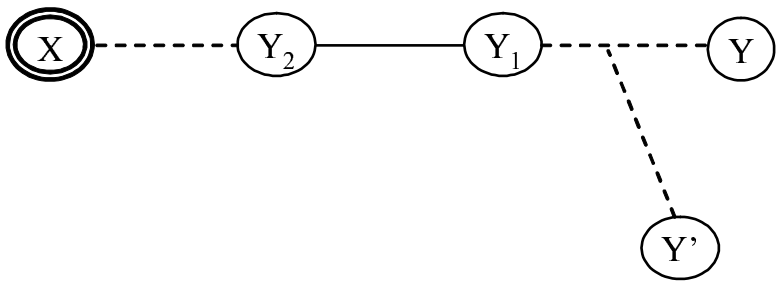

FIGURE 3.13 – Illustration for the proof of Proposition 3.2.

Suppose, on the contrary, that for node Y_2 preceding Y_1 in the path from X to Y, $\mathcal{V}_X[X,Y]$ contains the edge $\langle Y_2, Y_1\rangle$ also. Then, for some Y' in the extension of the path $\langle Y_2, Y_1\rangle$, $\mathcal{V}[X,Y']$ must contain $\langle Y_2, Y'\rangle$. Now, Y' may be in $\langle X,Y\rangle$ or in the extension of $\langle X,Y\rangle$, or may be in a completely different path from Y_1. (An instance of the last case is illustrated in Fig. 3.13.) In all cases, $\mathcal{V}[X,Y'] \cap H[X,Y_1]$ should equal $\mathcal{V}[X,Y_1]$ by uniform coherence and hence contain $\langle Y_2, Y_1\rangle$. However, again by uniform coherence, $\mathcal{V}[X,Y] \cap H[X,Y_1]$ should equal $\mathcal{V}[X,Y_1]$ and so $\mathcal{V}[X,Y]$ should include $\langle Y_2, Y_1\rangle$, a contradiction to the assumption otherwise.

□

Analogously, uniform correlation also facilitates single graph representation, but of the noticeability SoN_X of a node X. We give the proof in terms of coherence, that is, we show that uniformly coherent noticeability facilitates merging all the subgraphs of SoN_X, of a node X, into a single graph. The graph is denoted $\mathcal{N}_X$. The connected component of $\mathcal{N}_X \cap H[X,Y]$ that contains X is denoted $\mathcal{N}_X[X,Y]$.

Proposition 3.3 For a uniformly coherent noticeability assignment $\mathcal{N}$, $\mathcal{N}_X$ represents SoN_X in a "lossless" fashion, that is, $\mathcal{N}_X[X,Y]$ equals $\mathcal{N}[X,Y]$, for every Y.

Proof For any Y, clearly, $\mathcal{N}[X,Y] \subseteq \mathcal{N}_X[X,Y]$. We show the equality. We need to consider only the case where $\mathcal{N}_X[X,Y]$ is non-null. We first show that $\mathcal{N}[X,Y]$ is non-null. Assume the contrary. Then, for some node Z in (i) $H[X,Y]$ or (ii) its extension, $\mathcal{N}[X,Z]$ must contain at least the node X, that is, the path $\langle X\rangle$. In case (i), $\mathcal{N}[X,Y] \cap H[X,Z]$ is null and so not equal to $\mathcal{N}[X,Z]$, violating uniform coherence. In case (ii), $\mathcal{N}[X,Z] \cap H[X,Y]$ is non-null and should be equal to $\mathcal{N}[X,Y]$ by uniform coherence, contrary to the assumption.

If $\mathcal{N}[X,Y]$ is strong, that is, $\langle X,Y\rangle$, then clearly, $\mathcal{N}_X[X,Y]$ is also $\langle X,Y\rangle$. Now suppose $\mathcal{N}[X,Y]$ is a proper subgraph of $H[X,Y]$, say $\langle X,Y_1\rangle$ for some Y_1 in the

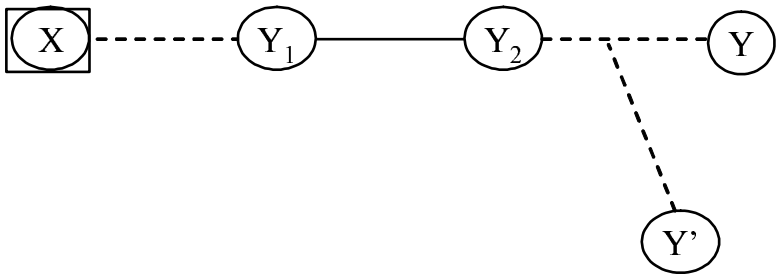

FIGURE 3.14 – Illustration for the proof of Proposition 3.3.

path from X to Y. We claim that $\mathcal{N}_X[X,Y]$ does not contain any more edges than in $\langle X, Y_1\rangle$. Suppose, on the contrary, that for a node Y_2 following Y_1 in the path from X to Y, $\mathcal{N}_X[X,Y]$ contains the edge $\langle Y_1, Y_2\rangle$. Then, for some Y' in the extension of the path $\langle Y_1, Y_2\rangle$, $\mathcal{N}[X,Y']$ must contain $\langle Y_1, Y_2\rangle$. Now Y' may be in $\langle X, Y\rangle$ or in the extension of $\langle X, Y\rangle$, or may be in a completely different path from Y_2. The last case is illustrated in Fig. 3.14. In all cases, $\mathcal{N}[X,Y'] \cap H[X,Y_2]$ should equal $\mathcal{N}[X,Y_2]$ by uniform coherence, and hence contain $\langle Y_1, Y_2\rangle$. However, again by uniform coherence, $\mathcal{N}[X,Y] \cap H[X,Y_2]$ should equal $\mathcal{N}[X,Y_2]$ and so $\mathcal{N}[X,Y]$ must contain the edge $\langle Y_1, Y_2\rangle$ also, a contradiction to the assumption otherwise.

□

Combining Propositions 3.2 and 3.3 with Proposition 3.1, we can state the following :

Proposition 3.4 For a uniformly correlated noticeability assignment $\mathcal{N}$, $\mathcal{V}_X$ represents SoV_X in a "lossless" fashion, that is, $\mathcal{V}_X[X,Y]$ equals $\mathcal{V}[X,Y]$, for every Y.

Proposition 3.5 For a uniformly correlated visibility assignment $\mathcal{V}$, $\mathcal{N}_X$ represents SoN_X in a "lossless" fashion, that is, $\mathcal{N}_X[X,Y]$ equals $\mathcal{N}[X,Y]$, for every Y.

We have shown that uniform coherence and correlation facilitate single graph representation of a node X. For different Xs, the graphs $\mathcal{V}_X$s will also be different. In the remaining part, we show that under certain special conditions, the SoVs of all nodes in H can be represented in a single graph. We need some additional definitions before giving the proposition.

First, we define a variation of the uniform coherence property. Recall that in uniform coherence, if $\mathcal{V}[X,Z] \cap H[X,Y]$ is null, then $\mathcal{V}[X,Y]$ must also be null. This must be so irrespective of whether $\mathcal{V}[X,Z]$ itself is null or non-null. We envisage that, in some applications, it is reasonable to have non-null $\mathcal{V}[X,Y]$ when $\mathcal{V}[X,Z]$ is non-null but $\mathcal{V}[X,Z] \cap H[X,Y]$ is null. Note that this possibility is included in the coherence

property definition because the null graph is a subgraph of any graph. Also, this possibility is contrary to the intuitive notion of inverse coherence, namely, the visibility of distant nodes is stronger than that of nearby nodes. We define a special kind of uniform coherence that allows this possibility :

Definition 3.11 (Non-null Uniformly Coherent Visibility) A visibility assignment $\mathcal{V}$ is *non-null uniformly coherent* if : for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, if $\mathcal{V}[X,Z] \cap H[X,Y]$ is non-null, then it is equal to $\mathcal{V}[X,Y]$.

Proposition 3.2 is applicable for non-null uniform coherency also as the proof only uses the "non-null" part of the definition. We explore the distinction between uniform and non-null uniform coherence with the help of Figures 3.15 and 3.16. Fig. 3.15 describes some characteristics of $\mathcal{V}_X$, for a uniformly coherent visibility $\mathcal{V}$. Essentially, if a node T is visible to X with strength $\langle T_1, T\rangle$, then every node T' in the extension of $\langle X, T\rangle$ is visible to X with strength $\langle T_1, T'\rangle$. In Fig. 3.15 :

- Y_i, for each i, is visible with strength $\langle X, Y_i\rangle$;
- Z_3, Z_4 and Z_5 are visible with strengths $\langle Z_3\rangle$, $\langle Z_3, Z_4\rangle$, and $\langle Z_3, Z_5\rangle$, respectively.
- Z_8 is visible with strength $\langle Z_8\rangle$;
- V_1 is visible with strength $\langle V_1\rangle$; and
- W_1 and W_2 are not visible.

Fig. 3.16 illustrates non-null uniformly coherent visibility. Here, in addition to the nodes visible according to Fig. 3.15, we have :

- Z_1 and Z_6 are visible with strengths $\langle X, Z_1\rangle$ and $\langle X, Z_6\rangle$, respectively ; and
- W_1 is visible with strength $\langle X, W_1\rangle$.

Fig. 3.17(a) illustrates a non-null uniformly coherent SoV of the user U in Fig. 3.1. Note the change in $\mathcal{V}[U, S\text{-}A]$ (for uniform coherence) and the addition of $\mathcal{V}[U, P]$ (for coherence), compared to the SoV of U shown in Fig. 3.2(e).

On the same lines, we define the "non-null" variant of uniformly correlated property.

Definition 3.12 (Non-null Uniformly Correlated Visibility) A visibility assignment $\mathcal{V}$ is *non-null uniformly correlated* if : for each pair of nodes X and Z, and every node Y in the path from X to Z in H, Y not equal to X or Z, if $\mathcal{V}[X,Z] \cap H[Y,Z]$ is non-null, then it is equal to $\mathcal{V}[Y,Z]$.

Proposition 3.5 is applicable for non-null uniform correlation as well (as the corresponding proof again uses only the "non-null" part of the definition). We explore the distinction between uniform and non-null uniform correlation with the help of Figures 3.18 and 3.19. Fig. 3.18 describes some characteristics of a uniformly correlated visibility $\mathcal{V}$. Essentially, if a node X is visible to a node T with strength $\langle T_1, X\rangle$, where T_1 is not T (not strong), then every T' in the extension of $\langle X, T\rangle$ has visibility over X with the same strength $\langle T_1, X\rangle$. In Fig. 3.18, X is visible to :

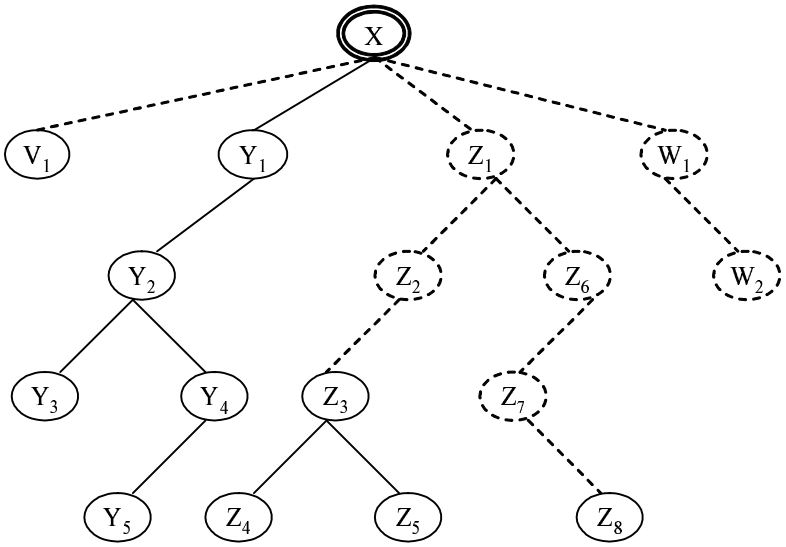

FIGURE 3.15 – Single graph representation - Uniformly coherent visibility.

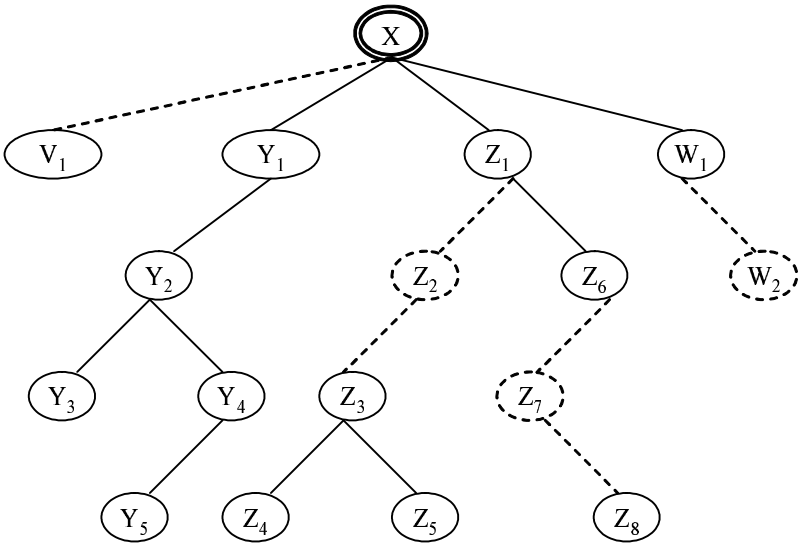

FIGURE 3.16 – Single graph representation - Non-null uniformly coherent visibility.

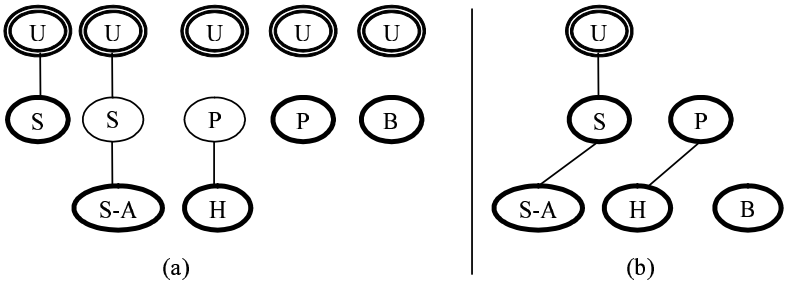

FIGURE 3.17 – Non-null uniformly coherent SoV of the user U in Fig. 3.1.

- Y_1 with strength $\langle X, Y_1 \rangle$;
- each Y_i, for $i > 1$, with strength $\langle X, Y_2 \rangle$;
- V_1 with strength $\langle X \rangle$;
- Z_j, for j equal to 6, 7 and 8, with strength $\langle X, Z_6 \rangle$;
- Z_k, for k from 1 to 5, with strength $\langle X, Z_1 \rangle$; and
- X is not visible to W_1 and W_2.

Fig. 3.19 illustrates a non-null uniformly correlated visibility. Note that X is not visible to Z_3, Z_4 and Z_5. Here, X may be visible to some nodes along a path (X, Y) and may become invisible to Y. If that happens, then X will remain invisible to every node Z in the extension of (X, Y).

As another example, consider Fig. 3.5. For uniform correlation, $\mathcal{V}[X_2, Z]$, $\mathcal{V}[X_1, Z]$ and $\mathcal{V}[X, Z]$ will be the same as $\mathcal{V}[X_3, Z]$. That is, the decreasing phase will not be there (the strength may increase, then remain the same). With non-null uniform correlation, the visibility may become null at some point. If so, it will stay null from thereon. Replacing $\mathcal{V}[X_2, Z]$ with null in Fig. 3.5 gives non-null uniformly correlated visibility.

Finally, we define (strong) symmetric visibility.

Definition 3.13 (sv-symmetricity) A visibility assignment $\mathcal{V}$ is *strong visibility symmetric*, abbreviated *sv-symmetric*, if for every pair of nodes X and Y, if $\mathcal{V}[X, Y]$ is strong (that is, equals $\langle X, Y \rangle$), then $\mathcal{V}[Y, X]$ is also strong (that is, equals $\langle Y, X \rangle$).

Fig. 3.20 shows a strong symmetric visibility assignment. Note that the shown visibility assignment is neither non-null uniformly coherent ($\mathcal{V}(X_1, X_3) \cap H[X_1, X_2]$ is not null, but $\mathcal{V}(X_1, X_3) \cap H[X_1, X_2] \neq \mathcal{V}(X_1, X_2)$) nor non-null uniformly correlated ($\mathcal{V}(X_1, X_3) \cap H[X_2, X_3]$ is not null, but $\mathcal{V}(X_1, X_3) \cap H[X_2, X_3] \neq \mathcal{V}(X_2, X_3)$). Basically, under uniformly coherent visibility assignment, the single graph representing $\mathcal{V}_X$ is implicitly a directed tree rooted at X, with edges directed towards X. Similarly, for a node Y different from X, $\mathcal{V}_Y$ is a directed tree rooted at Y. When we try to represent both $\mathcal{V}_X$ and $\mathcal{V}_Y$ in a single graph, the edges in the path from X to Y have to be

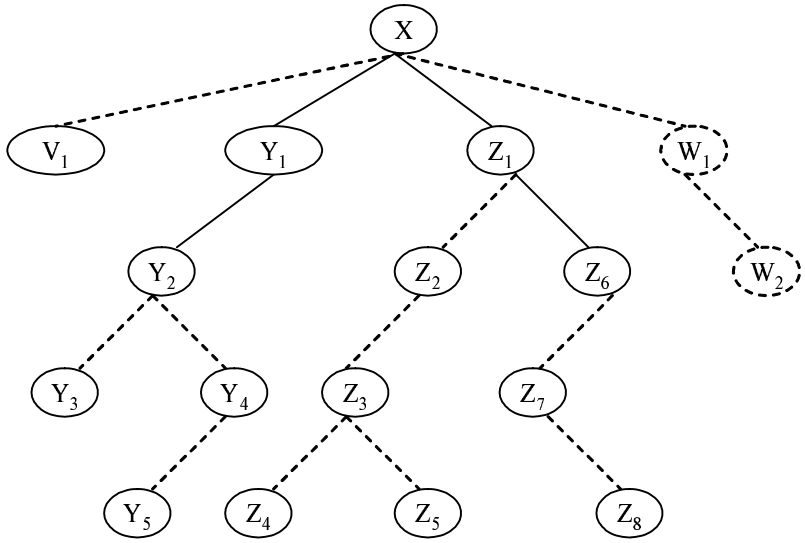

FIGURE 3.18 – Uniformly correlated visibility.

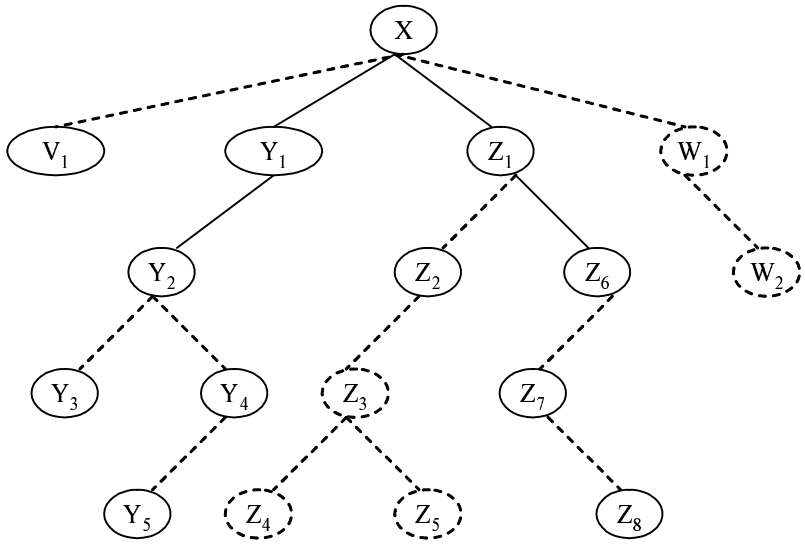

FIGURE 3.19 – Non-null uniformly correlated visibility.

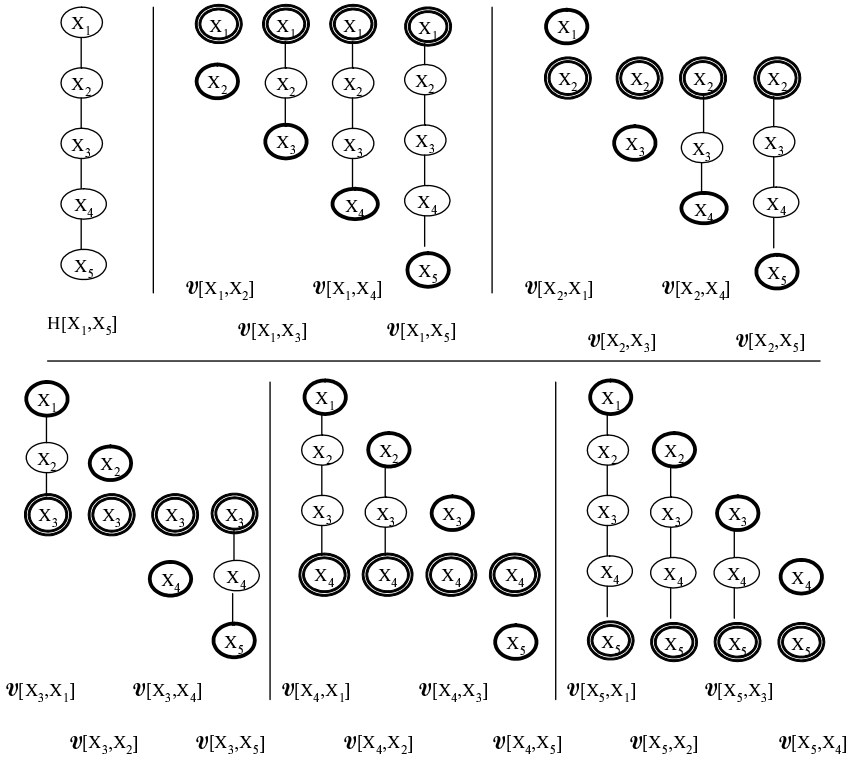

FIGURE 3.20 – Strong symmetric visibility assignment.

represented in both directions. The sv-symmetricity property enables representing the edges in undirected form, in a single graph representation.

We are now in a position to give the conditions required for single graph visibility representation of a hierarchy.

Definition 3.14 (Harmonious Visibility and Noticeability) A visibility assignment $\mathcal{V}$ is *harmonious* if it satisfies the following properties :

- non-null uniform coherence ;
- non-null uniform correlation ;
- sv-symmetricity ; and
- if a node is visible to some node in H, then it is visible to every node in H.

A noticeability assignment $\mathcal{N}$ is *harmonious* if its corresponding visibility assignment is harmonious.

It turns out that *for a harmonious visibility assignment, the visibility graph $\mathcal{V}_X$, for any node X, is the visibility graph for all nodes in H.*

Theorem 3.1 For a visibility assignment $\mathcal{V}$, $\mathcal{V}_X$ for any node X, represents $\mathcal{V}$ in a "lossless" fashion (that is, $\mathcal{V}_X[Y,Z] = \mathcal{V}[Y,Z]$ for any nodes Y and Z, where $\mathcal{V}_X[Y,Z] = \mathcal{V}_X \cap H[Y,Z]$) iff the visibility assignment is harmonious.

Proof First, we show the sufficiency.

If $\mathcal{V}[Y,Z]$ is null, then $\mathcal{V}[X,Z]$ is also null, by the last property in the statement of the proposition. Therefore, Z will not be in $\mathcal{V}_X$. In the following, we consider the case where $\mathcal{V}[Y,Z]$ is non-null. Let $\mathcal{V}_X[Y,Z]$ be $\langle Z_1, Z\rangle$, where Z_1 is a node in the path from Y to Z. Note that Z_1 could be Y, Z, or some other node. We will show that $\mathcal{V}[Y,Z]$ is also $\langle Z_1, Z\rangle$.

We consider the various possibilities of the relative positions of X, Y and Z in H, illustrated in Fig. 3.21.

(a) X is in an extension of the path $\langle Z, Y\rangle$.

Here, $\mathcal{V}[X,Z] \cap H[Y,Z]$ is also $\langle Z_1, Z\rangle$ and this equals $\mathcal{V}[Y,Z]$, by non-null uniform correlation.

(b) X is in the extension of the path $\langle Y, Z\rangle$.

Here, $\mathcal{V}[X,Z_1]$ contains $\langle Z, Z_1\rangle$. By non-null uniform correlation, $\mathcal{V}[X,Z_1] \cap H[Z,Z_1]$ equals $\mathcal{V}[Z,Z_1]$. By sv-symmetry, this equals $\mathcal{V}[Z_1,Z]$. Now, by non-null uniform correlation, $\mathcal{V}[Y,Z] \cap H[Z_1,Z]$ equals $\mathcal{V}[Z_1,Z]$. Therefore, $\mathcal{V}[Y,Z]$ includes $\langle Z_1, Z\rangle$. We show that it indeed equals $\langle Z_1, Z\rangle$. Suppose on the contrary that $\mathcal{V}[Y,Z]$ has Z_2 (preceding Z_1) and the edge $\langle Z_2, Z_1\rangle$, that is, it includes $\langle Z_2, Z\rangle$. Then, by uniform correlation $\mathcal{V}[Z_2,Z]$ must be $\langle Z_2, Z\rangle$; by sv-symmetry, $\mathcal{V}[Z,Z_2]$ must be $\langle Z, Z_2\rangle$; by uniform correlation, $\mathcal{V}[X,Z_2] \cap H[Z,Z_2]$ must be $\langle Z, Z_2\rangle$; and so $\mathcal{V}[X,Z_2]$ must include $\langle Z_2, Z\rangle$, and hence the edge $\langle Z_2, Z_1\rangle$. Then, $\mathcal{V}_X[Y,Z]$ will also have $\langle Z_2, Z_1\rangle$, contradicting the assumption otherwise.

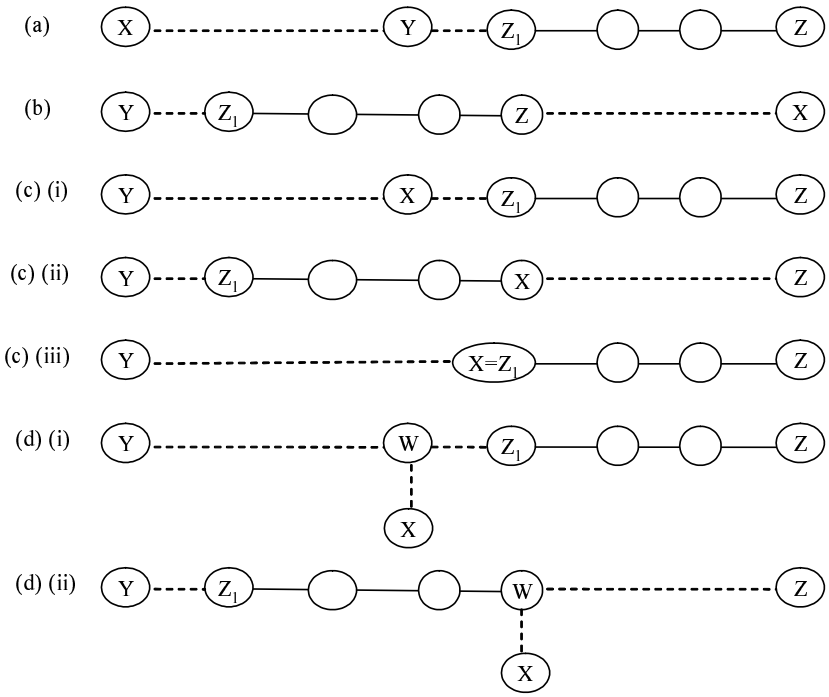

FIGURE 3.21 – Scenarios for the proof of Theorem 3.1.

(c) X is in the path $\langle Y, Z\rangle$.

Three subcases arise :

(i) Z_1 is in the path $\langle X, Z\rangle$ and is different from X. Then, by non-null uniform correlation, $\mathcal{V}[Y, Z] \cap H[X, Z]$ equals $\mathcal{V}[X, Z]$ which is $\langle Z_1, Z\rangle$. Therefore, $\mathcal{V}[Y, Z]$ is $\langle Z_1, Z\rangle$.

(ii) Z_1 is in the path $\langle Y, X\rangle$ and is different from X. Here, $\mathcal{V}[X, Z_1]$ must be $\langle X, Z_1\rangle$. By sv-symmetry, $\mathcal{V}[Z_1, X]$ is $\langle Z_1, X\rangle$. Also, $\mathcal{V}[Y, Z] \cap H[X, Z]$ equals $\langle X, Z\rangle$, and therefore $\mathcal{V}[Z_1, Z]$ is $\langle Z_1, Z\rangle$. By non-null uniform correlation, $\mathcal{V}[Y, Z] \cap H[Z_1, Z]$ is $\mathcal{V}[Z_1, Z]$. Thus, $\mathcal{V}[Y, Z]$ includes $\langle Z_1, Z\rangle$. Suppose $\mathcal{V}[Y, Z]$ includes another edge $\langle Z_2, Z_1\rangle$, for Z_2 preceding Z_1, as in the previous case. Then, by non-null uniform correlation, $\mathcal{V}[Y, Z] \cap H[Z_2, Z]$ is $\langle Z_2, Z\rangle$; by non-null uniform coherence, $\mathcal{V}[Z_2, Z] \cap H[Z_2, X]$ is $\langle Z_2, X\rangle$; by sv-symmetry, $\mathcal{V}[X, Z_2]$ is $\langle X, Z_2\rangle$. Therefore, the edge $\langle Z_2, Z_1\rangle$ must also be in $\mathcal{V}_X$, in contradiction to the assumption otherwise.

(iii) Z_1 is X. Here also, $\mathcal{V}[Y, Z] \cap H[Z_1, Z]$ is $\mathcal{V}[Z_1, Z]$. Thus, $\mathcal{V}[Y, Z]$ includes $\langle Z_1, Z\rangle$. The proof that $\mathcal{V}[Y, Z]$ does not contain additional edges follows as in subcase (ii) above.

(d) X is neither in $\langle Y, Z\rangle$ nor in any of its extensions.

Then, let W be the node in $\langle Y, Z\rangle$ common to the paths $\langle X, Y\rangle$ and $\langle X, Z\rangle$. Two subcases arise.

(i) Z_1 is in the path $\langle W, Z\rangle$, and is different from W. Then, $\mathcal{V}[X, Z]$ must be $\langle Z_1, Z\rangle$. By non-null uniform correlation, $\mathcal{V}[X, Z] \cap H[W, Z]$ is $\mathcal{V}[W, Z]$ and is equal to $\langle Z_1, Z\rangle$; by non-null uniform correlation, $\mathcal{V}[Y, Z] \cap H[W, Z]$ is $\langle Z_1, Z\rangle$. Therefore, $\mathcal{V}[Y, Z]$ is $\langle Z_1, Z\rangle$.

(ii) Z_1 is W or is in the path $\langle Y, W\rangle$. In this case, by non-null uniform coherence, $\mathcal{V}[Y, Z] \cap H[Y, W]$ is $\mathcal{V}[Y, W]$. Therefore, we need to show only that $\mathcal{V}[Y, W]$ is $\langle Z_1, W\rangle$. This follows from case (b), by substituting W in place of Z.

Next, we show the necessity of all four properties of a harmonious visibility assignment, for single graph representation. Necessity of :

- non-null uniform coherence : Fig. 3.22(a) satisfies all, but the non-null uniform coherence property required for a harmonious assignment (as $\mathcal{V}[X, Z] \cap H[X, Y]$ is non-null, but not equal to $\mathcal{V}[X, Y]$). Then, $\mathcal{V}_Y[X, Z] \neq \mathcal{V}[X, Z]$.
- non-null uniform correlation : Fig. 3.22(b) satisfies all, but the non-null uniform correlation property (as $\mathcal{V}[X, Z] \cap H[Y, Z]$ is non-null, but not equal to $\mathcal{V}[Y, Z]$). Then, $\mathcal{V}_X[Y, Z] \neq \mathcal{V}[Y, Z]$.
- sv-symmetricity : Fig. 3.22(c) satisfies all, but the sv-symmetricity property (as $\mathcal{V}[Y, X]$ is strong, but $\mathcal{V}[X, Y]$ is not strong). Then, $\mathcal{V}_Y[X, Z] \neq \mathcal{V}[X, Z]$.
- if a node is visible to some node in H, then it is visible to every node in H : Fig. 3.22(d) satisfies all, but the fourth property required for a harmonious assignment (as Z is visible to Y, but not to X). Then, $\mathcal{V}_Y[X, Z] \neq \mathcal{V}[X, Z]$.

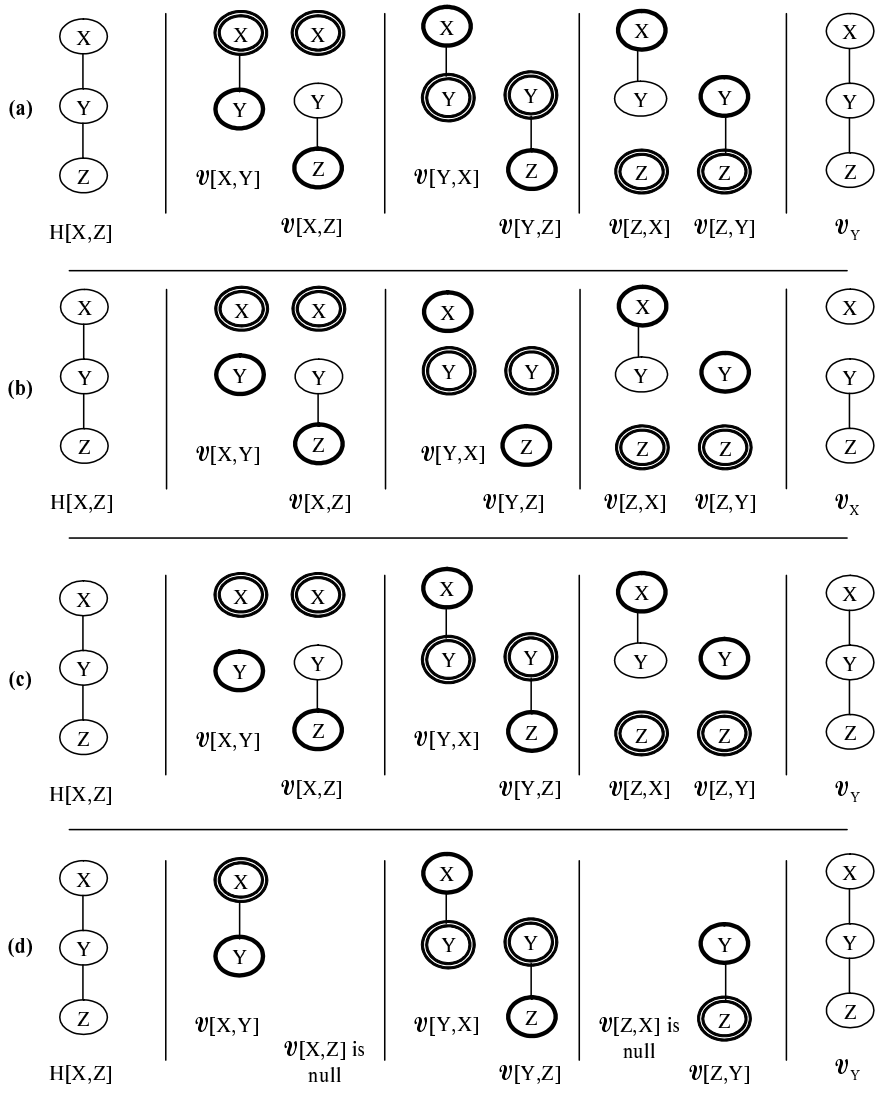

FIGURE 3.22 – Necessity of a harmonious visibility assignment for single graph representation.

□

The above proposition asserts that when the visibility assignment is harmonious, a single graph can represent the visibility graph of every node, including those which are not visible to any node at all. The latter part follows from the fact that the proof of the proposition did not make use of whether X itself is visible (to every node) or not, and similarly with the visibility of Y.

3.5 Discussion and Related Works

The notion of SoV is based on the concept of Spheres of Control (SoC) introduced by [Dav78]. A Sphere of Control encapsulates entities sharing a similar set of properties or having a dependency relation. The dependency relations considered in [Dav78] are atomicity, commitment, resource allocation, recovery, auditing, consistency, etc. SoV logically group the nodes (and their attributes) visible to another node in a hierarchy. Also, Davies [Dav78] considers homogeneous and non-autonomous systems where visibility is not an issue. Our work can be considered as complementary to the work in [Dav78] to heterogeneous and autonomous systems.

Later works have extended the initial concept of SoC to Spheres of Atomicity [AH00] and Commitment [YS01]. [AH00] utilize the properties of the processes (pivot, compensable and retriable) in a Sphere of Atomicity to determine if the sphere, as a whole, guarantees atomicity. [YS01] apply the concept of SoC to Multi-Agent Systems (MAS) to structure agents based on their commitment guarantees. These works are not directly related to the work presented in this thesis. We mention them for the sake of completeness. Some other works which have touched upon the visibility aspect : [Mof98] identifies real-life scenarios where there might be a need to deviate from the inheritance of access rights upwards through the hierarchy in a role-based access control. [Fie05] considers visibility with respect to sending publish/subscribe notifications for event based systems. However, none of them consider the effect of *relationships* which might exist among the visibilities of different entities of a system.

From a security standpoint, SoV seems close to the Role Based Access Control (RBAC) mechanism [Mof98]. In RBAC, the possible roles of an organization are categorized into a hierarchy, where each role has certain permissions/restrictions with respect to accessing resources (objects) of the organization. Each employee (subject) of the organization is assigned a role. Thus, whether a subject can access an object depends on the subject's role's accessibility over the object. Also, accessibility relations in RBAC are inherited, that is, parents inherit the permissions of their children while children inherit the restrictions of their parents. Deviations to inheritance can be modeled as exceptions in RBAC. In a SoV context, inheritance corresponds to "correlated" visibility, and we also consider other accessibility relations, e.g., coherence. And, "exceptions" are accommodated as first class citizens in the SoV formalism.

The other commonly used access control mechanism is Discretionary Access Control (DAC) [RS00]. In DAC, a system consists of subjects and objects. Subjects can only

access those objects over which they have access rights. By default, subjects have full access rights over the objects they create ; they are owners of those objects. The owners can then give (full or partial) access rights over their objects to other subjects. Once a subject has received access rights over an object, the question of whether it can propagate it further to other subjects arises. In practice, access control mechanisms are usually implemented using a pair of operations such as SQL Grant/Revoke which facilitate both giving access rights on objects to subjects and allowing the subjects to propagate the access rights to other subjects. In our formalism, access rights, subjects and objects correspond to visibility, nodes and their attributes, respectively. Referring to Fig. 3.3 again, suppose $\mathcal{V}[X, Z]$ is $\langle Y, Z\rangle$. This can be interpreted to imply that

1. Y has access rights over Z's objects,
2. Z has permitted Y to propagate those rights to X, and
3. Y has no objections in doing so.

Then, if $\mathcal{V}[W, Z]$ does not have Y, it could be due to any of the above properties (with W in place of X) not being true. With SQL Grant/Revoke operators, only (1) can be controlled. (2) cannot be controlled as SQL operators do not provide the necessary fine grained control where a subject is allowed to propagate a received access right to some, but not all. The extensions proposed in [RS00] allow SQL operators to control (2), but still not (3). There are no provisions for a subject s_3 to control the giving/propagating of access rights to s_2 by another subject s_1. Of course, we can impose such restrictions in our formalism. Also, in the above case, where $\mathcal{V}[X, Z]$ is $\langle Y, Z\rangle$, not only Y but also every node in the path $\langle Y, Z\rangle$ satisfies all the above three properties. This restriction cannot be imposed with SQL operators. There, the properties of the nodes are not interdependent. Similarly, revocation of access rights can also be done in a fine-grained way with our formalism but in a hierarchical setup. An additional benefit with our formalism is keeping track of access privileges and propagation rights in a distributed fashion. For example, in each node, the access rights over the objects created in that node can be recorded and updated in its SoN.

Other related scenarios include privacy preserving applications such as DARPA's Total Information Awareness (TIA) [TIA] or Microsoft's MyLifeBits Project [Lif]. The objective of both, and similar, applications are to build haystacks of electronic information about people. The needle in such haystacks corresponds to information about a particular person, e.g., that of a terrorist in the case of TIA. Here, the challenge is to return information only specific to the queried person. It should not be possible to derive information about others (to protect their privacy) from the returned information. Thus, there is a need to analyze the sphere of noticeability (of a piece of information) of a person over others.

From a programming language point of view, Aspect Oriented Programming (AOP) deserves special mention here. "Separation of Concerns" is an important characteristic of AOP, and according to Filman and Friedman [FF01], AOP can be differentiated from other programming languages based on the following two parameters : *quantification*

and *obliviousness*. Roughly, quantification corresponds to the restrictions a class A imposes on another dependent/related class B. Obliviousness corresponds to the degree of independence with which (the dependent/related) class B can be programmed (oblivious of the restrictions imposed upon it by A). AOP justifies two basic characteristics of our model : (i) nodes other than X and Y can have a say in whether X is visible to Y and (ii) asymmetric visibility (X is visible to Y does not imply that Y is also visible to X), or in other words, why we study SoV and SoN separately as two complementary notions. However, there are some underlying differences as well. While AOP relies on an "intelligent" run-time environment to resolve any visibility issues (including conflicts) at run-time, we consider a more static scenario where all visibility requirements/conflicts have been resolved (discussed in Section 3.4), and the main challenge is to model the assigned visibility/noticeability.

Part II

Minimal Visibility for Transactional Hierarchical Services

Introduction

In the previous part, we discussed how to model visibility. We assumed that the visibility requirements and restrictions have already been negotiated, and we were only interested in capturing and representing visibility. In this part, we consider the orthogonal problem of determining the visibility requirements in a restricted visibility environment. To be more precise, we discuss the minimal visibility required for a hierarchical Web services composition, such that atomicity (see Section 1.6) is guaranteed for any execution. The visibility attribute here corresponds to the execution logs maintained locally by each service in the hierarchy. We discuss two types of visibility requirements in this part : First, we discuss the visibility required over logs of services from a global perspective, that is, the visibility required by say a global co-ordinator over local service logs to provide atomicity. Later, we discuss a more secure visibility requirement, where it is sufficient for a pair of parent-child services to have visibility over some attributes of each other. The work is thus also relevant towards the long term objective of providing a transactional framework for Web services [Bis04, BK08, Bis08, VV04, WST].

Let us consider the hierarchical composition scenario as shown in Fig. 3.23. The execution proceeds in a hierarchical fashion downwards, starting from the root, subsequent parents invoking their children, and so on, until all leaves have finished execution successfully. Note that any functionality is provided by the leaf services, all others are virtual. We assume that all leaf services have corresponding compensating services which can be invoked to annul their effects. Given this, in the event of a service failure at any stage of the execution, atomicity is ensured by compensation, that is, by invoking the compensating services of all executed leaf services till then, in their *reverse order*. As the execution order is important here, we need a more descriptive representation of service compositions, one which reflects any control flow dependencies between service invocations. Thus, we switch to a behavioral description, where a composite service is represented as a FSM and a hierarchical composition as a hierarchical or distributed FSM (depending on the type of composition : top-down or bottom-up). For example, the hierarchical FSM M in Fig. 3.24 could be the corresponding behavioral description of the hierarchical composition in Fig. 3.23. The hierarchical FSM in Fig. 3.24 is a simple extension of the workflow logic in Fig. 1.11 (levels 2 and 3 correspond to the Travel Funds and Deliver Cheque services respectively). With such a representation, each transition corresponds to a service invocation (indicated by the letters in square brackets in Fig. 3.24). In reality, some may correspond to local actions as well. However, to maintain uniformity, we assume that each transition corresponds to a service invocation with local actions corresponding to local service invocations. Note that in contrast to the tree representation of a hierarchical composition in Chapter 3, here all children services of a node at any level may not get invoked (choice), and even if they do, they may not be invoked concurrently, but in the order given by its corresponding behavioral description FSM.

With this infrastructure, if a service S fails, the failure is detected by the composite service which had invoked S, and propagated to the root service or global co-ordinator responsible for failure recovery. As mentioned earlier, failure recovery is via compensa-

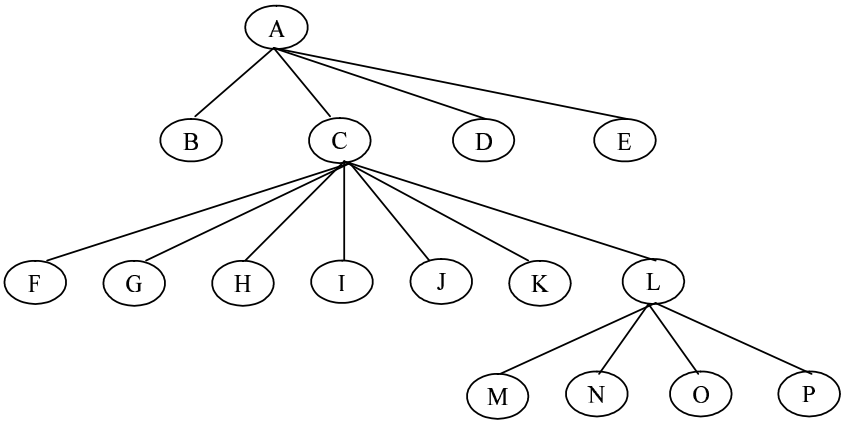

FIGURE 3.23 – Sample hierarchical Web services composition.

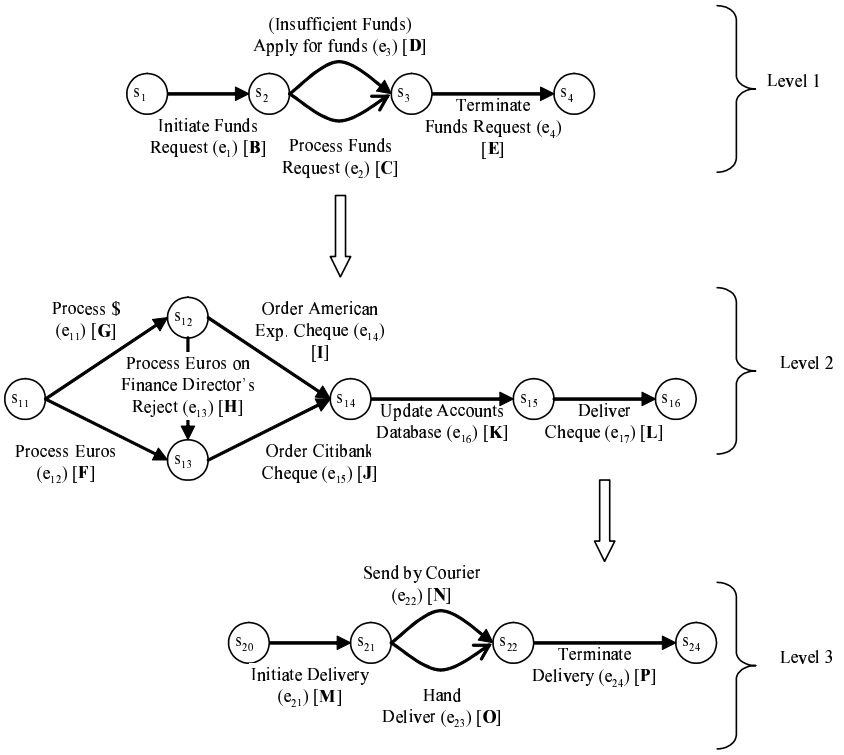

FIGURE 3.24 – Hierarchical FSM corresponding to the composition in Fig. 3.23.

tion, that is, invoking the compensating services of all executed leaf services in reverse order. With reference to Fig. 3.24, assuming service N failed and the path $GHJK$ was executed at level 2, then the compensating services of the following services need to be invoked in the order $MKJHG$. Thus, to be able to perform compensation, the global co-ordinator needs to be able to reconstruct the complete execution sequence (of executed leaf services). The usual way of tracking execution is of course to log details of executed services. However, logging is clearly expensive in terms of both space and time. Also, services may be provided by different providers maintaining logs in different formats, rendering some of them incomprehensible. Finally, we are in a limited trust environment, where we start with zero visibility and only assign as much visibility as is absolutely required. Thus, we would like to determine the minimal set of leaf services whose execution logs need to be visible, to be able to perform compensation (guarantee atomicity) in the event of a failure. For example, again with reference to Fig. 3.24, a minimal set of leaf services which need to be visible for compensability is $\{G, J, O\}$. We formalize why the above set is sufficient and minimal for compensability in the next section. The visibility discussion till now can be considered as global, where the minimal set of services $\{G, J, O\}$ once determined, are visible to all others in the hierarchy. We discuss a more secure decentralized mechanism in Section 4.1.2, where it is sufficient for parent and child services to have partial visibility over some attributes of each other (visibility over service descriptions or logs is not needed).

From an implementation perspective, the problem we are trying to solve is a middleware problem, which would be relevant to say a BPEL execution engine. In BPEL, as mentioned earlier, each invoke operation can be compensated. In practice, each invoke operation implicitly creates a scope and compensating operations are associated with the scope. Taking a particular BPEL execution engine implementation, namely ActiveBpel [ABp], a scope is implemented by the class *AeActivityScopeImpl* and the information which might be needed to compensate it is stored in a class variable *mCompInfo* of type *AeCompInfo*. On successful compensation of a scope, its compensation information *mCompInfo* is passed to its parent. An *AeCompInfo* object basically contains (i) a reference to the scope that completed, and (ii) a variable snapshot object to record the state of all of its variables. In our work, we differentiate between the two parts explicitly. We are mainly interested in the first part, referred to as the execution log of an invocation, as visibility over it is needed to determine the execution sequence (similar to functionality of the method *getMatchingScopes()* of class *AeActivityCompensateImpl* which determines the list of scopes to be compensated) in the event of a failure. Of course, in case of ActiveBpel, all executed scope references are visible, and the functionality of *getMatchingScopes()* is simply to traverse the hierarchy of scope references to generate the list of executed scopes. In our case, the list of executed scopes would need to be reconstructed from the visible scope references. And, the problem is then to determine a minimal set of scopes whose references need to be visible, based on which we can always determine the complete list of executed scopes. Finally, the second part of an *AeCompInfo* object (a snapshot record of the scope variables), referred to as the data log, is also important for compensation. However, we do not deal with this part directly in our work. We assume that any such required data is logged properly, is visible,

and can be used by the compensating services as and when required.

Formalizing Compensability

We defined a composite service as a FSM $M = \{Q, s_0, s_f, \mathcal{T}\}$. Under restricted visibility, M may not have visibility over (the execution logs of) all its transitions (invocations). Thus, we further need to consider the subset of transitions $\mathcal{T}_O \subseteq \mathcal{T}$ visible to M. For an execution sequence ρ of M, we call visibility projection the execution visibility we have after ρ was executed. We say that a visibility projection σ is uncertain if there exists two paths having the same projection. The service M is execution sequence detectable iff none of its visibility projections are uncertain.

Definition 3.15 *For a service $M = \{Q, s_0, s_f, \mathcal{T}\}$, let $\mathcal{T}_O \subseteq \mathcal{T}$ be the set of visible transitions. The visibility projection $Obs_O : \mathcal{T}^* \longrightarrow \mathcal{T}_O^*$ is the morphism with $Obs_O(a_1 \dots a_n) = o_1 \cdots o_n$ with $o_i = a_i$ if $a_i \in \mathcal{T}_O$, and $o_i = \epsilon$ if $a_i \in \mathcal{T} \backslash \mathcal{T}_\mathcal{O}$, with ϵ the empty word.*

That is, $\text{Obs}_O(\rho)$ is the subsequence of ρ obtained by eliminating from ρ every occurrence of a transition which is not in $\mathcal{T}_\mathcal{O}$. With such a visibility projection Obs_O, the only way of having compensability is to have every transition visible. Indeed, as soon as there exists even one invisible transition, the service is non-compensable. Else, let us take a path $\rho\tau$ with the last transition $\tau \notin \mathcal{T}_\mathcal{O}$. Then, $\text{Obs}_O(\rho\tau) = \text{Obs}_O(\rho)$. A usual way to overcome such a problem is to ask for certainty only up to the last few transitions of the sequence [OW90]. However, this workaround does not make sense in our framework since if we cannot compensate the very last transition, then there is no point in compensating any transition at all. As such, we design a new visibility mechanism, where the last state reached before failure is monitored, even if the last transition is not logged. In practice, it means that every state that is reached is logged, and overwrite the previous state in a special memory buffer.

Definition 3.16 (Visibility Projection) *Let $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a service, $\mathcal{T}_O \subseteq \mathcal{T}$. The visibility projection $Obs_O^{last} : \mathcal{T}^* \longrightarrow (\mathcal{T}_O^*, Q)$ is the function $Obs_O^{last}(\rho) = (Obs_O(\rho), q)$ for all $\rho \in \mathcal{P}(M)$ ending in q.*

Definition 3.17 (Compensability) *Given a service $M = (Q, s_0, s_f, \mathcal{T})$, we call $\mathcal{T}_O \subseteq \mathcal{T}$ a compensable set of transitions if the service is execution sequence detectable with Obs_O^{last}.*

We will stick with this definition of compensability for the rest of this work. As mentioned before, we are interested in minimal visibility, that is, visibility over as few transitions as possible.

Problem statement. Given a service $M = (Q, s_0, s_f, \mathcal{T})$, we would like to determine a minimal set of transitions $\mathcal{T}_O \subseteq \mathcal{T}$ which need to be visible such that the system is still compensable.

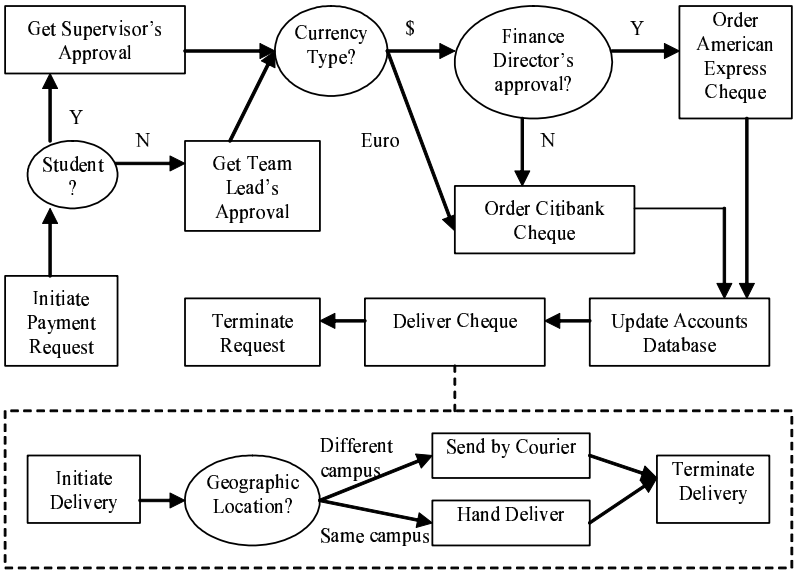

FIGURE 3.25 – Hierarchical Travel Funds service.

The cardinality of such a minimal compensable set $\mathcal{T}_\mathcal{O}$ of a service M is referred to as its compensable size $MO(M) = |\mathcal{T}_\mathcal{O}|$. Note that as is usual with decision and computation algorithms, given a service, it is sufficient to have an algorithm which gives its compensable size. That is, we can derive in polynomial time a minimal compensable set of the service based on an oracle algorithm given the compensable size.

For example, let us consider the hierarchical Travel Funds workflow and its FSM representation in Figures 1.11 and 1.12 (reproduced here in Figures 3.25 and 3.26) respectively. Note that the FSM representation is a simplification since for instance the choice between the Team Leader or Supervisor approvals is not represented. The reason is that they are both associated with an empty compensating service, hence knowing which was chosen is not necessary to be able to perform compensation. However, it is necessary to know which bank issued the cheque in order to be able to compensate it, by a "Cancel American Express (Citibank) Cheque". As mentioned earlier, we assume that each service logs any data which might be required for its compensation (e.g., cheque amount and account number), and that its compensating service has visibility over any such data logs. Now, let $\mathcal{T}_O = \{e_2, e_3, e_9\}$ and a failure occurs while processing e_8, that is, the cheque is not issued or delivered correctly. Then, $\mathrm{Obs}_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) = \mathrm{Obs}_O^{last}(e_1e_2e_4e_6e_7)$. Thus, we do not know if an American Express or Citibank cheque was processed. With $\mathcal{T}'_O = \{e_2, e_6, e_9\}$, we have $\mathrm{Obs}_O^{last}(e_1e_2e_5e_7) = (e_2, s_5) \neq \mathrm{Obs}_O^{last}(e_1e_2e_4e_6e_7) = (e_2e_6, s_5) \neq \mathrm{Obs}_O^{last}(e_1e_2e_4e_6) = (e_2e_6, s_4)$, and $\mathcal{T}'_\mathcal{O}$ is a compensable set of transitions.

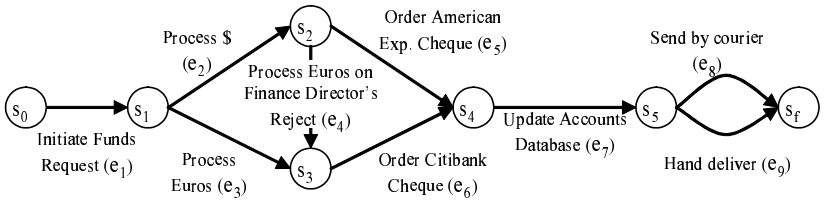

FIGURE 3.26 – Hierarchical FSM representation of the Travel Funds service in Fig. 3.25.

In the next section, we discuss the complexity of our problem.

Problem Hardness

We first relate the problem of computing $MO(M)$ using our definition of visibility projections with other known problems. We state now that computing a minimal compensable set is equivalent to the *uniconnected subgraph problem* [GJ79], also called the *minimal marker placement problem* [Mah76], in the meaning of the following proposition.

Proposition 3.6 *Let $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a service and $\mathcal{T}_\mathcal{C}$ a subset of transitions of M. Denote by $M' = \{Q, s_0, s_f, \mathcal{T} \backslash \mathcal{T}_O\}$ the service M obtained by deleting all transitions belonging to $\mathcal{T}_\mathcal{C}$. Then, $\mathcal{T}_\mathcal{C}$ is a compensable set M iff there does not exist a pair of states $q_1 \neq q_2$ in M' with more than one path between them.*

Proof First, we show that if there does not exist a pair of states $q_1 \neq q_2$ in M' with more than one path between them, then from any visibility projection (σ, q_{n+1}), we can reconstruct in a unique way the path ρ of M with $\text{Obs}_O^{last}(\rho) = (\sigma, q_{n+1})$ using the following algorithm :

Algorithm to reconstruct the execution sequence from a given visibility projection.
Input. FSM $M = \{Q, s_0, s_f, \mathcal{T}\}$, visible subset $\mathcal{T}_O \subseteq \mathcal{T}$, FSM $M' = \{Q, s_0, s_f, \mathcal{T} \backslash \mathcal{T}_O\}$ and visibility projection (σ, q_{n+1}).
Output. The unique path ρ of M with $\text{Obs}_O^{last}(\rho) = (\sigma, q_{n+1})$.
Initialization. Set $\rho := \epsilon$, current state $s := s_0$ and we use index i to iterate the projection $\sigma = \tau_1 \tau_2 \cdots \tau_n$.
if $n = 0$ (that is, no transitions were logged), then set ρ to the unique path connecting s to q_{n+1} and return.
else
begin
for $i = 1 \cdots n$ do
if τ_i is an outgoing transition of s, then append τ_i to ρ.

else determine the unique path ρ_1 of M' connecting s to ${}^*\tau_i$, and append $\rho_1\tau_i$ to ρ. (The unique path connecting s to ${}^*\tau_i$ can be determined in linear time.)
endif
Set $s := \tau_i^*$.
endfor
if $s = \tau_n^* = q_{n+1}$, then return ρ
else determine the unique path ρ_1 connecting s to q_{n+1}, append ρ_1 to ρ, and then return ρ.
endif

The converse is trivial. Clearly, if there exists more than one path between two states $q_1 \neq q_2$ of M', then we can find more than one execution sequence corresponding to a visibility projection which passes through q_1 and q_2. □

The fact is that the marker placement problem is an NP-complete problem. The question is then to know if there exists a structural subclass of graphs which has a tractable algorithm to give the minimal compensable size. We know from [Mah76] that the minimal marker placement problem is NP-complete even for acyclic graphs. However, the proof uses a graph with unbounded (in and out) degree. We show that the problem is NP-complete even if the graph is both acyclic and the sum of its in and outdegree bounded by 3 (that is, indegree 2 and outdegree 1, or vice versa). The core of the proof follows the same strategy as [Mah76], but the encoding to get a unique starting and ending point is both easier to understand and allows a lower in and outdegree.

Theorem 3.2 *Let $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a service, and k a number. Knowing whether $MO(M) \leq k$ is NP-complete, even if the corresponding graph is acyclic and the sum of in and outdegree of every node bounded by 3.*

Proof Let $M = \{Q, s_0, s_f, \mathcal{T}\}$ be a system. We reduce Vertex Cover [GJ79] to the problem of finding a subset of transitions $\mathcal{T}_\mathcal{O}$ of M, such that there are no two paths $\rho_1 \neq \rho_2$ beginning and ending at the same pair of states, and not using any transitions of $\mathcal{T}_\mathcal{O}$.

Let us take an *undirected* graph (V, E) and a number k. We would like to know whether there exists a subset V_O of V of size $\leq k$, such that for all $(v, w) \in E$, at least one of v, w belongs to V_O. This problem is NP-complete even with (V, E) of degree 3. The first FSM M we build has a state space $S = V_1 \cup V_2 \cup E_1 \cup E_2$ where $V_i = \{v_i \mid v \in V\}$ and $E_i = \{e_i \mid e \in E\}$. Furthermore, for $v, w \in V$ and $e \in E$, we have transitions :

1. $(e_1, v_1) \in \mathcal{T}$ iff $v \in e$ iff $(v_2, e_2) \in \mathcal{T}$
2. $(v_1, w_2) \in \mathcal{T}$ iff $v = w$.

A graphical representation of M appears in Fig. 3.27. Assume that there is a subset V_O of V of size k, such that for all $(v, w) \in E$, at least one of v, w belongs to V_O. Then,

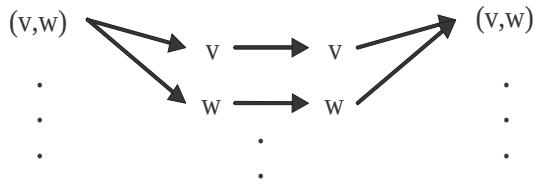

FIGURE 3.27 – Illustration for the proof of Theorem 3.2.

defining $\mathcal{T}_O = \{(v_1, v_2) \mid v \in V_O\}$, we have that there are no two paths $\rho_1 \neq \rho_2$ with ρ_1 and ρ_2 beginning and ending at the same pair of nodes, and not using transitions of $\mathcal{T}_\mathcal{O}$. By contradiction, else we would have ρ_1, ρ_2 both from some $e_1 \in E_1$ to some $f_2 \in E_2$, and not using transitions of $\mathcal{T}_\mathcal{O}$. By definition of $\mathcal{T}_\mathcal{O}$, it means that for ρ_1, there exists a node $v \in e$, $v \in f$, such that $v \notin V_O$. Similarly, for ρ_2 with a node w. Since $\rho_1 \neq \rho_2$, we have that $v \neq w$, hence $e = (v, w)$ contradicts V_O is a vertex cover.

Conversely, assume that there is a set of transitions $\mathcal{T}_\mathcal{O}$ of size k, such that there do not exist two distinct paths between a pair of nodes without using $\mathcal{T}_\mathcal{O}$. We build the set of nodes $V_O = \{v \mid (v_1, v_2) \in \mathcal{T}_\mathcal{O}\} \cup \{v \mid \exists e, (e_1, v_1) \in \mathcal{T}_\mathcal{O}\} \cup \{v \mid \exists e, (v_2, e_2) \in \mathcal{T}_\mathcal{O}\}$. Clearly, $|V_O| \leq |\mathcal{T}_O| = k$. We prove now that V is a vertex cover of (V, E). Assume by contradiction that there exists an edge $e = (v, w)$, such that $v, w \notin V_O$. Then, we argue that $e_1 v_1 v_2 e_2$ and $e_1 w_1 w_2 e_2$ are two paths not using $\mathcal{T}_\mathcal{O}$, a contradiction.

However, so far, the graph defined is not a system since it has several nodes with indegree 0 (the $(e_1)_{e \in E}$), and several nodes with outdegree 0 (the $(e_2)_{e \in E}$). Moreover, the indegree of nodes $(v_1)_{v \in V}$ and the outdegree of nodes $(v_2)_{v \in V}$ can be 3 (the degree of the undirected graph (V, E)). However, it is acyclic. For the degree, one can safely transform any node v_1 with 3 incoming transitions from nodes e_1, f_1, g_1 by having two nodes v_1, v_1' with transitions $(e_1, v_1'), (f_1, v_1'), (v_1', v_1)$ and (g_1, v_1). Hence, all nodes have indegree at most 2. The same can be done for outdegree. The size of the minimal compensable set of transitions will not change with such a transformation. Actually, with such a technique, we could start from an undirected graph of any degree.

Making the graph a system is a little more involved. We use the graph G from Fig. 3.26. It then suffices to create a balanced binary tree of transitions with root s_i, such that there are $|E|$ leaves. This tree has $O(2|E|)$ nodes, that we add to the system S we built from (V, E). The root of the tree is the unique initial node, and every leaf is connected to a node $(e_1)_{e \in E}$ through a copy of graph G. The same is done for nodes $(e_2)_{e \in E}$ connected through copies of G to a balanced binary tree with root s_f (the unique final node). This system has $O(|V| + |E|)$ nodes, is acyclic and of total degree 3. Now, it is easy to show that if the minimal vertex cover has k vertices, then the minimal compensable set of transitions is of size $k + 4|E|$. Indeed, there are $2|E|$ copies of the graph G each of which requires 2 visible transitions. Once these transitions have been deleted, the two balanced trees are totally disconnected from each other and from the first system we had built (since every path from the initial to the final node of the

graph G uses one of the two visible transitions), and hence we need exactly k more transitions to be visible. Note that connecting directly the tree with S without using G would not work since it would potentially connect s^0, s^f through two different paths $s^0 \longrightarrow e_1 \longrightarrow e_2 \longrightarrow s^f$ and $s^0 \longrightarrow f_1 \longrightarrow f_2 \longrightarrow s^f$, with $e, f \in E$. □

This theorem does not mean that the problem is impossible to solve, but that it cannot be solved for all possible services. For instance, the complexity of the brute force method which generates every subset of transitions and tests whether it is compensable, is $O(2^{|M|})$ for a service of size $|M|$ transitions. There are usually two ways to approach an NP-complete problem : (i) Determine structural properties which make the problem easier to solve and often hold for (real-life) services. We propose hierarchical services as a candidate and study divide and conquer algorithms in Chapter 4. (ii) Approximate the solution. We explore approximation algorithms for compensability in Chapter 5. Related works are discussed (Section 5.5) at the end of Chapter 5.

Chapter 4

Divide and Conquer

To recall, in a Web services scenario, the component services of a composite service may themselves be composite leading to a hierarchical composition. In this chapter, we propose divide and conquer algorithms for hierarchical services composed in both top-down and bottom-up fashion.

4.1 Top-down Hierarchical Services

First, we consider hierarchical services constructed in a top-down fashion, represented as a hierarchical FSM (see Definition 1.2). For example, Fig. 3.24 depicts a hierarchical service H. Now, levels of a hierarchical service describe its functionality at different levels of abstraction. Thus, each level of a hierarchical service (except the leaf) has at least one supertransition (component). The presence of components leads us to think if we can apply "divide and conquer", that is, try to determine the minimal compensable set $MO(\mathcal{H})$ of a hierarchical service H by computing $MO(C)$ of its components independently. For example, let us consider the simple hierarchical FSM H with two levels shown in Fig. 1.5 (reproduced here in Fig. 4.1). The top level M_1 has two states, one initial and one final, with two transitions a, b from the initial to the final state. Transition a is a supertransition which describes a system M_2 similar to M_1, that is, two transitions c, d from the initial to the final state (but without supertransitions). The set $T_2 = \{c\}$ is a minimal compensable set of transitions of M_2. Now, looking at M_1 as a normal FSM (without supertransitions), $T_1 = \{b\}$ is also a minimal compensable set of transitions of M_1. We have furthermore that $T_1 \cup T_2$ is a minimal compensable set of transitions of $\mathcal{H}$. However, as expected, such a naive decomposition mechanism does not always work. If we take M_2 to be the FSM described in Fig. 3.26 and the associated minimal compensable set $T_2 = \{e_2, e_6, e_9\}$ as shown in Fig. 4.2, then $T_1 \cup T_2$ is not a minimal compensable set of transitions of $\mathcal{H}$. The reason is that T_2 is already a compensable set of transitions of $\mathcal{H}$ because all paths that pass through M_2 use at least one transition of T_2, so they can be differentiated from the path b. That is, the fact that a subset of transitions is a minimal compensable set of transitions is global to the whole FSM, not local.

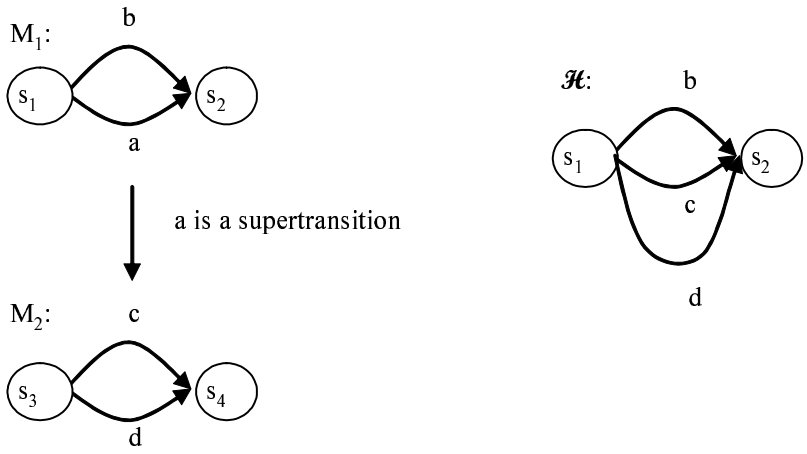

FIGURE 4.1 – A hierarchical FSM H and its corresponding $\mathcal{H}$.

In the following, we present a divide and conquer algorithm which uses the hierarchical structure to compute a minimal compensable set. Basically, we show that it suffices to run the algorithm with slightly different parameters on each component.

4.1.1 Simple Divide and Conquer

We consider simple components of an FSM M, that is, components which have only one initial and one final state.

Definition 4.1 (Simple Component) *An FSM $C = (Q', s'_0, s'_f, \mathcal{T}')$ is a simple component of $M = (Q, s_0, s_f, \mathcal{T})$ if*

- *$Q' \subsetneq Q$ and $\mathcal{T}' \subsetneq \mathcal{T}$,*
- *$\forall q \in Q \setminus Q'$, $q' \in Q'$, we have $(q, q') \in \mathcal{T}$ or $(q', q) \in \mathcal{T}$ implies $q' \in \{s'_0, s'_f\}$.*

For example, the service $M = (Q, s_0, s_f, \mathcal{T})$ in Fig. 4.3 has a simple component $C = (Q', s'_0, s'_f, \mathcal{T}')$. We say that a path $\rho = \tau_i|_{i=1\cdots n}$ *passes through* an FSM $M = (Q, s_0, s_f, \mathcal{T})$ if $\exists \tau_i, \tau_i \in \mathcal{T}$. We say that a path $\rho = \tau_i|_{i=1\cdots n}$ *belongs to* an FSM $M = (Q, s_0, s_f, \mathcal{T})$ if $\forall \tau_i, \tau_i \in \mathcal{T}$. We say that a path $\rho = \tau_i|_{i=1\cdots n}$ *does not touch* an FSM $M = (Q, s_0, s_f, \mathcal{T})$ if $\forall \tau_i, \tau_i \notin \mathcal{T}$. Further, for an FSM $M = (Q, s_0, s_f, \mathcal{T})$ and subset of transitions $\mathcal{T}_O \subseteq \mathcal{T}$, we define the following predicates : A path ρ is referred to as an invisible path if it does not use any transitions of $\mathcal{T}_\mathcal{O}$.

- $P_0(M, \mathcal{T}_O)$ holds if there does not exist more than one invisible path between any two states $s_1 \neq s_2 \in Q$ ($\mathcal{T}_O$ is a compensable set of transitions).
- $P_1(M, \mathcal{T}_O)$ holds if (i) $P_0(M, \mathcal{T}_O)$ holds, and (ii) there does not exist an invisible path from s_0 to s_f. Basically, the existence of an invisible path from the initial

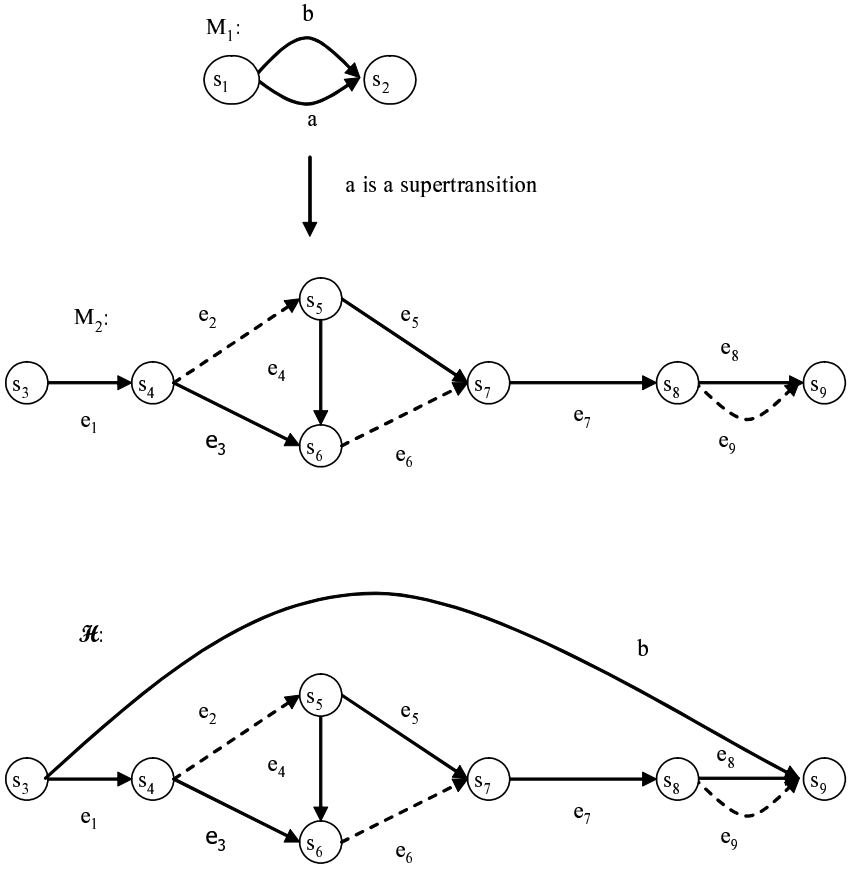

FIGURE 4.2 – A hierarchical FSM H and its $\mathcal{H}$ with M_2 corresponding to the FSM in Fig. 3.26 (dashed arrows represent visible transitions).

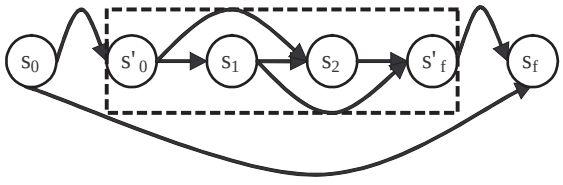

FIGURE 4.3 – Service $M = (Q, s_0, s_f, \mathcal{T})$ having simple component $C = (Q', s'_0, s'_f, \mathcal{T}')$.

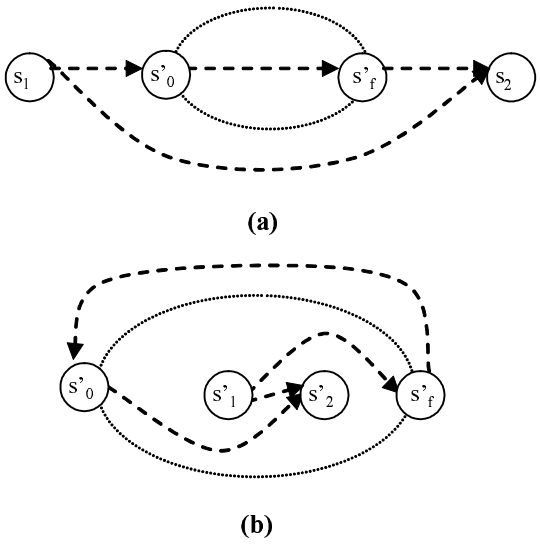

FIGURE 4.4 – Significance of (a) $P_1(M,\mathcal{T}_O)$ and (b) $P_{1'}(M,\mathcal{T}_O)$.

to final state of a component C might be a problem for the compensability of the enclosing M, if there exists a pair of states $s_1 \neq s_2$ of M with one path passing via C and the other not touching C as shown in Fig. 4.4(a).

- $P_{1'}(M,\mathcal{T}_O)$ holds if (i) $P_0(M,\mathcal{T}_O)$ holds, and (ii) there do not exist states s_1, $s_2 \in Q$, such that :
  - there is an invisible path from s_0 to s_2,
  - there is an invisible path from s_1 to s_f, and
  - there is an invisible path from s_1 to s_2.

  We refer to such a combination of states and transitions as an invisible reverse cyclic pattern between s_1 and s_2 (within M). Here also, the existence of an invisible reverse cyclic pattern within a component C of M, might be a problem with respect to the compensability of M, if there exists a path from the final to initial state of C which does not touch C as shown in Fig. 4.4(b) [because then there are two paths from s'_1 to s'_2 : (i) a direct path using (s'_1, s'_2) and (ii) a path via s'_f and s'_0].

By definition, $P_{1'}(M,\mathcal{T}_O) \Rightarrow P_1(M,\mathcal{T}_O) \Rightarrow P_0(M,\mathcal{T}_\mathcal{O})$, since for all s, there always exists a path from s to s. Let $\epsilon < 0 < 1 < 1'$. We define $\text{Best}(M,\mathcal{T}_O) = x \in \{\epsilon, 0, 1, 1'\}$, such that $P_x(M,\mathcal{T}_\mathcal{O})$ holds but not $P_{xx}(M,\mathcal{T}_\mathcal{O})$ with $xx > x$, with the convention $P_\epsilon(M,\mathcal{T}_\mathcal{O})$ is always true. Informally, Best refers to the best properties a given set of transitions can ensure, if visible. For instance, in Fig. 3.26 with $\mathcal{T}_O = \{e_2, e_6, e_9\}$, $P_0(\mathcal{T}_\mathcal{O})$ holds because $\mathcal{T}_\mathcal{O}$ is compensable, $P_1(\mathcal{T}_\mathcal{O})$ holds because every path from s_0 to s_f uses at least one transition of $\mathcal{T}_\mathcal{O}$, but $P_{1'}(\mathcal{T}_\mathcal{O})$ does not hold as there exists three invisible paths : e_4 from s_2 to s_3 ; e_1e_3 from s_0 to s_3 ; $e_5e_7e_8$ from s_2 to s_f. Thus,

$\text{Best}(M, \mathcal{T}_O = \{e_2, e_6, e_9\}) = 1$. For $x \in \{0, 1, 1'\}$, we define $MO_x(M)$ as the smallest $|\mathcal{T}_\mathcal{O}|$, such that $P_x(M, \mathcal{T}_\mathcal{O})$ holds.

We are now in a position to present the main result of this section.

Proposition 4.1 *Let C be a simple component of M, and $\mathcal{T}_1, \mathcal{T}_2$ be subsets of transitions of C respectively, such that Best$(C, \mathcal{T}_1)$ = Best$(C, \mathcal{T}_2)$. Then, for all subset of transitions $\mathcal{T}_\mathcal{O}$ of $M \setminus C$, we have Best$(M, \mathcal{T}_O \cup \mathcal{T}_1)$ = Best$(M, \mathcal{T}_O \cup \mathcal{T}_2)$.*

Proof Let $C = (Q', s'_0, s'_f, \mathcal{T}')$ and $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = x$. Let us assume that $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \epsilon$, that is, there exists a pair of states s_1, s_2 of M with invisible (for $\mathcal{T}_O \cup \mathcal{T}_2$) paths ρ_1, ρ_2 from s_1 to s_2, such that the states traversed by ρ_1 and ρ_2 are disjoint, but for s_1 and s_2. We show now that $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

If both ρ_1 and ρ_2 do not touch C, then $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$. If both ρ_1 and ρ_2 belong to C, then $P_0(C, \mathcal{T}_2)$ does not hold, which means $P_0(C, \mathcal{T}_1)$ does not hold, implying $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

If $s_1, s_2 \in (Q \setminus Q') \cup \{s'_0, s'_f\}$, and $\rho = \rho_1$ or ρ_2 passes through C, then there exists an invisible (for $\mathcal{T}_2$) path from s'_0 to s'_f (a subpath of ρ). Given this, $P_1(C, \mathcal{T}_2)$ does not hold, which implies that $P_1(C, \mathcal{T}_1)$ does not hold. Hence, there exists an invisible (for $\mathcal{T}_1$) path from s'_0 to s'_f, and an invisible (for $\mathcal{T}_O \cup \mathcal{T}_1$) path ρ' can be constructed from this path and ρ. As such, there are two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 : $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

If $s_1, s_2 \in Q'$, and $\rho = \rho_1$ or ρ_2 passes through C, then there exists an invisible reverse cyclic pattern (for $\mathcal{T}_2$) between s'_0 and s'_f. Given this, $P_{1'}(C, \mathcal{T}_2)$ does not hold, which implies that $P_{1'}(C, \mathcal{T}_1)$ does not hold. Hence, there exists an invisible (for $\mathcal{T}_1$) reverse cyclic pattern between s'_0 and s'_f, and an invisible (for $\mathcal{T}_O \cup \mathcal{T}_1$) path ρ' can be constructed from this pattern and ρ. As such, there are two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 : $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$.

The cases where $s_1 \in Q' \setminus \{s'_0, s'_f\}$, $s_2 \notin Q'$, or both ρ_1, ρ_2 pass through C, are not possible because then the paths would meet in s'_0 and/or s'_f.

Hence, $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \epsilon \implies \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) = \epsilon$. By symmetry between $\mathcal{T}_1$ and $\mathcal{T}_2$, we have the equivalence : $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) \geq 0$ iff $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1) \geq 0$. Now, for all $x \in \{1, 1'\}$, we can enrich M to $F_x(M)$ with $\text{Best}(M, \mathcal{T}_O) \geq x$ iff $\text{Best}(F_x(M), \mathcal{T}_O) \geq 0$. Applying it to M, we get $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1)$. The functions F_x are given schematically in Fig. 4.5

□

For $x \in \{0, 1, 1'\}$, we define $\mathcal{T}_x(M)$ as a smallest subset $\mathcal{T}_\mathcal{O}$ of transitions of M, such that $P_x(M, \mathcal{T}_\mathcal{O})$ holds. For a subset of transitions T of a component C of M, we also denote by $\mathcal{T}_x^{T,C}(M)$ a smallest set $\mathcal{T}_\mathcal{O}$, such that $\mathcal{T}_O \cap C = T$ and $P_x(M, \mathcal{T}_\mathcal{O})$ holds. As far as we know, every algorithm to compute the minimal compensable set of transitions is recursive taking the set of transitions considered compensable as input. It is easy to modify them to input in the beginning not $\emptyset$ but T, and disallowing to select any new transitions in C such that they compute $\mathcal{T}_x^{T,C}(M)$, and they do it *faster* than $\mathcal{T}_x(M)$ because they do not need to choose among the transitions of C. As proved in

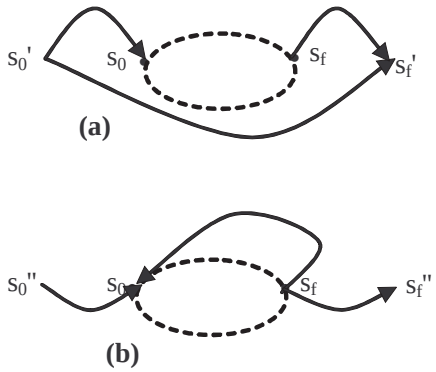

FIGURE 4.5 – Computation of (a) $F_1(M)$ and (b) $F_{1'}(M)$.

Proposition 4.1, the size of $\mathcal{T}_{\mathcal{O}}$ is constant for several T, such that $Best(C,T) = y$. If $|T'| > |T|$ with $Best(C,T) = Best(C,T')$, then $|\mathcal{T}_x^{T',C}(M)| > |\mathcal{T}_x^{T,C}(M)|$. We can use this idea to compute $\mathcal{T}_x(M)$ in a compositional manner, for a service M having simple component C :

Algorithm MinimalDecomposition(M, C) :

1. Compute a minimal compensable set $\mathcal{T}_y(C)$ of transitions of C, $\forall y \in \{0, 1, 1'\}$.
2. Compute a minimal compensable set $\mathcal{T}_0^{\mathcal{T}_y(C),C}(M)$ of transitions of M, for all y.
3. Output a set of smallest size among $\mathcal{T}_0^{\mathcal{T}_y(C),C}(M)$.

For example, let us consider the service M having simple component C in Fig. 4.3.

1. A minimal set $\mathcal{T}_0(C) = \{(s'_0, s_2), (s_1, s'_f)\}$, $\mathcal{T}_1(C) = \{(s'_0, s_1), (s_2, s'_f)\}$, and $\mathcal{T}_{1'}(C) = \{(s'_0, s_2), (s_1, s'_f), (s_1, s_2)\}$.
2. The corresponding compensable sets of M : $\mathcal{T}_0^{\mathcal{T}_0(C),C}(M) = \{(s'_0, s_2), (s_1, s'_f), (s_0, s_f)\}$ of size 3, $\mathcal{T}_0^{\mathcal{T}_1(C),C}(M) = \{(s'_0, s_1), (s_2, s'_f)\}$ of size 2, and $\mathcal{T}_0^{\mathcal{T}_{1'}(C),C}(M) = \{(s'_0, s_2), (s'_1, s_f), (s_1, s_2)\}$ of size 3.
3. $\mathcal{T}_0^{\mathcal{T}_1(C),C}(M)$ is a minimal compensable set of M.

The next theorem states the complexity of the above algorithm.

Theorem 4.1 *Let $H = (M_i)_{i=1}^n$ be a hierarchical service. It is NP-complete in the size of H to compute $MO(\mathcal{H})$. Moreover, it takes at most time $O(\sum_{i=1}^n 2^{|M_i|})$.*

Proof It suffices to compute a minimal set $\mathcal{T}_v(M_i)$ of transitions of M_i, for all $v \in V = \{0, 1, 1'\}$. These actions are performed from bottom of the hierarchy to top. To compute $\mathcal{T}_v(C_i)$ for a module M_i using $M_j, M_k, j, k > i$, we use $\mathcal{T}_{v'}(M_j)$ and $\mathcal{T}_{v''}(M_k)$ which have already been computed. Indeed, it suffices to compute a minimal set $\mathcal{T}_v^{\mathcal{T}_{v'}(M_i)\cup\mathcal{T}_{v''}(M_k),M_j\cup M_k}(M)$ of transitions of M, for all valuations v', v''. Then, it suffices to output a set among $\mathcal{T}_v^{\mathcal{T}_{v'}(M_i)\cup\mathcal{T}_{v''}(M_k),M_j\cup M_k}(M)$ of minimal size. For more details, the interested reader is referred to Theorem 4.3. □

It is important to note that since a service is in reality a hierarchical service (with hierarchy height of 1), we know that the problem is at least NP-hard. However, the complexity could be exponentially worse for hierarchical graphs, since a small hierarchical graph can represent an exponentially bigger flat graph. We prove that this is not the case. Moreover, we prove that the complexity is linear in the number of hierarchy level, and exponential only in the size of each component. That is, we prove that with a smart algorithm, one can compute efficiently the absolute minimal compensable size even for huge hierarchical systems, as long as each component is small enough. The best case comparison is with respect to a hierarchical service of diameter $O(2^n)$, having n components of size 2. The brute force non-compositional method run on $\mathcal{H}$ takes time $O(2^{2^n})$, while our method takes $O(n)$, that is a doubly exponential improvement (one exponential due to the reuse of components and another due to decomposition).

An obvious drawback of the above algorithm is that it does not actually reduce the size of M on which to compute $MO(M)$. While it is definitely faster to compute $MO(M)$ with a fixed set T of transitions of its component C, the check whether a subset of transitions satisfies the predicates $P_0, P_1, P_{1'}$ is still not linear with respect to $|M|$. We experienced it during our experimental evaluations as well (see Section 4.1.4). Thus, in the sequel, we present a mechanism which uses Proposition 4.1 to actually reduce the size of M on which to compute $MO(M)$, by allowing us to substitute a component C of M by a much smaller D.

Given a simple component $C = (Q', s'_0, s'_f, \mathcal{T}')$ of $M = (Q, s_0, s_f, \mathcal{T})$ and $D = (Q'', s''_0, s''_f, \mathcal{T}'')$, we denote by $M_C(D)$ the FSM obtained on substituting C by D, that is, $M_C(D) = (\bar{Q}, \bar{s_0}, \bar{s_f}, \bar{\mathcal{T}})$ with

- $\bar{Q} = Q \setminus Q' \cup Q''$,
- if $s_0 \in \bar{Q}$ then $\bar{s_0} = s_0$, else $\bar{s_0} = s''_0$,
- if $s_f \in \bar{Q}$ then $\bar{s_f} = s_f$, else $\bar{s_f} = s''_f$,
- $\bar{\mathcal{T}} = \mathcal{T} \setminus \mathcal{T}' \cup \mathcal{T}'' \cup \mathcal{I}$, where $\mathcal{I} = \{(q, s''_y) \mid (q, s'_y) \in \mathcal{T} \wedge y \in \{0, f\}\} \cup \{(s''_y, q) \mid (s'_y, q) \in \mathcal{T} \wedge y \in \{0, f\}\}$.

Note that $M_C(C) = M$ for all C. Also, note that $|M_C(D)| = |M| - |C| + |D|$. We now define the common characteristics C and D should have, such that we can safely replace C by D.

Theorem 4.2 *Let C be a simple component of an FSM $M = M_C(C)$. We select an FSM D, such that $\forall x \in \{0, 1, 1'\}$, $MO_x(D) - MO(D) = MO_x(C) - MO(C)$. Then,*

$MO(M) = MO(M_C(D)) + MO(C) - MO(D)$.

Proof Let $M = (Q, s_0, s_f, \mathcal{T})$ and $C = (Q', s'_0, s'_f, \mathcal{T}')$. Further, let $\mathcal{T}_{\mathcal{OM}}$ be a minimal compensable set of $M_C(C)$. Then, $\mathcal{T}_{\mathcal{OM}}$ can be partitioned as follows : $\mathcal{T}_{OM} = \mathcal{T}^1_O \uplus \mathcal{T}_{\mathcal{OC}}$, where $\mathcal{T}^1_O = \{t \mid t \in \mathcal{T}_{OM} \wedge t \in (\mathcal{T} \backslash \mathcal{T}')\}$ and $\mathcal{T}_{OC} = \{t \mid t \in \mathcal{T}_{OM} \wedge t \in \mathcal{T}'\}$. Let $\text{Best}(C, \mathcal{T}_{OC}) = x$. By definition, $|\mathcal{T}_{OC}| \geq MO_x(C)$. Now, we consider a set $\mathcal{T}_{OM'} = \mathcal{T}^1_O \uplus \mathcal{T}_{\mathcal{OD}}$, where $\text{Best}(D, \mathcal{T}_{OD}) = x$ and $\mathcal{T}_{OD}$ is a minimal set such that $P_x(D, \mathcal{T}_{\mathcal{OD}})$ holds, that is, $|\mathcal{T}_{\mathcal{OD}}| = MO_x(D)$. Then, by Proposition 4.1, $\mathcal{T}_{OM'}$ is at least a compensable set of $M_C(D)$, and $\mathcal{T}_{OM'} \geq MO(M_C(D))$.

Given this, applying the hypothesis

$MO_x(D) - MO(D) = MO_x(C) - MO(C)$

$\Rightarrow |\mathcal{T}_{OD}| - MO(D) \leq |\mathcal{T}_{OC}| - MO(C)$ (as $MO_x(D) = |\mathcal{T}_{OD}|$ and $MO_x(C) \leq |\mathcal{T}_{OC}|$)

$\Rightarrow |\mathcal{T}^1_O| + |\mathcal{T}_{OD}| - MO(D) \leq |\mathcal{T}^1_O| + |\mathcal{T}_{OC}| - MO(C)$ (adding $|\mathcal{T}^1_O|$ on both sides)

$\Rightarrow |\mathcal{T}_{OM'}| - MO(D) \leq |\mathcal{T}_{OM}| - MO(C)$

$\Rightarrow MO(M_C(D)) - MO(D) \leq MO(M) - MO(C)$ (as $MO(M_C(D) \leq |\mathcal{T}_{OM'}|$ and $|\mathcal{T}_{OM}| = MO(M)$).

Symmetrically, D is a component of $N = M_C(D)$, and we have with the same reasoning $MO(N_D(C)) - MO(C) \leq MO(N) - MO(D)$. Noting that $N_D(C) = M$, we get $MO(M) = MO(M_C(D)) + MO(C) - MO(D)$. □

We now give exhaustively every possible FSM D one can need to replace a simple component C. We may need an FSM D having one of the following characteristics :

- $MO_{1'}(D) - MO(D) = 0$ (which implies $MO_1(D) - MO(D) = 0$).
- $MO_{1'}(D) - MO(D) = 1$ and $MO_1(D) - MO(D) = 0$. The service in Fig. 3.26 exhibits this characteristic.
- $MO_{1'}(D) - MO(D) = 1$ and $MO_1(D) - MO(D) = 1$. It is the case for the service $D = (\{s_1, s_2\}, s_1, s_2, \{(s_1, s_2)\})$.
- $MO_{1'}(D) - MO(D) = 2$ and $MO_1(D) - MO(D) = 1$. The service $D = (Q, s_1, s_4, \mathcal{T})$ where $Q = \{s_1, s_2, s_3, s_4\}$ and $\mathcal{T} = \{(s_1, s_3), (s_1, s_2), (s_2, s_3), (s_2, s_4), (s_3, s_4), (s_1, s_4)\}$ is such an example.

Indeed, for all $C, MO_1(C) - MO(C) \leq 1$. Moreover, if $MO_{1'}(C) - MO_1(C) \geq 1$, it is sufficient to consider a D having $MO_{1'}(D) - MO_1(D) = 1$. Basically, for a simple component C of M, the presence of several invisible reverse cyclic patterns within C is an issue with respect to the compensability of M, iff there exists an invisible path ρ of M_C connecting the final state of C to its initial state. However, such a path ρ is unique and it is sufficient for a transition τ of ρ to be visible, irrespective of the actual number (≥ 1) of invisible reverse cyclic patterns within C. For example, let us consider the FSM $M_C(C)$ in Fig. 4.6 having component C as shown in Fig. 4.7. A minimal compensable set of C is shown by dashed arrows in Fig. 4.7. Then, for C, $MO(C) = 11, MO_1(C) - MO(C) = 0$ and $MO_{1'}(C) - MO(C) = 2$. Fig. 4.8 shows $M_C(D)$ on substituting component C with a suitable $D = (Q', s'_i, s'_j, \{a', b', c', d', e'\})$.

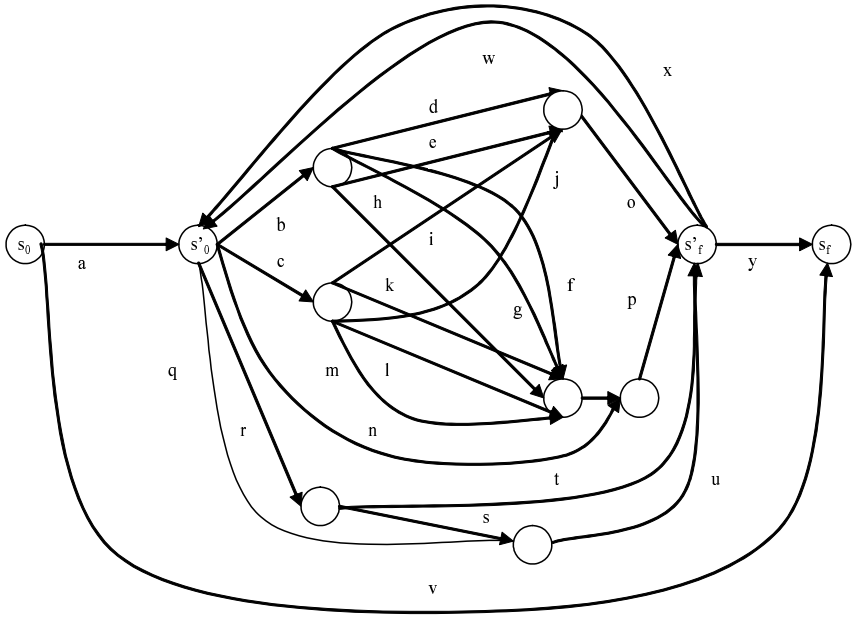

FIGURE 4.6 – Sample $M_C(C)$.

Now, $MO(D) = 2$, and $MO(M_C(D)) = 4$ as shown by the dashed arrows in Fig. Fig. 4.8. Then, applying Theorem 4.2, $MO(M) = MO(M_C(D)) + MO(C) - MO(D) = 4 + 11 - 2 = 13$.

Hence, we can state the following proposition :

Proposition 4.2 *There exists a (small) constant Cst such that for all FSMs C, there exists an FSM D with $|D| \leq Cst$ and $MO_x(D) = MO_x(C)$ for all $x \in \{0, 1, 1'\}$. Also, MO_x can be computed efficiently using an algorithm computing MO.*

Proof It remains to show how to compute MO_1 and $MO_{1'}$. We simply extend M to FSMs M_1 and $M_{1'}$, such that, $MO_1(M) = MO(M_1)$ and $MO_{1'}(M) = MO(M_{1'})$ respectively. The corresponding M_1 and $M_{1'}$ to compute $MO_1(M)$ and $MO_{1'}(M)$, are shown in Fig. 4.5(a and b) respectively. □

4.1.2 Hierarchical Recovery

The visibility requirement discussed so far in this section can be considered as global where we needed complete visibility over the service descriptions and logs of all descendents (the minimal compensable set was computed based on the flat representation $\mathcal{H}$ of the given hierarchical composition H). Here, we present a more secure framework on the lines of Chapter 3, with services in a hierarchy having only as much visibility

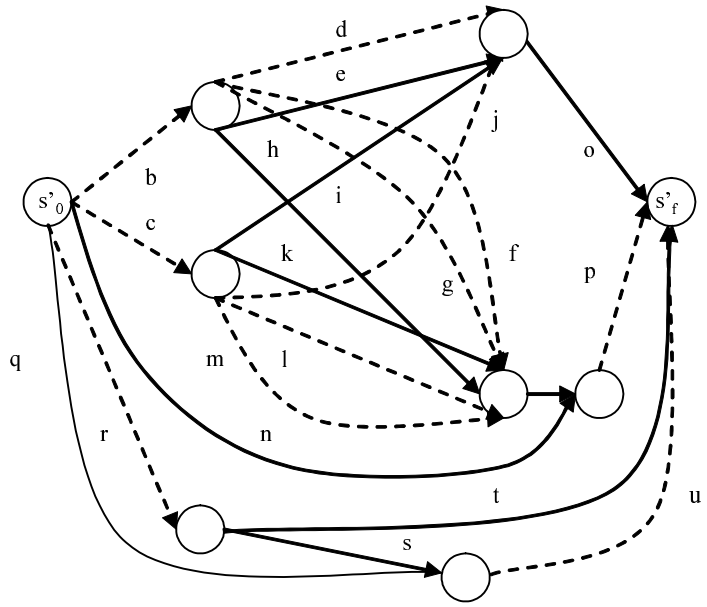

FIGURE 4.7 – Component C of M in Fig. 4.6 (the dashed arrows show a minimal compensable set of C).

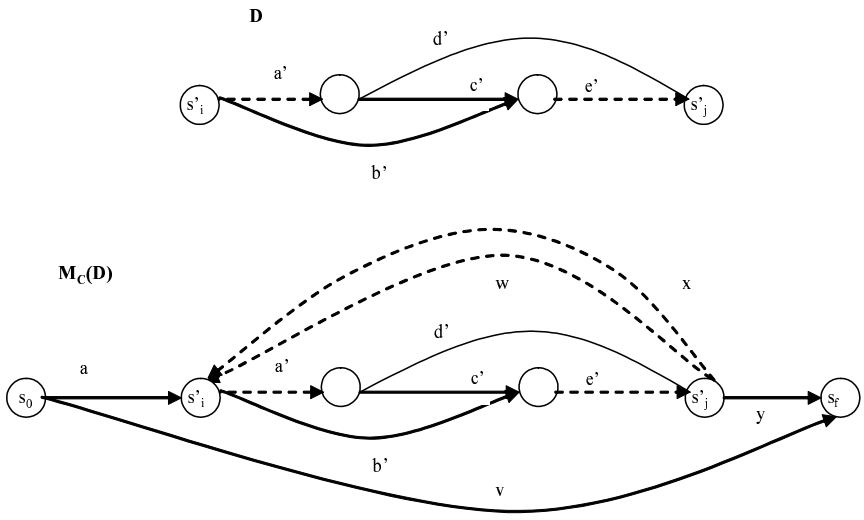

FIGURE 4.8 – FSM D and $M_C(D)$ corresponding to the $M_C(C)$ in Fig. 4.6 (the dashed arrows show a minimal compensable set of D and $M_C(D)$).

over others as required. We discuss the information sharing (visibility) required between services in such a restricted visibility environment. During computation of the minimal compensable size (Section 4.1.1), the interactions occur in a bottom-up fashion with children sharing information with their parents. Then, the services assign a minimal compensable set of transitions as visible in a top-down fashion. Finally, the actual execution, including any compensation required in the event of a failure, also proceeds in a top-down fashion. Here, we only need parent-child interactions (no interaction needed with ancestors or descendents), which can be considered as a basic visibility assignment of Chapter 3. To ease illustration, we only present the information sharing (visibility) required between a pair of parent $M_C(C) = (Q, s_0, s_f, \mathcal{T})$ and child $C = (Q', s'_0, s'_f, \mathcal{T}')$ services of a hierarchy H :

Computation Stage : By computation stage, we refer to the static computation of the minimal compensable set $\mathcal{T}_O$ of M, and assigning the transitions in $\mathcal{T}_O$ as visible. During computation of MO, using Theorem 4.2, it is sufficient if a child C only shares $\forall x \in \{1, 1'\}$, $MO_x(C) - MO(C)$ with its parent M. Once $MO(M)$ has been computed using $D = (Q'', s''_0, s''_f, \mathcal{T}'')$ as the substitute FSM, let $\mathcal{T}_{O'}$ be a minimal compensable set of $M_C(D)$. Then, $\mathcal{T}_{O'}$ can be partitioned as follows : $\mathcal{T}_{O'} = \mathcal{T}_{O1} \uplus \mathcal{T}_{O2}$, where $\mathcal{T}_{O1} = \{t \in \mathcal{T}_{O'} \mid t \in \mathcal{T} \backslash \mathcal{T}''\}$ and $\mathcal{T}_{O2} = \{t \in \mathcal{T}_{O'} \mid t \in \mathcal{T}''\}$. Then, it is sufficient for M to assign the transitions in $\mathcal{T}_{O1}$ as visible (will be logged on execution). Note that M does not need visibility over the service description of C as $\mathcal{T}_{O1}$ was computed based on $M_C(D)$. The subset of transitions which C needs to assign as visible depends on the characteristics of D with visible set $\mathcal{T}_{O2}$, that is, if $\text{Best}(D, \mathcal{T}_{O2}) = x \in \{0, 1, 1'\}$, then C should have visibility over a minimal subset $\mathcal{T}_{OC} \subseteq \mathcal{T}'$ which satisfies $P_x(C, \mathcal{T}_{OC})$. Recall that we substitute a component C, for which $MO_{1'}(C) - MO_1(C) \geq 1$, with an FSM D having $MO_{1'}(D) - MO_1(D) = 1$. Thus, we need to make an exception to the above visibility requirement of C : if $\text{Best}(D, \mathcal{T}_{O2}) = 1'$ and $MO_{1'}(C) > MO_1(C)$, then we need to add a transition τ of the invisible path of M connecting final state s'_f to initial state s'_0 of C, to $\mathcal{T}_{O1}$ (make τ visible). And, with this addition, it is sufficient for C to have visibility over a minimal subset $\mathcal{T}_{OC} \subseteq \mathcal{T}'$ which satisfies $P_1(C, \mathcal{T}_{OC})$.

Execution Stage : In this stage, we discuss the interactions required to reconstruct the actual execution sequence from the log (or observation projection) in the event of a failure, and performing the actual compensation. Without global visibility, the logs or visibility projections are maintained locally by each service, and are visible only to that service. Other services, including even their parents or children, do not have direct visibility over the logs. Given this, we need some mechanism to synchronize the logs of at least parent-child services. For example, let us consider the FSM M in Fig. 4.9 with component $C = (\{s'_0, s'_f\}, s'_0, s'_f, \{f\})$ and a minimal compensable set $\{c, d, e, j\}$ visible. Then, for the execution sequence $abcdehibcghjbfehibcghibfehk$, the local visibility projections at M and C would be j and $cdecece$ respectively. With these visibility projections, we know that C was invoked five times, but we need some synchronization mechanism to know how many times C was invoked (the invisible i loop was repeated) before j's execution, and after its execution. To overcome this, we assume the use of global timestamps to synchronize parent-child logs (alternate strategies are discussed later). First, we give the execution sequence reconstruction

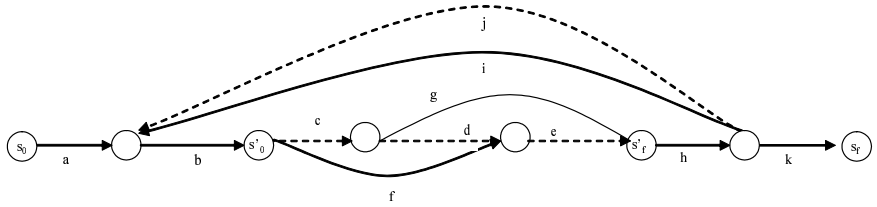

FIGURE 4.9 – Sample FSM $M_C(C)$ to show the need for log synchronization.

algorithm. Once the execution sequence of M has been computed, M compensates its executed transitions in the reverse order. For each invocation of component C, it asks C to perform the necessary compensation, which would again involve C computing the execution sequence corresponding to that invocation of itself, and compensating its executed transitions.

Execution sequence reconstruction algorithm under limited visibility.

Input. FSM $M_C(C) = \{Q, s_0, s_f, \mathcal{T}\}$, $C = \{Q', s'_0, s'_f, \mathcal{T}'\}$, visible set $\mathcal{T}_{O1}$ of M_C (as defined above), and visibility projection $(\sigma = \tau_1\tau_2\cdots\tau_n, q_{n+1})$. We assume that $q_{n+1} = s_f$ if a failure did not occur with respect to M (but failure occurred at a higher level, and the successful execution of component FSM M needs to be compensated).

Output. The unique path ρ to be compensated with $\text{Obs}_O^{last}(\rho) = (\sigma, q_{n+1})$.

Initialization. Set $\rho := \epsilon$, index $i := 1$, current state $s_{curr} := s_0$. The next state $s_{next} := q_{n+1}$ if $n = 0$ (that is, no transitions were logged), else $s_{next} :=^* \tau_1$ where $\sigma = \tau_1\tau_2\cdots\tau_n$.

FSM $M' = \{Q \setminus Q' \cup \{s'_0, s'_f\},\ s_0,\ s_f,\ \mathcal{T}\setminus(\mathcal{T}' \cup \mathcal{T}_{O1}) \cup \{(s'_0, s'_f)\}\}$.

while $(s_{curr} \neq q_{n+1})$ do

Determine the executed path ρ_1 from s_{curr} to s_{next}.

- If there exists only one path ρ_2 of M' connecting s_{curr} to s_{next}, then set $\rho_1 := \rho_2$. Note that ρ_1 could contain (s'_0, s'_f), which implies that the component C will be asked to compensate its execution corresponding to that invocation later during compensation (after the execution sequence has been reconstructed).
- If there exists two paths $\rho_2 \neq \rho_3$ of M' connecting s_{curr} to s_{next}, with ρ_3 containing the transition (s'_0, s'_f). Then, M needs to ask C if C was invoked. If so, set $\rho_1 := \rho_3$, otherwise set $\rho_1 := \rho_2$.
- If there exists more than two paths, then it implies the existence of a cycle $s_{curr} \xrightarrow{\rho_4} s_c \xrightarrow{\rho_5} s'_0 \xrightarrow{(s'_0,s'_f)} s'_f \xrightarrow{\rho_6} s_c$ with respect to an intermediate state s_c, and let ρ_7 be the alternate path connecting s'_f to s_{next}. Given this, M needs to check with C the number of times (say $m \geq 0$) C was invoked. If
  - $m = 0$: Set ρ_1 to the alternate path connecting s_{curr} to s_{next}, which does not pass via C.
  - $m = 1$: Set $\rho_1 := \rho_4\rho_5(s'_0, s'_f)\rho_7$.
  - $m > 1$: Set $\rho_1 := \rho_4\rho_5(s'_0, s'_f)\rho_6\cdots[\rho_5(s'_0, s'_f)\rho_6]_{m-1}\rho_5(s'_0, s'_f)\rho_7$.

Append ρ_1 to ρ. Further, append τ_i to ρ if $n \neq 0$.
if $s_{next} = q_{n+1}$, then set $s_{curr} := q_{n+1}$ (to terminate).
else
Set $s_{curr} := \tau_i^*$. If $i < n$, then increment i and set $s_{next} :=^* \tau_i$, else set $s_{next} := q_{n+1}$.
endif
endwhile

Proposition 4.3 *With reference to the execution sequence reconstruction algorithm, if there exists greater than one path between a pair of states $\tau_1^* \neq^* \tau_2, \tau_1, \tau_2 \in \mathcal{T}_{O1}$ of M', then C can determine the number of times $m \geq 0$ it was invoked between an execution of τ_1 followed by τ_2.*

Proof If there exists two paths between $\tau_1^* \neq^* \tau_2$, then $\mathcal{T}_{OC}$ satisfies at least $P_1(C, \mathcal{T}_{OC})$. Otherwise, $\text{Best}(D, \mathcal{T}_{O2}) = 0$, and there exists two paths between $\tau_1^* \neq^* \tau_2$ in $M_C(D)$ as well with $\mathcal{T}_{O'}$ visible. The implication of $\mathcal{T}_{OC}$ satisfying at least $P_1(C, \mathcal{T}_{OC})$ is that at least one transition of C is logged each time it is invoked. Given this, C can answer the number of times $m \geq 0$ it was invoked between an execution of τ_1 followed by τ_2, based on its log between the logged times (global timestamps) of τ_1 and τ_2. □

Finally, M needs to compensate the reconstructed execution sequence ρ. M starts compensating in the usual reverse manner. As soon as it encounters a (s'_0, s'_f), it asks C to perform the necessary compensation. As $\text{Best}(C, \mathcal{T}_{OC}) \geq 0$ (otherwise, $\text{Best}(D, \mathcal{T}_{O2}) < 0$ and there exists greater than one path between a pair of states in D with $\mathcal{T}_{O2}$ visible), C can determine its execution sequence corresponding to an invocation based on its local synchronized log, and perform the necessary compensation.

If global timestamps are infeasible, an alternate strategy would be as follows : The FSM M, in addition to logging the transitions in $\mathcal{T}_{O1}$, also inserts a special marker (say X) in its local log each time it invokes component C. For example, for the FSM M in Fig. 4.9 with compensable set $\{c, d, e, j\}$ visible, the execution sequence *abcdehibcghjbfehibcghibfehk* would lead to logging $XXjXXX$ at M. Note that this is equivalent to having (s'_0, s'_f) visible in M', and clearly, if we delete (s'_0, s'_f) from M', then there does not exist greater than one path between any pair of states in M'.

The use of markers leads to some redundancy, and as our goal is clearly to minimize logging, we give another strategy below which in most cases leads to a shorter combined log. With this strategy, M no longer needs to log a marker X each time it invokes C. Rather, for each invocation of C, M logs a marker X stamped with its local time (or some unique local identifier) *only if* the last element in its log is a visible transition (and not another marker). For the example scenario (Fig. 4.9), it would lead to the log XjX at M. With logs XjX at M and *cdecece* at C, we still do not know how many times C was invoked before and after the execution of j, and hence cannot reconstruct the execution sequence. To overcome this, for each invocation of C that M inserts a

marker X in its local log, it passes the same to C, and C also inserts X in its local log (before logging anything corresponding to that invocation). For our example scenario, this would lead to the logs XjX and $XcdecXece$. Given this, each time a marker X is encountered while parsing M's log, we know that the portion of C's log between X and the next marker (or end of the log) corresponds to the execution between that of the visible transitions t_1 and t_2, logged before and after X in M's log respectively, and hence can determine the number of times C was invoked between t_1's and t_2's execution.

Finally, we give a sample run of the hierarchical recovery mechanism outlined in this section on the hierarchical system in Fig. 3.24. We refer to the FSMs at levels $1, 2$ and 3 as M, M^1 and M^2, respectively. To start with (in a bottom-up fashion), M^2 computes $x \in \{0, 1, 1'\}, MO_x(M^2)$, and sends the differences $MO_{1'}(M^2) - MO(M^2) = MO_1(M^2) - MO(M^2) = 1$ to M^1. Then, M^1 substitutes e_{17} with an FSM $D = \{\{s'_0, s'_f\}, s'_0, s'_f, \{(s'_0, s'_f)\}\}$ having similar characteristics (by Theorem 4.2), that is, $MO_{1'}(D) - MO(D) = MO_1(D) - MO(D) = 1$. Let the substituted M^1 be M', then it computes $x \in \{0, 1, 1'\}, MO_x(M')$, and sends the differences $MO_{1'}(M') - MO(M') = 1$ and $MO_1(M') - MO(M') = 0$ to M. On receiving this, M substitutes e_2 by the FSM $E = (Q'', s''_0, s''_f, \mathcal{T}'')$ where $Q'' = \{s''_0, s''_1, s''_2, s''_f\}$ and $\mathcal{T}'' = \{(s''_0, s''_1), (s''_0, s''_2), (s''_1, s''_2), (s''_1, s''_f), (s''_2, s''_f)\}$ (for which $MO_{1'}(E) - MO(E) = 1$ and $MO_1(E) - MO(E) = 0$). Let the substituted FSM M be M'', then we have a minimal compensable size $MO(M'') = 2$.

Once the minimal compensable size has been computed, the next step is to assign the compensable sets, which proceeds in top-down order. Let $\mathcal{T}_O = \{(s''_0, s''_1), (s''_2, s''_f)\}$ be a minimal compensable set of M''. Given this, M does not need to assign any of its transitions as visible. As $\text{Best}(E, \{(s''_0, s''_1), (s''_2, s''_f)\}) = 1$, then M^1 needs to assign a minimal set $\mathcal{T}_{O^1}$ as visible, such that $\text{Best}(M^1, \mathcal{T}_{O^1}) = 1$, say $\mathcal{T}_{O^1} = \{e_{11}, e_{15}\}$. On the same lines, with only the transitions in $\mathcal{T}_{O^1}$ of M^1 visible, none of D's transitions are visible, as such $\text{Best}(D, \emptyset) = 0$. Then, M^2 only needs to assign a minimal set $\mathcal{T}_{O^2}$ as visible, such that $\text{Best}(M^2, \mathcal{T}_{O^2}) = 0$, say $\mathcal{T}_{O^2} = \{e_{23}\}$. The hierarchical system, with the visible transitions denoted by dashed arrows, is shown in Fig. 4.10.

Now, let us assume that a failure occurs while executing e_4, and the execution sequence till then is $e_1e_{11}e_{14}e_{16}e_{21}e_{22}e_{24}$. For simplicity, we also assume that global timestamps are used. Then, the logs at M, M^1 and M^2 are s_3 (the state before failure), e_{11} and empty, respectively. Given this, M starts the execution sequence reconstruction. There exists two paths $e_1e_3 \neq e_1e_2$ between the states s_1 and s_3, one containing the supertransition e_2. Then, the component M^1 corresponding to e_2 is asked to check the number of times it was invoked between the start time and logging time of s_3. M^1 checks its log, finds e_{11} logged during that time, and replies that it was invoked once. So, the execution sequence at M is set to e_1e_2, and M starts compensation in the reverse order. As the first transition to be compensated e_2 is a supertransition, the corresponding component M^1 is asked to compensate its execution first. Before it can compensate, M^1 also first needs to reconstruct its execution sequence. As the first logged transition e_{11} is an outgoing of its initial state s_{11}, e_{11} is appended to its execution sequence ρ. Then, it checks for paths between $e^*_{11} = s_{12}$ and its final state s_{16}. Recall that it is checking for paths in the FSM M^1 from which the visible transitions (including e_{15}) have

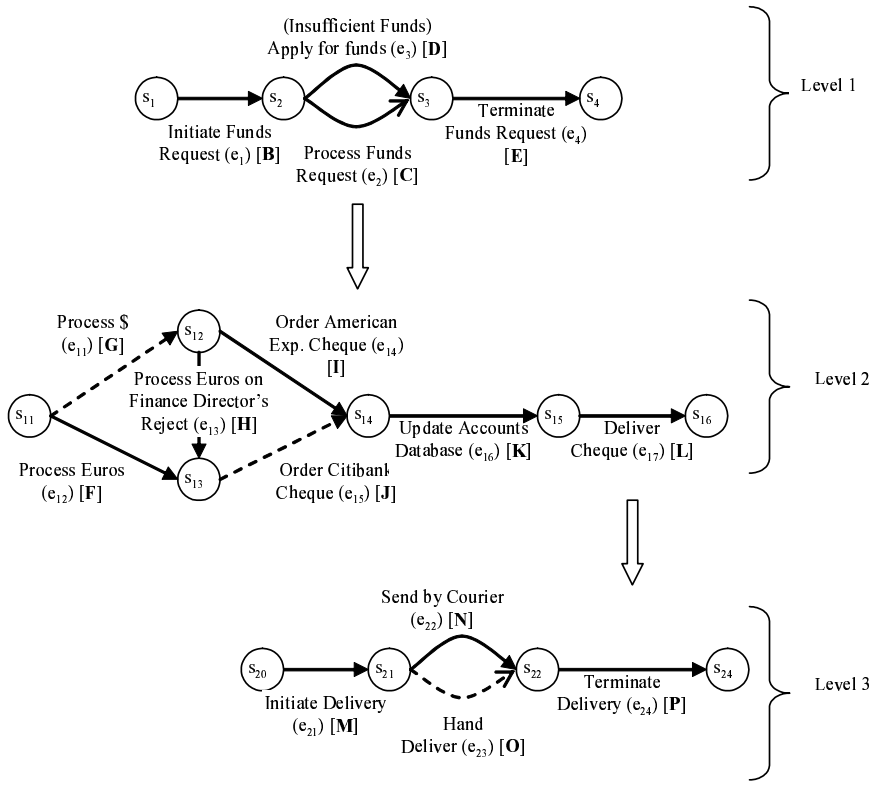

FIGURE 4.10 – Visibility assignment of the hierarchical system in Fig. 3.24 (visible transitions denoted by dashed arrows).

been deleted. As there exists only one such path $e_{14}e_{16}e_{17}$, it is appended to ρ, leading to the execution sequence $e_{11}e_{14}e_{16}e_{17}$. With the execution sequence determined, M^1 starts compensation in the usual reverse order. As the first transition to be compensated e_{17} is again a supertransition, it asks the corresponding component M^2 to compensate its execution first. For M^2, its log is empty and there exists only one path $e_{21}e_{22}e_{24}$ (after the visible transition e_{23} has been deleted) between its initial state s_{20} and final state s_{24}, leading to the execution sequence $e_{21}e_{22}e_{24}$. The rest of the process is simply invoking the respective compensating transitions of the transitions in the determined execution sequences.

4.1.3 Extensions

Here, we address the problem that a service may have been constructed hierarchically, but the hierarchical structure specifying the components at each level is not available (or accessible) anymore. That is, given a (flat representation of a) composite service, we would like to recover the hierarchical structure from it. Also, the effectiveness of our divide and conquer algorithms (Section 4.1.1) are clearly proportional to the size the components, that is, the larger the components the better as large components can possibly be refined further (which would imply that we should be interested in the smallest component C, however then the M_C would be large). Towards this end, we show how to recover the largest simple component from a given composite service M in Section 4.1.3.1. The algorithm can then be called inductively to obtain the hierarchical structure of M. We also extend the divide and conquer algorithms to handle complex components (having more than one initial and final states) in Section 4.1.3.2.

4.1.3.1 Computing the Largest Simple Component

First, we present a linear time (in the number of transitions) algorithm to compute a smallest simple component C of an FSM M, knowing its initial state, final state and an outgoing transition of the initial state.

Algorithm to compute a smallest simple component C of the given service M.
Input. A service $M = (Q, s_0, s_f, \mathcal{T})$, $\tau = (s, s_1) \in \mathcal{T}$ and $t \in Q$.
Output. A smallest simple component $C = (Q', s, t, \mathcal{T}')$ of M with $\tau \in \mathcal{T}'$.
Initialization. $\mathcal{T}' = \phi$, $S = \{\tau\}$, $Q' = \{s, s_1, t\}$.

1. Select a transition $\tau' = (s'_1, s'_2) \in S$. If $s'_2 \neq t$, then $S = S \cup \{(s'_2, s'_3) \mid (s'_2, s'_3) \in \mathcal{T} \setminus \mathcal{T}'\}$. If $s'_1 \neq s$, then $S = S \cup \{(s'_3, s'_1) \mid (s'_3, s'_1) \in \mathcal{T} \setminus \mathcal{T}'\}$. Finally, $S = S \setminus \{\tau'\}$, $\mathcal{T}' = \mathcal{T}' \cup \{\tau'\}$, and $Q' = Q' \cup \{s'_1, s'_2\}$.
2. If $S \neq \emptyset$, repeat step 1.
3. If $(s \neq s_0 \wedge s_0 \in Q') \vee (t \neq s_f \wedge s_f \in Q')$, then return that a simple component between s and t with respect to τ does not exist. Else, return C.

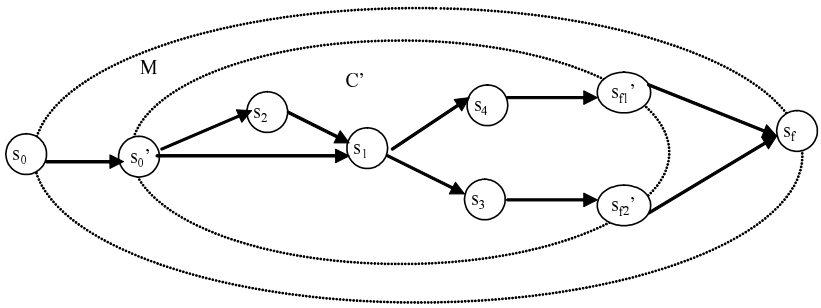

FIGURE 4.11 – Service $M = (Q, s_0, s_f, \mathcal{T})$ having complex component $C' = (Q', s'_0, s'_{f_1}, s'_{f_2}, \mathcal{T}')$.

The above algorithm can be iteratively invoked to compute the set S_C of all simple components of an FSM M. We now give an algorithm to compute a largest simple component of M.

Algorithm to compute a largest simple component C of the given service M.

1. For a pair of components $D, E \in S_C$, if D is a subgraph of E, delete D.
2. For a pair of components $D = (Q', s'_0, s'_f, \mathcal{T}')$ and $E = (Q'', s''_0, s''_f, \mathcal{T}'')$ of S_C, if $s'_0 = s''_0$ and $s'_f = s''_f$, then create a new component $F = (Q' \cup Q'', s'_0, s'_f, \mathcal{T}' \cup \mathcal{T}'')$. If $F \neq M$, then delete D and E from S_C, and add F to S_C.
3. Return the biggest $C \in S_C$.

Using the above algorithm, a largest simple component of given service M can computed in quadratic time. The algorithm can thus be called inductively until there are no more components in the determined component FSM N of a level, and then the hierarchical structure of M has been obtained.

4.1.3.2 Complex Divide and Conquer

In this section, we try to extend the divide and conquer algorithms presented in Section 4.1.1 to complex components, that is, FSMs having more than one initial and/or final states. For example, the service $M = (Q, s_0, s_f, \mathcal{T})$ in Fig. 4.11 has a complex component $C' = (Q', s'_0, s'_{f_1}, s'_{f_2}, \mathcal{T}')$ with two final states s'_{f_1} and s'_{f_2}.

We consider services $M = (Q, s_{0_1}, \cdots, s_{0_{b_1}}, s_{f_1}, \cdots, s_{f_{b_2}}, \mathcal{T})$ having a set P of $b = b_1 + b_2$ port states, consisting of b_1 initial and b_2 final states. Let $\mathcal{T}_\mathcal{O}$ be a subset of transitions of M. We define b^2+1 predicates $P_0(M, \mathcal{T}_O), P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ for all $p_1, p_2 \in P$.

– $P_0(M, \mathcal{T}_\mathcal{O})$ holds if there does not exist more than one invisible path between any two states $s_1 \neq s_2 \in Q$ ($\mathcal{T}_\mathcal{O}$ is a compensable set of transitions).

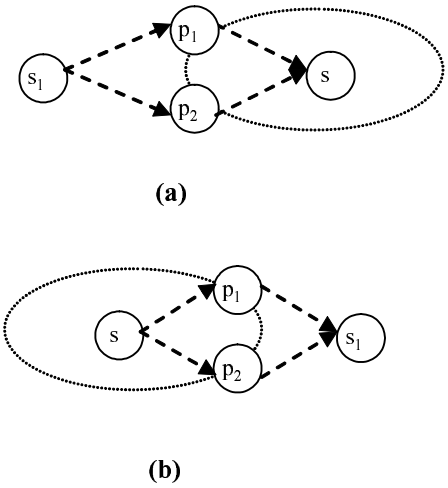

FIGURE 4.12 – Significance of $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ where both p_1 and p_2 are (a) initial and (b) final states.

- if p_1 is an initial and p_2 is a final state : $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ holds if (i) $P_0(M, \mathcal{T}_\mathcal{O})$ holds, and (ii) there does not exist an invisible path from p_1 to p_2. The significance of $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ is analogous to that of predicate $P_1(M, \mathcal{T}_\mathcal{O})$ for simple components.
- if p_1 is a final and p_2 an initial state : $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ holds if (i) $P_0(M, \mathcal{T}_\mathcal{O})$ holds, and (ii) there do not exist states $s_1, s_2 \in Q$, such that (a) there is an invisible path from p_2 to s_2, (b) there is an invisible path from s_1 to p_1, and (c) there is an invisible path from s_1 to s_2. The significance of $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ is analogous to that of predicate $P_{1'}(M, \mathcal{T}_\mathcal{O})$ for simple components.
- if p_1, p_2 are two initial states : $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ holds if (i) $P_0(M, \mathcal{T}_\mathcal{O})$ holds, and (ii) there does not exist a state s in M with invisible paths from both p_1 and p_2 to s. If $P_{p_1,p_2}(C, \mathcal{T}_\mathcal{O})$ does not hold for a component C with visible set of transitions $\mathcal{T}_\mathcal{O}$, then it might be a problem for the compensability of the enclosing M if there exists a state s_1 of M with invisible paths to both p_1 and p_2 as shown in Fig. 4.12(a).
- if p_1, p_2 are two final states, $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$ holds if (i) $P_0(M, \mathcal{T}_\mathcal{O})$ holds, and (ii) there does not exist a state s in M with invisible paths from s to both p_1 and p_2. If $P_{p_1,p_2}(C, \mathcal{T}_\mathcal{O})$ does not hold for a component C with visible set of transitions $\mathcal{T}_\mathcal{O}$, then it might be a problem for the compensability of the enclosing M if there exists a state s_1 of M with invisible paths from both p_1 and p_2 to s_1 as shown in Fig. 4.12(b).

Note that some predicates imply others as $P_{p_1,p_2} = P_{p_2,p_1}$ for two initial or final states, and $P_{p_1,p_2} \implies P_{p_2,p_1}$ for p_1 and p_2 a final and initial state respectively. However, we do not have a total order between predicates. That is, we define $\text{Best}(M, \mathcal{T}_O) : P^2 \rightarrow \{0, 1\}$ as a function with $\text{Best}(M, \mathcal{T}_O)(p_1, p_2) = 1$ iff $P_{p_1,p_2}(M, \mathcal{T}_\mathcal{O})$. We can then extend Proposition 4.1.

Proposition 4.4 *Let C be a component of M, and $\mathcal{T}_1, \mathcal{T}_2$ be subsets of transitions of C, such that Best$(C, \mathcal{T}_1)$ = Best$(C, \mathcal{T}_2)$. Then, for all subset of transitions $\mathcal{T}_\mathcal{O}$ of $M \setminus C$, we have Best$(M, \mathcal{T}_O \cup \mathcal{T}_1)$ = Best$(M, \mathcal{T}_O \cup \mathcal{T}_2)$.*

Proof Let $C = (Q', s'_{0_1}, \cdots, s'_{0_{b_1}}, s'_{f_1}, \cdots, s'_{f_{b_2}}, \mathcal{T}')$. Let us assume that $P_0(M, \mathcal{T}_O \cup \mathcal{T}_2)$ does not hold, that is, there exists a pair of states s_1, s_2 of M with invisible (for $\mathcal{T}_O \cup \mathcal{T}_2$) paths ρ_1, ρ_2 from s_1 to s_2, such that the states traversed by ρ_1 and ρ_2 are disjoint, but for s_1 and s_2. We now show that $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

If both ρ_1 and ρ_2 do not touch C, then $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold. If both ρ_1 and ρ_2 belong to C, then $P_0(C, \mathcal{T}_2)$ does not hold, which means $P_0(C, \mathcal{T}_1)$ does not hold, implying $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

If $s_1, s_2 \notin Q'$, and $\rho = \rho_1$ or ρ_2 passes through C, then there exists an invisible (for $\mathcal{T}_2$) path from an initial p_1 to final p_2 (a subpath of ρ). Given this, $P_{p_1,p_2}(C, \mathcal{T}_2)$ does not hold, which implies that $P_{p_1,p_2}(C, \mathcal{T}_1)$ does not hold. Hence, there exists an invisible (for $\mathcal{T}_1$) path from p_1 to p_2, and an invisible (for $\mathcal{T}_O \cup \mathcal{T}_1$) path ρ' can be constructed from this path and ρ. As such, there are two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 : $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

If $s_1, s_2 \in Q'$, and $\rho = \rho_1$ or ρ_2 passes through C, then there exists an invisible reverse cyclic pattern (for $\mathcal{T}_2$) between a final p_1 and initial state p_2. Given this, $P_{p_1,p_2}(C, \mathcal{T}_2)$ does not hold, which implies that $P_{p_1,p_2}(C, \mathcal{T}_1)$ does not hold. Hence, there exists an invisible (for $\mathcal{T}_1$) reverse cyclic pattern between p_1 and p_2, and an invisible (for $\mathcal{T}_O \cup \mathcal{T}_1$) path ρ' can be constructed from this path and ρ. As such, there are two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 : $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

If $s_1 \in Q'$ and $s_2 \notin Q'$, then there exists invisible (for $\mathcal{T}_2$) paths from s_1 to a pair of final states p_1 and p_2. Given this, $P_{p_1,p_2}(C, \mathcal{T}_2)$ does not hold, which implies that $P_{p_1,p_2}(C, \mathcal{T}_1)$ does not hold. Hence, there exists invisible (for $\mathcal{T}_1$) paths from s_1 to both p_1 and p_2, and two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 can be constructed : $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

If $s_1 \notin Q'$ and $s_2 \in Q'$, then there exists invisible (for $\mathcal{T}_2$) paths from a pair of initial states p_1 and p_2 to s_2. Given this, $P_{p_1,p_2}(C, \mathcal{T}_2)$ does not hold, which implies that $P_{p_1,p_2}(C, \mathcal{T}_1)$ does not hold. Hence, there exists invisible (for $\mathcal{T}_1$) paths from both p_1 and p_2 to s_2, and two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 can be constructed : $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

If $s_1, s_2 \in Q'$, and both ρ_1, ρ_2 pass through C, then there exists invisible (for $\mathcal{T}_2$) paths from an initial p_1 to final p_2, and from another initial p_3 to final p_4. Given this, both $P_{p_1,p_2}(C, \mathcal{T}_2)$ and $P_{p_3,p_4}(C, \mathcal{T}_2)$ do not hold, which implies that $P_{p_1,p_2}(C, \mathcal{T}_1)$ and $P_{p_3,p_4}(C, \mathcal{T}_1)$ do not hold. Hence, there exists invisible (for $\mathcal{T}_1$) paths from p_1 and p_3 to p_2 and p_4 respectively, and two disjoint paths invisible for $\mathcal{T}_O \cup \mathcal{T}_1$ between s_1 and s_2 can be constructed : $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold.

Hence, $P_0(M, \mathcal{T}_O \cup \mathcal{T}_2)$ does not hold $\implies$ $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ does not hold. By symmetry between $\mathcal{T}_1$ and $\mathcal{T}_2$, we have the equivalence : At least $P_0(M, \mathcal{T}_O \cup \mathcal{T}_2)$ holds iff at least $P_0(M, \mathcal{T}_O \cup \mathcal{T}_1)$ holds. Now, $\forall p_1, p_2 \in \{s'_{0_1}, \cdots, s'_{0_{b_1}}, s'_{f_1}, \cdots, s'_{f_{b_2}}\}$, we can enrich M to $F_{p_1,p_2}(M)$ with Best$(M, \mathcal{T}_O)(p_1, p_2) = 1$ iff $P_0(F_{p_1,p_2}(M), \mathcal{T}_O)$ holds.

Applying it to M, we get $\text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_2) = \text{Best}(M, \mathcal{T}_O \cup \mathcal{T}_1)$. □

We can use the above theorem to compute a minimal compensable set of an FSM M having component C as follows :

Algorithm MinimalDecomposition-Complex(M, C) :

1. Compute a minimal set $\mathcal{T}_v(C)$ of transitions of C, for all valuation $v : P^2 \to \{0, 1\}$.
2. Compute a minimal set $\mathcal{T}_0^{\mathcal{T}_v(C),C}(M)$ of transitions of M, for all v.
3. Output a set of smallest size among $\mathcal{T}_0^{\mathcal{T}_v(C),C}(M)$.

It allows to extend the previous result on simple components, for hierarchical services, where each component uses at most p ports :

Theorem 4.3 *Let $p \geq 1$. It is NP-complete in the size of a hierarchical service $H = (M_i)_{i=1}^n$ using at most p ports to compute $MO(\mathcal{H})$, with p fixed. Moreover, it takes time at most $O(2^{3p^2} \sum_{i=1}^n 2^{|M_i|})$.*

Proof Let us consider a hierarchical service $(C_i)_{i=1,\cdots,n}$. The time complexity comes directly from Proposition 4.4. It suffices to compute a minimal set $\mathcal{T}_v(C_i)$ of transitions of C_i, for all valuation $v : P^2 \to \{0, 1\}$. We call V the set of valuations. These actions are performed from bottom of the hierarchy to top. To compute $\mathcal{T}_v(C_i)$ for a component C_i using C_j, C_k, $j, k > i$, we use $\mathcal{T}_{v'}(C_j)$ and $\mathcal{T}_{v''}(C_k)$ which have already been computed. Indeed, it suffices to compute a minimal set $\mathcal{T}_v^{\mathcal{T}_{v'}(C_i) \cup \mathcal{T}_{v''}(C_k), C_j \cup C_k}(M)$ of transitions of M, for all valuations v', v''. Then, it suffices to output a set of minimal size among the $\mathcal{T}_v^{\mathcal{T}_{v'}(C_i) \cup \mathcal{T}_{v''}(C_k), C_j \cup C_k}(M)$ computed. Note that unlike the non-decompositional algorithm, this algorithm can very efficiently parallelize the load up to 2^{3p^2} processors.

Let us turn now to the NP-complete proof. The NP-hard part follows from the NP-hardness in the non-hierarchical case. We give a proof that can be checked in polynomial time, which proves the NP inclusion. A proof that $MO(\mathcal{H}) \leq k$ is given by $n2^{p^2}$ data $f(i, v) = (T, v', v'')$, where T is a subset of transitions of C_i, and v', v'' two valuations. The proof is correct if we have :

- $size(1, v = 0) \leq k$, where *size* is defined inductively as $size(i, v) = |T| + size(j, v') + size(k, v'')$, where $f(i, v) = (T, v', v'')$ and C_i uses components C_j and C_k,
- for all i, v, if $f(i, v) = (T, v', v'')$, then deleting from C_i transitions of T, the resulting graph has property P_v assuming that the two supertransitions have properties $P_{v'}$ and $P_{v''}$ respectively. We say that a graph has properties P_v when $P_{p,q}$ is true whenever $v(p, q) = 1$.

Indeed, if we have such a proof, then we can choose a compensable alphabet for (C_n, v) of size $size(n, v)$, and so on until an alphabet of $size(1, v = 0)$. The correctness of such a proof can be checked in polynomial time.

Assume that $MO(\mathcal{H}) \leq k$. It remains to prove that there exists such a proof. Let $|\mathcal{T}_O| \leq k$ be a compensable alphabet of $\mathcal{H}$. Then, on each component C of $\mathcal{H}$ which corresponds to some H_i, we can compute $\text{Best}(C, \mathcal{T}_O|_C) = v$. We do the same with its (sub)components D, E, $\text{Best}(D, \mathcal{T}_O|_D) = v', \text{Best}(E, \mathcal{T}_O|_E) = v''$. Then, we input in the proof $f(i, v) = (\mathcal{T}_O|_{C\setminus(D\cup E)}, v', v'')$.

The problem is that there are many components C corresponding to H_i, and several can have the same best properties v. We fill the entry with some component C having the smallest $\mathcal{T}_O|_C$. The reason why we can do it is given by Proposition 4.4, that is we can replace any set of transitions by another one, provided we keep the same best properties. Taking the smallest set ensures that the proof will show an alphabet of size at most $|\mathcal{T}_O|$.

If there are some empty initial states, e.g. component H_i for which we never find $\text{Best}(H_i, \mathcal{T}_O|_{H_i}) = (x, y, z)$, then it is irrelevant, and hence useless in the proof. □

A simple rule of thumb to know whether it is worth applying our technique on a component is when it has a large number of transitions with respect to the square of the number of its ports. In order to find components having few ports, one can use heuristics on graph partitioning, for instance [KL70].

Unfortunately, the divide and conquer algorithm by replacement is not scalable to complex components as the addition of even one more initial or final state makes it much more complicated. We discuss the difficulty by considering complex components having two final states (e.g., Fig. 4.11). As the component has two final states, in addition to the predicates P_0, P_1 and $P_{1'}$, we would also need to consider the following predicate :

$P^1(M, \mathcal{T}_\mathcal{O})$ holds for a complex FSM $M = (Q, s_0, s_{f_1}, s_{f_2}, \mathcal{T})$ and visible subset of transitions $\mathcal{T}_\mathcal{O}$ if : (i) $P_0(M, \mathcal{T}_\mathcal{O})$ holds, and (ii) there does not exist a state s of M with invisible paths to both the final states s_{f_1} and s_{f_2}. Let $MO^1(M)$ denote a minimal subset of transitions $\mathcal{T}_\mathcal{O}$, such that $P^1(M, \mathcal{T}_\mathcal{O})$ holds.

Then, we might need to find a replacement for a complex component $C' = (Q', s'_0, s'_{f_1}, s'_{f_2}, \mathcal{T}')$, for which $MO^1(C') - MO(C') > 1$. Now, the effect of the existence of a state $s' \in Q'$ with invisible paths to both s'_{f_1} and s'_{f_2}, on the compensability of the enclosing M, *depends* on the number of distinct pairs of paths in $M_{C'}(C')$ connecting s'_{f_1} and s'_{f_2} to a common state s of M. That is, if $MO^1(C') - MO(C') = m$, then $MO^1(D') - MO(D')$ also needs to be equal to m, *meaning that D' cannot have a constant number of states* (which implies that such a replacement may not always lead to a significant reduction in the size of M). For example, let us consider the FSM M having complex component $C' = (Q', s_i, s_j, s_k, \mathcal{T}')$ in Fig. 4.13(a). Now, for C', $MO(C') = 1$ and $MO^1(C') - MO(C') = 2$. Given this, if we replace C' by a $D' = (Q', s_x, s_y, s_z, \mathcal{T}')$ as shown in Fig. 4.13(b), for which $MO^1(D') - MO(D') = 1$. Then, $MO(M) = 4 \neq MO(M_{C'}(D')) + MO(C') - MO(D') = 2 + 1 - 0 = 3$.

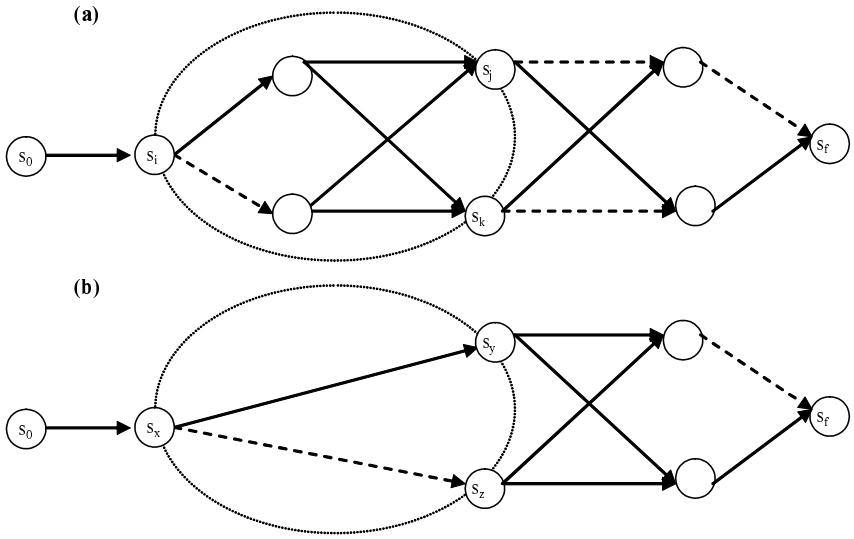

FIGURE 4.13 – Sample (a) $M_{C'}(C')$, for which $MO^1(C') - MO(C') > 1$, and (b) $M_{C'}(D')$ (the dashed arrows represent a minimal compensable set of M and $M_{C'}(D')$).

4.1.4 Experimental Evaluation

We tested our divide and conquer algorithms on hierarchical graphs. First, we choose a number (between one and nine) of base subcomponents in the graph. Then, we generate each of them randomly using the Synthetic DAG generation tool [dag]. We then generate inductively a hierarchical graph having these base subcomponents randomly using the same tool, by assigning two edges to these components. There is no reuse of components. For each value, we generate each hierarchical graph and each base subcomponent five times to compute the mean values (because of variation in graph size, runtime and compensable size). We then unfold the hierarchical graphs as (flat) graphs, whose size is linear in the number of base components. We then run both a brute force algorithm and our hierarchical algorithms on these graphs. We do not input the hierarchical structure of the graph, instead it uses the polynomial time algorithm presented in Section 4.1.3.1 to determine the hierarchical structure.

Fig. 4.14 shows the times (in logarithmic scale) needed to compute a minimal compensable set using brute force and our divide and conquer algorithm based on best property (Proposition 4.1) and replacement (Theorem 4.2) with respect to the number of edges (which is linear with respect to the number of base subcomponents). In Fig. 4.14, Decomposition 1 and Decomposition 2 refer to the divide and conquer algorithms by best property and replacement, respectively. The best property algorithm is clearly much faster than the brute force algorithm. While the brute force is exponential in the number of edges, already timing out at a little over 40 edges, the best property algorithm on an average takes $17s$ for 106 edges. However, it is still not polynomial time with respect to the number of base subcomponents/number of edges (on an average,

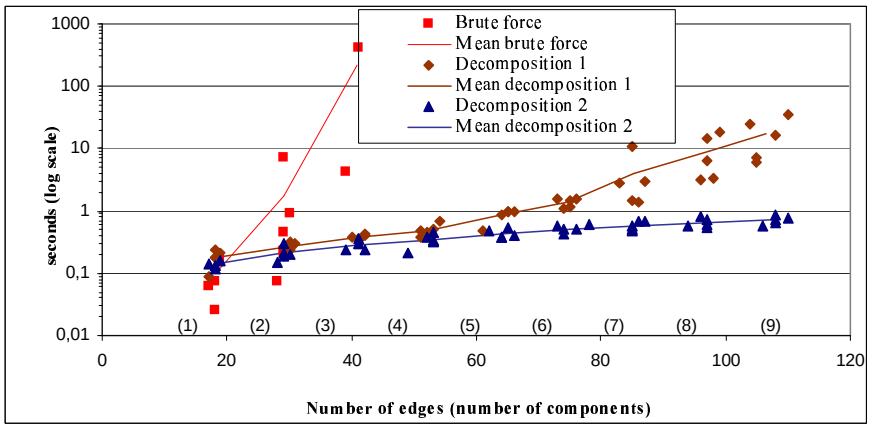

FIGURE 4.14 – Execution time vs. Number of subcomponents/edges of the brute force, decomposition by best property and replacement algorithms.

0.17s for 18 edges and 17.6 for 106 edges). Now, let us consider the test timings of our divide and conquer algorithm by replacement. The replacement algorithm is indeed linear time with respect to the number of base subcomponents/number of edges (on an average, 0.14s for 18 edges and 0.73s for 108 edges). For one subcomponent, the overhead of our method makes the decomposition slightly worse than the brute force method.

Fig. 4.15 shows the percentage of edges needed to be visible among all the edges. As expected, both the brute force and our divide and conquer algorithms answer the same number on the same graphs but there is a large variation among graphs, from one out of 4 edges needing to be logged to one out of 15 edges. The mean value seems to tend to one out of 6 edges. This implies that logging only (execution details of) the visible transitions can lead to significant space savings.

4.2 Bottom-up Hierarchical Services

In this section, we tackle hierarchical services composed in a bottom-up fashion. Recall that composite services formed in a bottom-up fashion are described as a product of their components' FSMs (see Definition 1.1). As in Section 4.1, here also we would like to use knowledge of minimal compensable sets of the components to determine a minimal compensable set of the composite service. The visibility (logging) of each service is done locally. Hence, our divide and conquer algorithms cannot be applied on the product since choosing to make a transition visible in a component of the product might force it to be visible in another as well (if the same component is reused).

We first consider services having no interaction between them. We explain later how to deal with interacting services (having a non-interacting component). For non-interacting component services M, N, they do not have any shared transitions or need for synchronization, and hence $Sync_{M,N} = \emptyset$.

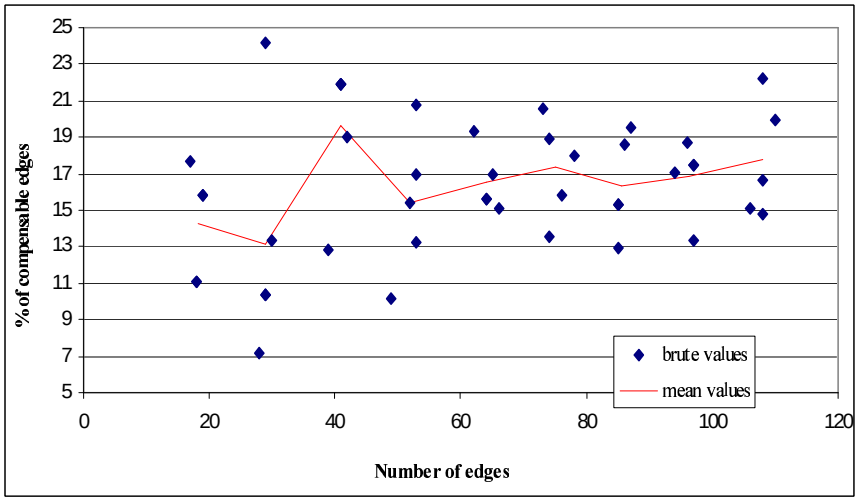

FIGURE 4.15 – Percentage of edges needed to be visible.

Definition 4.2 (Commutativity) *Given services* $M = (Q_1, s_1, s_2, \mathcal{T}_1)$ *and* $N = (Q_2, s_3, s_4, \mathcal{T}_2)$*, two consecutive transitions* τ_i *and* τ_{i+1} *of a path* ρ *of* $M \times N$ *can commute if* $\tau_i \in \mathcal{T}_1$ *and* $\tau_{i+1} \in \mathcal{T}_2$*, or vice versa.*

Let us consider a product $M \times N$ of non-interacting FSMs M and N as shown in Fig. 4.16. Now, $\mathcal{T}_{OM} = \{e\}$ and $\mathcal{T}_{ON} = \{f\}$ are minimal compensable sets of M and N respectively. Fig. 4.17(c) shows the $M \times N$ after the visible transitions e and f have been deleted. Note that while there still exists greater than one path between states in Fig. 4.17(c), we contend that they are equivalent from a compensation perspective as "strict" execution sequence detection is not required for compensation. We formalize this in the sequel.

Definition 4.3 (Equivalence) *Two paths* $\rho_1 \neq \rho_2$ *of* $M \times N$ *are equivalent* $\rho_1 \equiv \rho_2$*, if* $\rho_1 = \rho_2$ *after a finite sequence of commutations on* ρ_1 *(or* ρ_2*).*

Again, with reference to the $M \times N$ in Fig. 4.17(c), the invisible paths acd, cad, cda between states (s_1, s_4) and (s_2, s_6) are equivalent. The paths are equivalent from a compensability perspective as :

- they consist of the same set of distinct transitions $\{a, c, d\}$ to be compensated, and
- the execution order of transitions with respect to each component FSM is maintained (c always occurs before d in all three invisible paths).

Intuitively, if something was executed concurrently, then it can also be compensated concurrently. To integrate the above notion of equivalent paths into the compensability definition, we extend Definition 3.17 as follows :

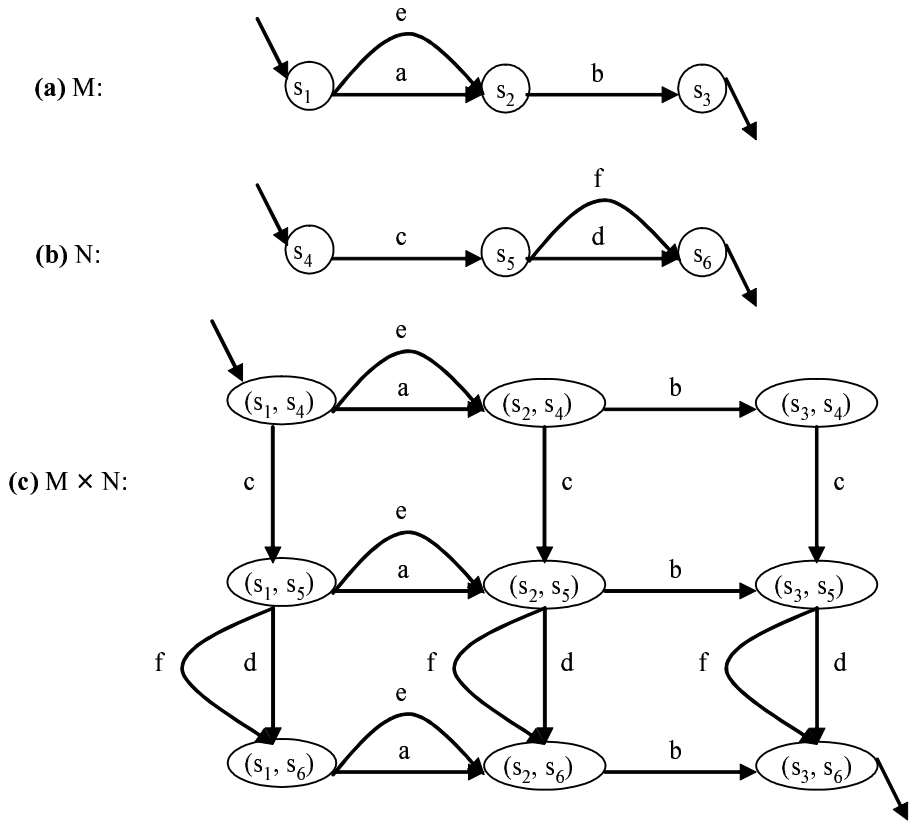

FIGURE 4.16 – Product $M \times N$ of non-interacting FSMs M and N.

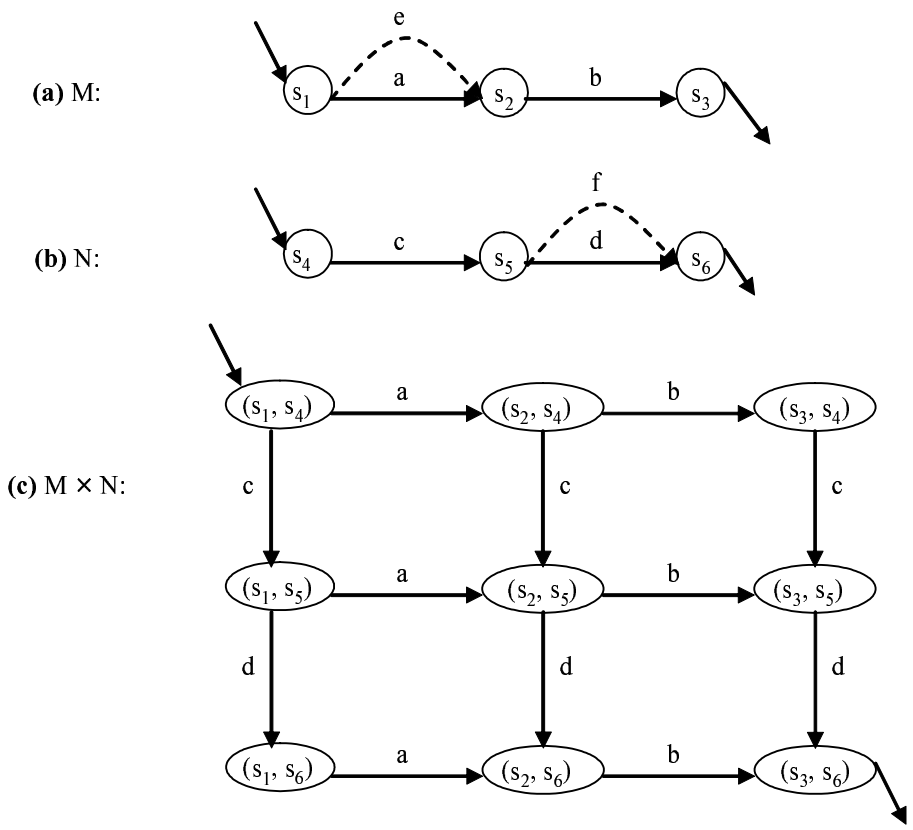

FIGURE 4.17 – Product $M \times N$ of non-interacting FSMs M and N, after deleting the visible transitions (dashed arrows represent a minimal compensable set of M and N).

Definition 4.4 (Compensability for Product FSMs) *Given FSMs M, N, their product $M \times N$ is compensable with $\mathcal{T}_1$ in M and $\mathcal{T}_2$ in N visible, if for all paths ρ_1, ρ_2 of $M \times N$, such that $Obs^{last}_{\mathcal{T}_1 \cup \mathcal{T}_2}(\rho_1) = Obs^{last}_{\mathcal{T}_1 \cup \mathcal{T}_2}(\rho_2)$, we have $\rho_1 \equiv \rho_2$.*

Given this, compensation can be performed using any of the equivalent (reversed) execution sequences, this will result in the same consistent state. We then have the following desirable property :

Proposition 4.5 *For a pair of non-interacting FSMs M, N, and any of their respective minimal compensable sets $\mathcal{T}_{OM}$ and $\mathcal{T}_{ON}$, $\mathcal{T}_{OM} \cup \mathcal{T}_{ON}$ is a minimal compensable set of $M \times N$.*

Finally, we consider the more general case where M and N interact. Clearly, Proposition 4.5 cannot be extended to interacting systems. However, we can still state the following desirable property if one of the services M has a component C that does not interact with N.

Proposition 4.6 *Given interacting FSMs $M = (Q_1, s_1, s_2, \mathcal{T}_1)$ and $N = (Q_2, s_3, s_4, \mathcal{T}_2)$, let M have a component $C = (Q_C, s_0, s_f, \mathcal{T}_C)$, such that $\mathcal{T}_C \cap \mathcal{T}_2 = \emptyset$. Further, let $\mathcal{T}_{OC}$ and $\mathcal{T}_{O1}$ be minimal compensable sets of C and $M \setminus C \times N$ respectively. Then, $\mathcal{T}_{OC} \cup \mathcal{T}_{O1}$ is a minimal compensable set of $M \times N$.*

For example, let us consider the FSMs $M = (Q_1, s_1, s_5, \mathcal{T}_1)$, $N = (Q_2, s_6, s_9, \mathcal{T}_2)$ and their product $M \times N$ in Fig. 1.13 (reproduced in Fig. 4.18). Now, the FSM M has a simple component $C = (Q_C, s_1, s_2, \mathcal{T}_C)$ (shown by the dashed oval in Fig. 4.19), such that $\mathcal{T}_C \cap \mathcal{T}_2 = \emptyset$. Then, let us consider minimal compensable sets $\mathcal{T}_{OC} = \{a\}$ and $\mathcal{T}_{O1} = \{e\}$ of C and $M \setminus C \times N$ respectively. Given this, $\mathcal{T}_{OC} \cup \mathcal{T}_{O1}$ is a minimal compensable set of $M \times N$. Fig. 4.19(c) shows $M \times N$ after the visible transitions have been deleted. As before, while there still exists greater than one path between states in $M \times N$, they are equivalent.

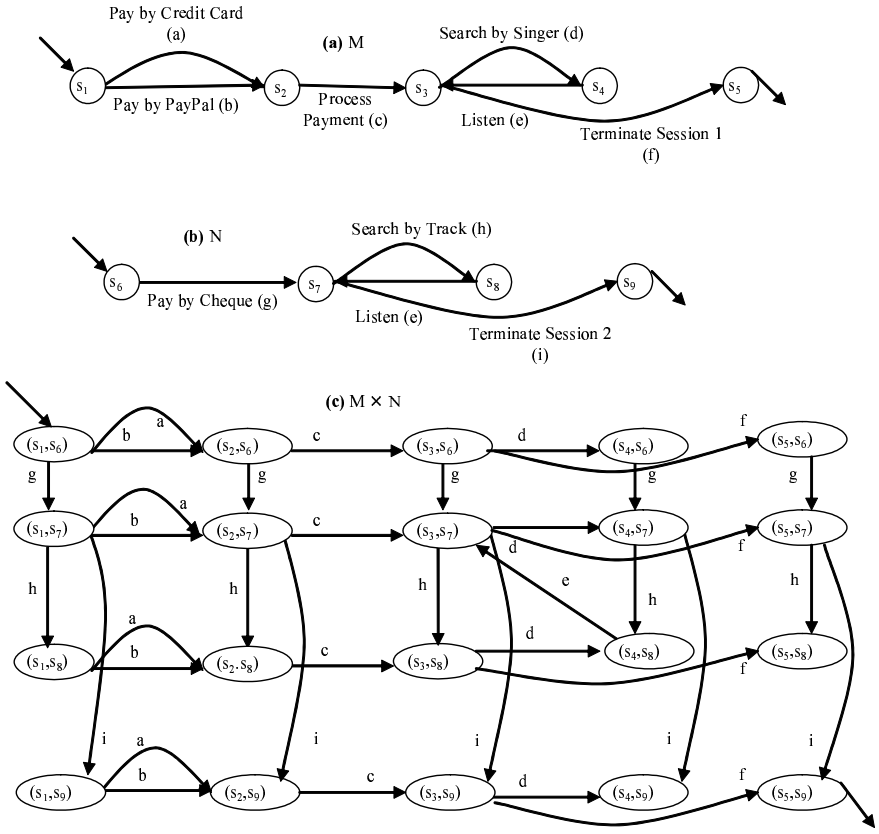

FIGURE 4.18 – Interacting FSMs (a) M, (b) N, and (c) their product $M \times N$.

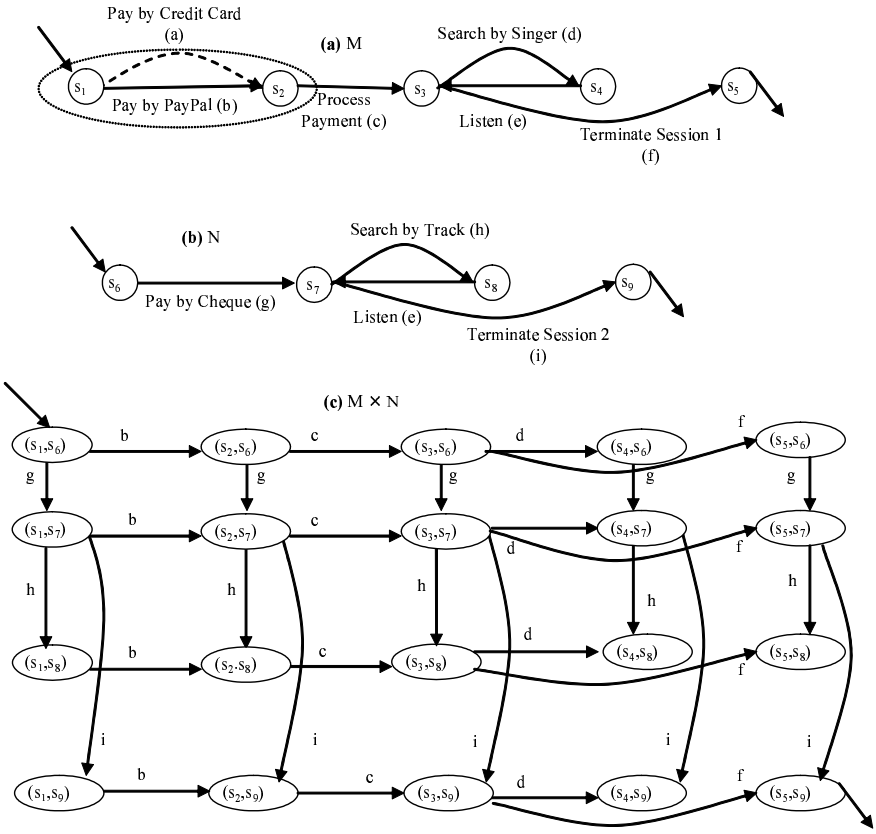

FIGURE 4.19 – Fig. 4.18 after the visible transitions have been determined and deleted.

Chapter 5

Approximation

In this chapter, we study heuristics to solve the minimal compensability problem. The need for heuristics mainly arises from the fact that the problem is NP-complete even for very restricted graphs. The divide and conquer strategy proposed in Chapter 4 is not always applicable as all complex services cannot be decomposed hierarchically. Also, in cases where the cost of logging is not prohibitive, logging a few more transitions is generally acceptable to the computationally expensive process of computing the exact minimum. We propose two heuristics in the sequel, and experimentally analyze their complexity and closeness to the absolute minimum.

We present our first heuristic in Section 5.1. The underlying idea is to log the transitions required to *(positively) discriminate* the occurrence of a path from the remaining paths. Note that the occurrence of a visible transition also implies non-occurrence of some paths, that is, it also allows to *distinguish* several paths. Thus, our second idea, presented in Section 5.3, is to optimize a compensable set, such that each visible transition implies the non-occurrence of a maximum number of paths (not using that transition). We test both heuristics experimentally in Section 5.4. The results shows that the distinguishing algorithm gives result from 1 (at least as good, which we prove theoretically) to 10 times smaller than the discriminating algorithm, with an average of 1.9 times smaller.

5.1 (Positively) Discriminating Algorithm

Our first idea is to use visibility as a discriminator. A path can be (positively) discriminated from other paths if, for all (intermediate) states it passes through, we know the outgoing transition which was executed. Clearly, for intermediate states having one outgoing transition, the choice is unique and obvious. For intermediate states having $n > 1$ outgoing transitions, it is sufficient if $n - 1$ are visible. Below, we present the polynomial time discriminating algorithm :

Discriminating Algorithm.
Input. FSM $M = (S, s_0, s_f, \mathcal{T})$.

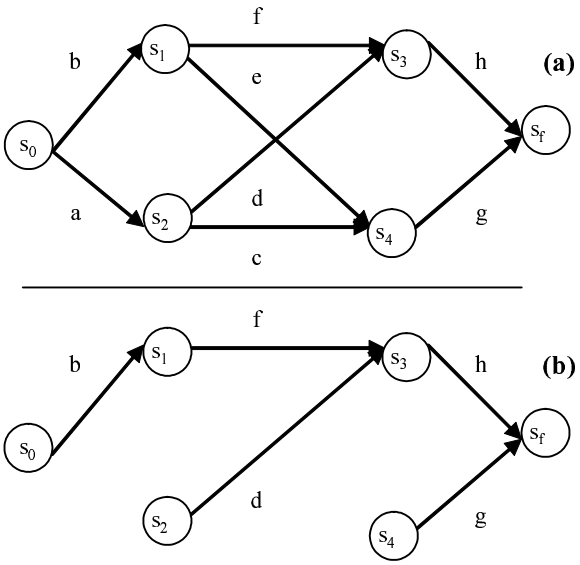

FIGURE 5.1 – (a) Sample FSM M and (b) after deleting the transitions in $\mathcal{T}_O = \{a, c, e\}$ from M.

Output. A compensable set $\mathcal{T}_O \subseteq \mathcal{T}$.
Initialization. $\mathcal{T}_O = \epsilon$.
for each state $s \in S$
  if $(s^* = \{\tau_1 \cdots \tau_n\}, n > 1)$
  then $\mathcal{T}_O = \mathcal{T}_O \cup \{\tau_1 \cdots \tau_{n-1}\}$.
  endif
endfor

For example, let us consider the FSM M in Fig. 5.1(a). An output $\mathcal{T}_\mathcal{O}$ of the discriminating algorithm with M as input would be $\{a, c, e\}$. Fig. 5.1(b) shows M after the transitions in $\mathcal{T}_O$ have been deleted from M, and is indeed a compensable set of M.

Proposition 5.1 *For a given FSM $M = (S, s_0, s_f, \mathcal{T})$, the output $\mathcal{T}_O$ of the discriminating algorithm is a compensable set of M. And, $|\mathcal{T}_O| = |\mathcal{T}| - |S| + 1$.*

Proof The compensability of $\mathcal{T}_O$ follows from the fact that at termination of the algorithm, all states (except the final) of M are left with exactly one outgoing transition. Then, $|\mathcal{T}| - |\mathcal{T}_O| = |S| - 1$, and the compensable size $|\mathcal{T}_O| = |\mathcal{T}| - |S| + 1$. □

While the discriminating algorithm is simple, it is not so easy to better the compensable size $|\mathcal{T}|-|S|+1$ computed by it. To illustrate, we present a much smarter algorithm in the next section (taken from [Gaz07, BGG08]), but which still gives compensable size $|\mathcal{T}| - |S| + 1$.

	a	b	c	d	e	f	g	h
s_0	-1	-1	0	0	0	0	0	0
s_1	0	1	0	0	-1	-1	0	0
s_2	1	0	-1	-1	0	0	0	0
s_3	0	0	0	0	1	1	0	-1
s_4	0	0	1	1	0	0	-1	0
s_f	0	0	0	0	0	0	1	1
a	1	0	0	0	0	0	0	0
c	0	0	1	0	0	0	0	0
e	0	0	0	0	1	0	0	0

FIGURE 5.2 – Matrix of the FSM M in Fig. 5.1(a) and $\mathcal{T}_O = \{a, c, e\}$.

5.2 Matrix Algorithm

Here, we try to characterize the selected compensable set of transitions algebraically. Let us define an extension of the classical incidence matrix of a directed acyclic graph (and of a FSM seen as a directed graph), which encodes a directed graph in a matrix. Its first rows express the directed graph, and its last rows express the visibility from a compensable set of transitions $\mathcal{T}_O$. For convenience, we index the matrix by states and transitions rather than numbers.

Definition 5.1 (extended incidence matrix) *Let $M = (Q, s_0, s_f, \mathcal{T})$ be an FSM and $\mathcal{T}_O \subseteq \mathcal{T}$ be a subset of transitions of M. The* incidence matrix *of M, relative to $\mathcal{T}_O$ is a matrix $A_{M,\mathcal{T}_O}$ of size $(|Q| + |\mathcal{T}_O|) \times |\mathcal{T}|$ defined as follows :*

For every $(s, \tau) \in Q \times \mathcal{T}$:

- *$A_{M,\mathcal{T}_O}[s, \tau] = 1$ if τ ends in s ;*
- *$A_{M,\mathcal{T}_O}[s, \tau] = -1$ if τ starts from s ;*
- *$A_{M,\mathcal{T}_O}[s, \tau] = 0$, otherwise.*

For every $(\tau, \tau') \in \mathcal{T}_O \times \mathcal{T}$:

- *$A_{M,\mathcal{T}_O}[\tau, \tau] = 1$;*
- *$A_{M,\mathcal{T}_O}[\tau, \tau'] = 0$, if $\tau' \neq \tau$.*

The first $|Q|$ rows of this matrix correspond exactly to the classical incidence matrix. We just append to it a (almost identity) $|\mathcal{T}_O| \times |\mathcal{T}|$ matrix in order to obtain an extended incidence matrix. For example, the matrix corresponding to the FSM M in Fig. 5.1(a) and $\mathcal{T}_O = \{a, c, e\}$, is shown in Fig. 5.2.

Furthermore, as we translated an FSM $M = (Q, s_0, s_f, \mathcal{T})$ into an algebraic object $A_{M,\mathcal{T}_O}$, we do now transform a path ρ of $\mathcal{P}(M)$ into an algebraic object. Let us denote by $\chi(\rho)$ the vector in $\{0, 1\}^{|\mathcal{T}|}$, such that $\chi(\rho)[\tau] = 1$ if τ is fired in ρ, otherwise $\chi(\rho)[\tau] = 0$. Note that $\chi(\rho)$ characterizes any path ρ of an acyclic graph (the order of transitions can be recovered unambiguously). We now define the observation $V_{M,\mathcal{T}_O}(X)$ associated with a vector X in $\{0, 1\}^{|\mathcal{T}|}$ representing a path.

Definition 5.2 (visibility vector) *The* visibility *of a vector X in $\{0,1\}^{|\mathcal{T}|}$, relative to $\mathcal{T}_O$ is a vector $V_{M,\mathcal{T}_O}(X)$ of size $|Q|+|\mathcal{T}_O|$, such that for every $s \in Q$*
- *$V_{M,\mathcal{T}_O}(X)[s] = -1$ if X starts from s ;*
- *$V_{M,\mathcal{T}_O}(X)[s] = 1$ if X ends in s ;*
- *otherwise $V_{M,\mathcal{T}_O}(X)[i] = 0$.*

Moreover, for every $\tau \in \mathcal{T}_O$, we have :
- *$V_{M,\mathcal{T}_O}(X)[\tau] = 1$ if $X[\tau] = 1$;*
- *otherwise $V_{M,\mathcal{T}_O}(X)[\tau] = 0$.*

Clearly, $\mathcal{T}_O$ is a compensable set of transitions iff there *does not* exist $X \neq Y$ with $V_{M,\mathcal{T}_O}(X) = V_{M,\mathcal{T}_O}(Y)$. We use now the algebraic characterization.

Proposition 5.2 *Let $X, Y \in \{0,1\}^{|\mathcal{T}|}$. Then we have :*

$$V_{M,\mathcal{T}_O}(X) = V_{M,\mathcal{T}_O}(Y)$$
$$\Leftrightarrow$$
$$A_{M,\mathcal{T}_O} \cdot Y = V_{M,\mathcal{T}_O}(X) = A_{M,\mathcal{T}_O} \cdot X$$

Using proposition 5.2, we obtain that *$A_{M,\mathcal{T}_O}$ is injective implies that $\mathcal{T}_O$ is a compensable set of transitions*. It can be shown that the discriminating algorithm gives a minimal set for which *$A_{M,\mathcal{T}_O}$ is injective*, thus giving us the algebraic characterization we were looking for.

Now, it is well known that a matrix is injective iff its kernel has dimension 0. In our case, it is also well known [GR01] that the kernel of the incidence matrix $A_{M,\emptyset}$ corresponds exactly to the so-called cycle space of M. Moreover, the dimension of the cycle space (also known as the cyclomatic number) of a graph with n vertices, m edges and K connected components (considering edges are unoriented) is exactly $m-n+K$. That is, it suffices to log $m-n+K$ edges to have an injective matrix (and equivalently a compensable set of transitions).

The discriminating and matrix algorithms are very fast to give an approximation of the compensable size, since it suffices to count $|\mathcal{T}| - |S| + 1$ of the input FSM $M = (S, s_0, s_f, \mathcal{T})$, which is an immediate number to compute. For instance, in the example of Fig. 5.1, $|\mathcal{T}| - |S| + 1 = 8 - 6 + 1 = 3$. However, note that the minimal compensable size of M is 2. Our experimental evaluations (Section 5.4) also reveal that the discriminating/matrix algorithm leads to 20% more transitions being logged in the worst case, 6% in mean value. In the next section, we present an alternate heuristic which is always better (though slower).

5.3 Distinguishing Algorithm

Clearly, thinking in terms of positive discrimination is not always good. For example, for the FSM M in Fig. 5.1, observing any of the transitions in $\{c, d, e, f\}$ leads to a compensable size greater than 2. Thus, we need an alternate strategy.

We first remark that the deletion of a transition from the given FSM M, can lead to M having more than one initial and/or final state. We call initial (final) state a state which has no incoming (outgoing) transitions. For example, in Fig. 5.1(b), s_2 becomes an initial state after the deletion of transition a. The strategy we propose in this section is to select transitions which, for a pair of initial and final states $s_1 \neq s_2$ connected by a path using transition τ, maximize the number of paths not using τ but connecting s_1 to s_2. That is, we now want to maximize the number of paths distinguished by a visible transition. For example, let us consider the FSM M in Fig. 5.3(a). $\{b, h, i, j, p\}$ is a minimal compensable set of M, and Fig. 5.3(b) shows M after those transitions have been deleted. Thus, at an intermediate stage Fig. 5.3(c), we would like to choose one of g, h, i, j, as selecting any of the remaining inadvertently leads to a compensable size 6. Now, let us apply the above heuristic. With respect to say i and pair of initial and final states s_0, s_f, there are three paths connecting s_0 to s_f without using i. The three paths are $adjmo, acgko$ and $achmo$. However, with respect to say k and s_0, s_f, there are only two paths connecting s_0 to s_f without using k : $adjmo$ and $achmo$. Thus, our heuristic would lead to choosing i over k, which is what we want. However, computing exactly that number of paths is really inefficient. We thus present an efficient but slightly less accurate version is the sequel.

We want to choose a transition τ which maximizes the number of initial (final) states from (to) which the following condition holds : For an initial (final) state s_1, there exists a final (initial) state s_2 with at least two paths from s_1 to s_2 (s_2 to s_1), one using τ and one not using τ. Clearly, if a transition τ does not figure in one of two such paths with respect to any initial or final state, then it is useless for compensability.

We present the algorithm *Count* based on Depth First Search (DFS). Starting DFS from an initial state s_1, if the target state $t.dest$ of the current transition t has been previously explored, then we know that there exists another path not using t from s_1 to $t.dest$. That is, we have two paths connecting s_1 to a final state reachable from $t.dest$ (say s_2), one using t and one not using t. We keep a counter $t.count$ associated with each transition t, and increase it as soon as we reach an already explored state s via t. Furthermore, for each such hit, we also need to increment the counter of the transition $s.first$ via which s was first reached. Note that we only need to increment the counter of $s.first$ once for each initial state s_1, and not each time s is reached via a different path from s_1. To accommodate this, we use the flag $s.firstflag$, which indicates whether the counter of transition $s.first$ has been incremented with respect to an initial state. We cannot increment the counter of $s.first$ initially, because then we do not know if there exists another path from s_1 to s not using $s.first$. The algorithm keeps a stack K, a hash table H of states which have already been explored by the search, and each transition t has an associated flag to tag it as explored or unexplored. $K.head$ designates the head of the stack and push/pop are standard stack operations.

Algorithm Count $M = (S, s_0, s_f, \mathcal{T})$.
Create Hash table H and $\forall t \in \mathcal{T}, t.count := 0$.
for each initial state s_0 of M
  Initialize H to empty, K to s_0.

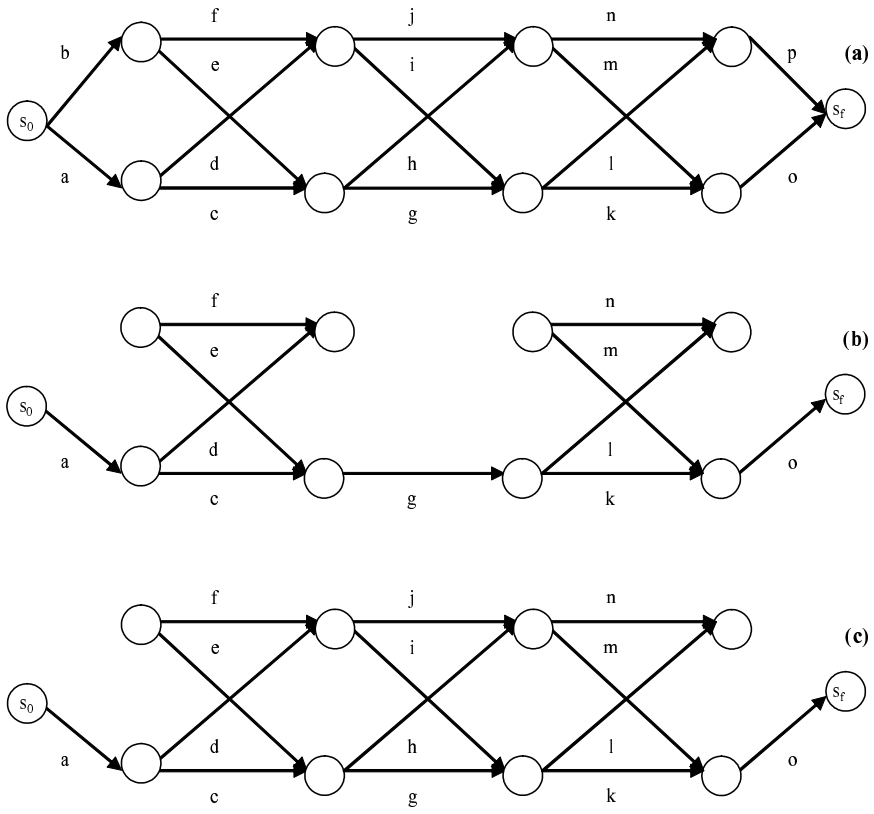

FIGURE 5.3 – FSM (a), after after deleting a minimal compensable set (b), and an intermediate stage (c).

```
Set all transition tags to unexplored and ∀s ∈ S, s.firstflag := false.
Run DFS from s_0 :
while K is nonempty do
  while there is an unexplored transition t from state K.head do
    Tag t as explored.
    if t.dest ∈ H
    then increment t.count, and
      if t.dest.firstflag = false
      then increment t.dest.first.count and set t.dest.firstflag := true
      endif
    else set t.dest.first := t, insert t.dest into H and push t.dest on K.
    endif
  endwhile
  pop K
 endwhile
endfor
```

Obviously, an algorithm *CountBack* can be similarly devised, running from all final states and traversing transitions backwards. Now, our distinguishing algorithm proceeds as follows :

```
Algorithm Distinguish FSM M = (S, s_0, s_f, T)
Set T_O := ∅.
loop
   Set all transition counters to 0.
   Run at random Count or CountBack.
   if all transition counters are 0, then return T_O.
   else select one transition t with maximal counter.
        Add t to T_O.
        Delete t from T.
   endif
endloop
```

Proposition 5.3 *For a service $M = (S, s_0, s_f, \mathcal{T})$, the distinguishing algorithm returns a compensable set $\mathcal{T}_O$ of transitions. Moreover, its size is always $|\mathcal{T}_O| \leq (|\mathcal{T}| - |S| + 1)$.*

Proof The compensability of $\mathcal{T}_O$ follows from the termination condition. Clearly, if all transition counters are 0 at the end of a *Count/Countback*, then there does not remain greater than one path between any pair of states.

The cardinality of $\mathcal{T}_O$ follows from the fact that M remains *connected* at any stage of the algorithm. Let us consider the transitions deleted from $\mathcal{T}$ (added to $\mathcal{T}_O$) during the algorithm. Each such transition t is either the incoming of a state having indegree more than one (by *Count*) or outgoing of a state having outdegree more than one (by

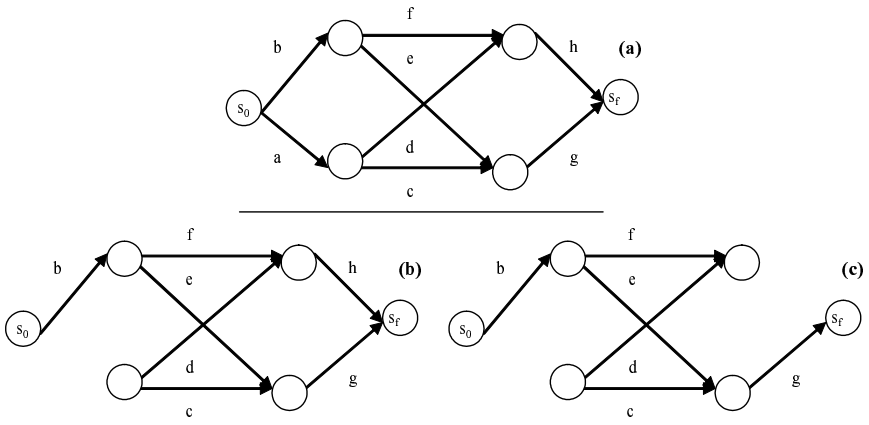

FIGURE 5.4 – (a) Sample FSM M, after the deletion of transitions (b) a and (c) h from M.

CountBack). As such, even after the deletion of t from $\mathcal{T}$, there are no disconnected subgraphs, and there still exists a path from the source state of t to a final state (or from an initial state to the target state of t). Given this, $|\mathcal{T}_O| \leq (|\mathcal{T}| - |S| + 1)$ follows from the property that any undirected connected graph has at least $|S| - 1$ edges.

□

For example, let us consider a run of the distinguishing algorithm on the FSM M in Fig. 5.4(a). Initially, let us assume that *CountBack* is run. Then, the counter values of the transitions a, b, c, d, e, f, g, h at the end of *CountBack* are $1, 1, 1, 1, 1, 1, 0, 0$ respectively. Given this, let a is chosen and added to $\mathcal{T}_O$. Fig. 5.4(b) shows M after the deletion of a. Note that M now has two initial states. Further, let *Count* is run for this iteration. Then, the counter values of the transitions b, c, d, e, f, g, h at the end of *Count* are $0, 0, 0, 0, 0, 2, 2$ respectively, leading to the addition of say h to $\mathcal{T}_O$. Fig. 5.4(c) shows M after the deletion of a and h. At this stage, irrespective of whether *Count* or *CountBack* is run, the counter values of all remaining transitions are 0, terminating the algorithm. Indeed, $\mathcal{T}_O = \{a, h\}$ is a compensable set of M.

Note that there is a certain amount of randomness inherent in the algorithm. For example, recall that after the initial run of *CountBack*, the transitions c, d, e, f had the same counter value as a. And, if any of them had been chosen, then we would have ended up with $MO(M) > 2$. Also, if *CountBack* was again run for the second iteration, then the counter values of the transitions b, c, d, e, f, g, h would have been $0, 1, 1, 1, 1, 0, 0$ respectively, leading to the selection of one of c, d, e, f, again leading to $MO(M) > 2$. To offset this randomness, we took the minimum of ten runs during our experimental evaluation (discussed in the next section).

5.4 Experimental Evaluation

We have some theoretical clues about how our algorithms fair against each other, and how close they can approximate the absolute minimal compensable size. Because our discriminating and distinguishing algorithms are polynomial time, we know that there are FSMs on which they give an answer far away from the optimum [RKK06]. The question is how far they are, and how often it happens. The second fact is that the distinguishing algorithm gives a set never bigger than the one given by the discriminating algorithm. The question then is : is it better, and if yes, by how much and how often is it much better.

The first question is difficult to answer accurately, since obtaining the absolute minimal compensable size is intractable. One solution could be to look at small enough FSMs to get the values, but the problem is a variation of one visible transition having a big impact percentage wise in small sets, so the results would not be very meaningful. Instead, we focus on particular non-trivial FSMs, namely hierarchical FSMs. For a hierarchical FSM H, we can use the polynomial time divide and conquer algorithm presented in Section 4.1 to compute the minimal compensable size of $\mathcal{H}$ based on the compensable sizes of its components. This allows us to compute the absolute minimal compensable size of large hierarchical FSMs, as long as the components are small enough. We thus tested our two heuristics plus the absolute minimal algorithm on hierarchical FSMs, giving the results table on the left part of Fig. 5.5, analyzing the data in the next section. Furthermore, to confirm or infirm the conclusions we draw on hierarchical FSMs, we also tested our two heuristics on general FSMs, whose results are given by the table on the right part of Fig. 5.5.

5.4.1 Hierarchical FSMs

We generate hierarchical FSMs randomly, using the same methodology as described in Section 4.1.4 (using the *Synthetic DAG Generation Tool*). On these FSMs, we run the absolute minimal algorithm, as well as the discriminating and distinguishing algorithms. Fig. 5.6 shows the results we obtained, according to the number of transitions of the flat FSM.

The graph confirms that the discriminating algorithm gives worse results than the distinguishing algorithm, which gives worse results than the absolute minimal algorithm, but the difference does not seem very significant. Let us analyze more precisely the percentage difference between the absolute minimal compensable size and the compensable sizes given by the discriminating and distinguishing algorithms, on the same data (see Fig. 5.7). It seems that the distinguishing algorithm comes much closer to the absolute minimum, from 0% to 6%, 2% in mean value. The discriminating algorithm is sometimes as good as the absolute minimum, sometimes much worse (20% more transitions need to be logged), 6% in mean value, that is 3 times more than the distinguishing algorithm.

Last, we compare the percentage of transitions logged by the different algorithms (see Fig. 5.8). As mentioned earlier, this number is quite close for the 3 algorithms, ranging from 17% to 33%. In mean values, the algorithms needs to log 20%, 21% and

No. of edges	Compensable size Absolute	Discr.	Disting.
42	9	9	9
58	14	14	14
75	18	19	18
103	20	24	20
102	32	33	32
123	31	33	32
146	33	34	34
146	42	42	42
178	36	37	36
185	38	38	38
223	43	47	44
221	53	57	55
241	57	62	58
273	51	57	53
280	72	75	72
294	74	76	75
325	69	78	73
326	75	79	76
355	74	81	77
345	88	91	88
382	84	90	85
387	92	96	94
410	81	87	83
445	89	97	90
448	100	106	104
460	101	108	105
484	99	107	103
489	104	111	107
540	106	122	112
550	93	101	95
570	108	118	111
605	108	124	111
618	121	136	125
634	120	131	124
631	115	121	118
657	131	141	134
672	134	146	139
699	126	141	131
704	142	156	150

No. of Edges	Compensable size Discr.	Disting.
97	41	11
127	55	13
85	29	19
115	47	20
137	52	23
126	43	27
264	120	20
312	143	22
173	63	36
201	77	52
431	197	28
452	205	40
103	22	17
200	72	39
133	36	25
356	147	74
301	131	45
114	27	19
785	367	40
169	45	30
98	16	16
101	17	17
175	49	34
987	462	72
132	29	24
490	203	113
464	188	110
158	37	27
115	17	17
620	268	140
121	17	17
631	268	141
1013	468	90
128	18	18
165	53	37
114	56	56
214	58	36
137	19	19
508	223	86
228	53	39
747	315	178
222	55	38
947	416	217
790	340	219
161	23	23
1023	447	257
225	46	34
997	431	230
311	89	60
253	58	40
923	419	106
321	90	63
182	24	24
1110	480	276

FIGURE 5.5 – Test results data on hierarchical (left) and general (right) FSMs.

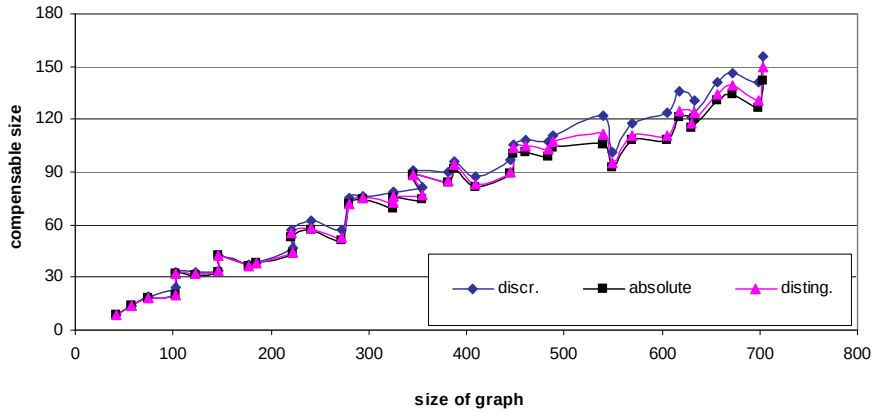

FIGURE 5.6 – Compensable size vs. number of transitions, over 40 randomly generated hierarchical FSMs.

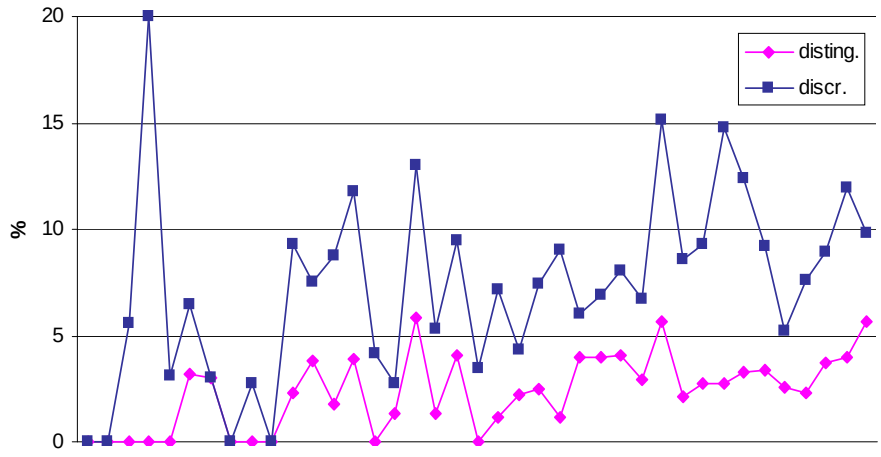

FIGURE 5.7 – Deviation in percentage from the absolute minimal compensable size over 40 randomly generated hierarchical FSMs.

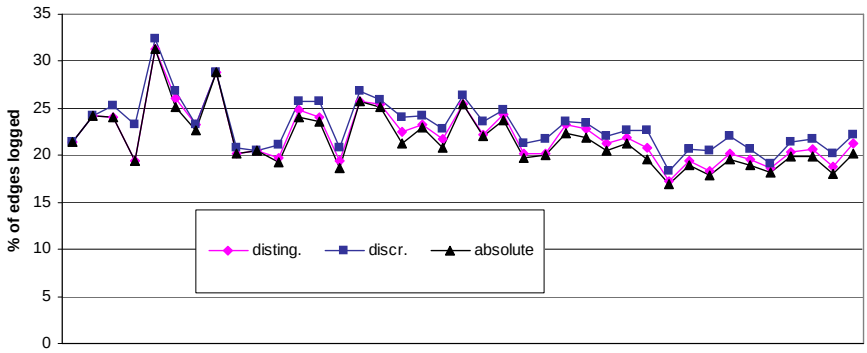

FIGURE 5.8 – Percentage of edges logged by the different algorithms, over 40 randomly generated hierarchical FSMs.

22% of transitions, respectively. In terms of time taken, the discriminating algorithm is instantaneous for our biggest FSM (700 transitions), the distinguishing algorithm takes 2.5 seconds to finish, and the absolute minimum takes half an hour.

5.4.2 General FSMs

Our previous analysis was made only on hierarchical FSMs, hence no general conclusions can be made. We now turn to more general FSMs (on which however we cannot know the absolute minimum), to get an idea whether our initial conclusions are true or not. We again use the *Synthetic DAG Generation Tool*, with random parameters for each size of FSM from 80 to 1200 transitions.

Fig. 5.9 shows the results using the discriminating and distinguishing algorithms, according to the number of transitions of the FSM. The graph shows a much more chaotic picture than the one obtained on hierarchical FSMs. Furthermore, the distinguishing algorithm seems to often do much better than the discriminating algorithm. Still, there are several cases (around 100 transitions) where both give the same results. Concerning time, the distinguishing algorithm takes at most 15 seconds to perform (note that the time taken is proportional to the number of transitions logged rather than to the number of transitions in the FSM).

Let us now analyze the percentage of transitions logged by the different algorithms (Fig. 5.10). Again, we see the chaosness of the picture, ranging from 15% to 50% of transitions logged by the discriminating algorithm (mean value 34%), and from 4% to 50% for the distinguishing algorithm (mean value 18%). The comparison with results obtained on hierarchical FSMs (Fig. 5.8) is quite interesting. The percentage of transitions can vary from 1 to 10, while it was from 1 to 2 in the hierarchical case. The discriminating algorithm does much worse in mean value (34% vs 22%), which is understandable since the FSMs are less regular and more complicated than in the hierarchical case. On the other hand, the distinguishing algorithm succeeds slightly better on these unrestricted FSMs than on hierarchical FSMs (18% vs 21%).

Finally, we give a summing up graph in Fig. 5.11, where we put each random FSM

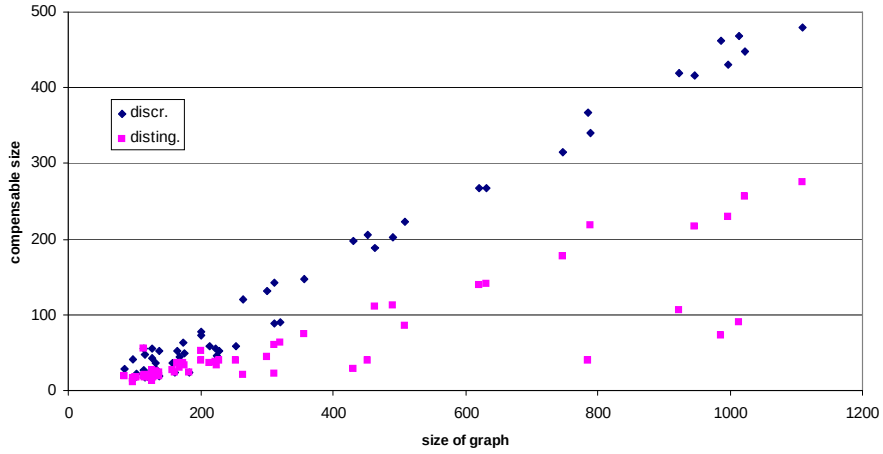

FIGURE 5.9 – Compensable size vs. number of transitions, over 60 randomly generated general FSMs.

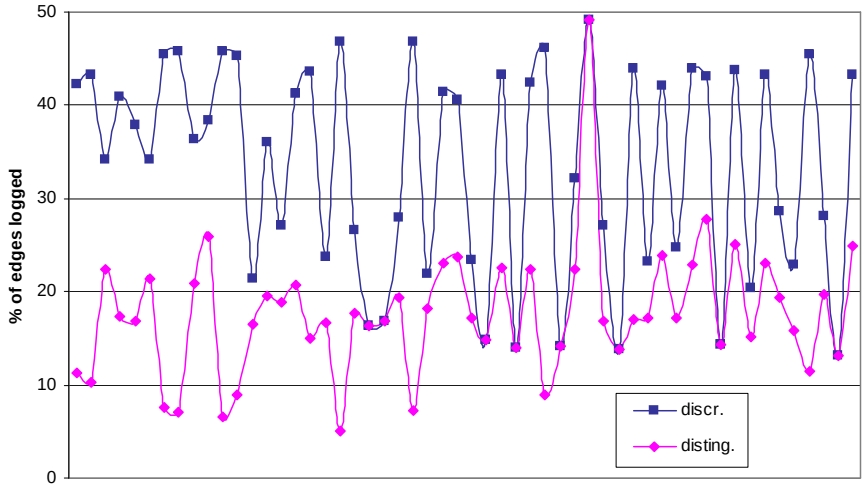

FIGURE 5.10 – Percentage of edges logged by the discriminating and distinguishing algorithms, over 60 randomly generated general FSMs.

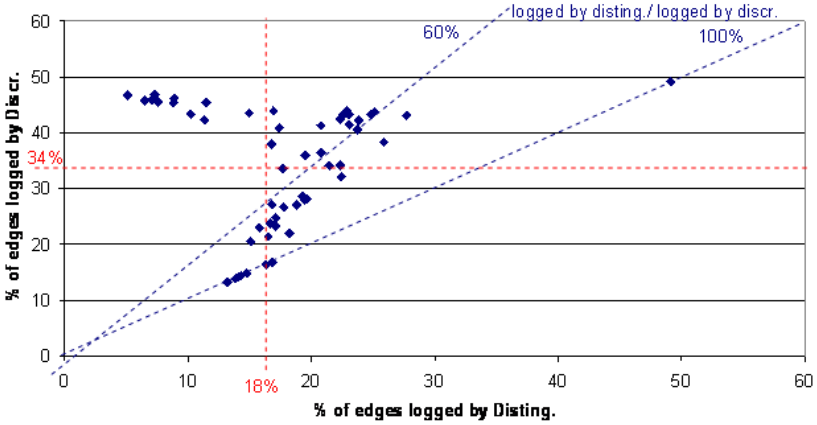

FIGURE 5.11 – Percentage of edges logged by the discriminating vs. distinguishing algorithm, over 60 randomly generated general FSMs.

we generated according to the percentage of transitions logged by the discriminating algorithm and by the distinguishing algorithm, together with two broken lines labeled by 18% (vertically for the distinguishing algorithm) and 34% (horizontally for the discriminating algorithm) showing the mean values of the percentage of transitions logged. On this graph, we can draw the line (broken line labelled 100%) on which both algorithms perform similarly. We can see that it happens several times, but mainly when the discriminating algorithm already gives good results (from 12% to 18% of transitions logged, which shall be close if not equal to the absolute minimum). Only once, the discriminating is bad and so is the distinguishing algorithm (around 50% of transitions logged). It was to be expected that such cases occur, since we know that the absolute minimum is not approximable, but luckily, it is pretty rare.

Overall, the distinguishing algorithm gives an observable size 0.6 times the size returned by the discriminating one (we draw a broken line labeled by 60% to separate the experiments under and over that value), and almost all of its answers are within 0.7 times of the discriminating algorithm. Moreover, it sometimes gives one tenth the number of transitions to log compared to the discriminating algorithm (which implies that the discriminating algorithm can be very inaccurate). Also, note that only once, the distinguishing algorithm gives more than 30% of transitions to log (1.5% of the FSMs), while it is the case for 50% of the FSMs with the discriminating algorithm.

5.5 Related Works

The minimality problem such that a given system property holds under partial observation has received considerable attention in the Discrete Event Systems (DES) literature. A Discrete Event System [Lin94] can be represented by the pair $G = (M, \mathcal{T}_c)$.

The first component M denotes a Mealy Automaton $M = (\Sigma, Q, Y, \delta, h)$ where Σ is the set of events, Q is the set of states, and $\delta : \Sigma \times Q \rightarrow 2^Q$ is the state transition function. Y is the output space and $h : \Sigma \times Q \rightarrow Y$ is the output function ($h(\sigma, q)$ is the observed output when σ occurs at q). In our case, outputs are the events observed themselves, that is, $Y = \Sigma_O \cup \{\epsilon\}$ where $\Sigma_O \subseteq \Sigma$. The second component $\mathcal{T}_C \subseteq \mathcal{T}$ is the set of controllable events, where the controllability of events is interpreted in a strong sense : a controllable event can be made to occur if physically possible.

With this definition, the problem is to select an optimum output function (or minimal observable set of events with $Y = \Sigma_O \cup \{\epsilon\}$) such that certain properties, e.g., observability, normality, diagnosability, etc. hold. For observability [LW88] to hold, if the control events following a pair of event traces are different, then the events observation information provided by the output function should be able to discriminate between such a pair of traces. The same property, in a distributed setting, translates to at least one controller being able to distinguish between such a pair of traces, known as Co-observability [RW92, CDFV88]. If in addition to the observability condition, we further require that all events that a controller can control be observable as well, then we have the stronger property of normality [KG94]. We cannot use the above research results directly as we do not have any such "special" events in our framework, that is, we require the capability to distinguish between any *pair* of events traces (and not only those which enable some special transitions). For diagnosability [SSL$^+$95], we need to be able to distinguish between a normal and faulty trace within a bounded number of steps from the time of occurrence of a fault. Our systems stop execution as soon as a transition fails, as such we need the capability to be able to distinguish between traces within 0 steps of failure occurrence. A variant of the above properties is to determine the exact state at any point of time, that is, we need to be able to distinguish between only those event traces which lead to different states, also known as state-observability [OW90]. The states can be further divided into a set of partitions T, and as long as we can at least distinguish between the partitions using the events observation information, then we have testability [BC94, Lin94].

The problem of determining a minimal observable set has been shown to be NP-complete for all the above properties. [YL02, RKK06, JKG03] prove the NP-completeness based on the vertex cover problem, directed graph *st*-cut problem, and *SATISFIABILITY* problem, respectively. Of course, the most relevant NP-completeness proof for us is that of [Mah76] which shows that determining a minimal observable set for compensability is NP-hard even for directed acyclic graphs. We extend the proof to show that the problem is NP-hard even for directed acyclic graphs with indegree and outdegree bounded by 3.

To the best of our knowledge, we are not aware of any heuristics or approximation algorithms for the minimal sensor selection or the uniconnected graph problem. Some works which have considered heuristics for related observability problems are mentioned below. Given observable events with an associated (installation) cost, [YG93] presents a greedy algorithm based on the observation cost, to select the minimal set of observed events. [RvS05] discusses approximation algorithms to select the minimal set of events to be communicated in a distributed supervisory controllers setting. [RKK06] gives a

polynomial time approximation algorithm for a special case of the observability problem.

From a pure transactional perspective, we are close to logging for nested transactions. For the basics of (single level) undo/redo recovery, the interested reader is referred to [PABG87]. [Mos87, Lom92] discuss the logging required to perform undo/redo recovery for nested transaction. Note that redo is required during recovery if a failure occurs before all transactional updates have been written to the stable storage (after commit). In our scenario, transaction updates are applied as and when they occur. Thus, we interested in an undo/no-redo strategy [BV04], where undo is performed using compensating operations. Of course, the above works assume that all the logs are accessible (visible), and the focus is on specifying the log format so that the nested structure of transactions/subtransactions is maintained, based on which the correct set of updates to be undone/redone can be determined and executed in the right order. Curiously, while the effect of partial log visibility has not received much attention from an atomicity perspective, it is close to the concurrency control problem of providing global serializability in the absence of a central/global CCM [GRS94]. However, the proposed solutions are more on the lines of mechanisms (e.g., tickets) to make the conflicts at different sites explicit, rather than determining a minimal set of conflicts/sites which need to be visible to be able to provide global serializability.

Conclusion

In this work, we studied visibility in hierarchical systems, a special type of distributed systems. A distributed system, by its very literal meaning, consists of more than one entity, say n entities. In an ideal world, each entity has knowledge of all the remaining $n-1$ entities, and there would be no need to consider the visibility notion explicitly. However, in the real world, an entity's visibility is restricted by : (i) its goals (necessity of the visibility), (ii) its security policies over others, and (iii) security policies of others towards it. Access control models have been well researched in literature and provide an elegant way of imposing the security restrictions. Unfortunately, security is only one piece of the puzzle and other middleware aspects, e.g., discovery, transactions, monitoring, etc. are equally essential for a successful distributed systems implementation. Assuming that there exists a priority among the aspects and the security restrictions are the first to be set for a given system, each of the other aspects would like to know if the allowed (not restricted) visibility is sufficient for its own purpose. Here, we cannot use the access control rules imposing the security restrictions to check compatibility with the other aspects, as they are based on different formalisms and use different terminologies. Thus, our objective was to study a high level conceptual model, which reflected both the allowed and restricted visibility of the given distributed system, and which was also generic enough to be compatible with more than one middleware aspect. We proposed visibility models for two specific hierarchical systems in this work, namely P2P Communities (Chapter 2) and Web services compositions (Chapter 3). On the other hand, a visibility model cannot model anything unless the visibility requirements and restrictions of the entities are first known. By default, each entity in a distributed system knows its visibility requirements and restrictions to satisfy its own goals. However, from a global perspective, we still need mechanisms to determine the visibility requirements and restrictions needed to satisfy global properties of the given system. Towards this end, we showed how to determine the minimum visibility required by a hierarchical Web services composition, to provide the transactional property of compensability. As the problem happens to be NP-complete, we presented decomposition algorithms to compute the absolute minimum required visibility in Chapter 4, and fast approximation algorithms in Chapter 5. A chapter wise summary of the contributions follows :

Chapter 2

We proposed a multi-level visibility graph formalism to capture the visibility of peers and communities in P2P systems. The graph model accommodates, with equal ease, any number of levels in the hierarchy as well as any modifications to the hierarchical structure.

Chapter 3

The main contribution of this chapter starts with a formalization of the visibility notion for hierarchical Web services compositions. The SoV definition is intuitive and encompasses "strong", "weak" and a variety of partially strong notions. The SoN definition follows naturally from that of SoV. We identified the properties of coherence and correlation (and their variants), and showed how the properties can be used to define meaningful visibility policies and represent the visibility model in a compact fashion.

Chapter 4

We studied compensation under partial log visibility. To the best of our knowledge, this problem has never been considered in the context of transactional services. We proposed a general divide and conquer framework which works on hierarchical services, and gives the absolute minimum number of transitions to log in order to get compensability. It provides good complexity results, that is, up to two exponentials better than using the brute force method on a flattening of the hierarchical graph. We also addressed services composed in a bottom-up fashion, and services for which their hierarchical structure is not given explicitly.

Chapter 5

We proposed two polynomial time algorithms to get an (over)approximation of the minimal number of actions to log in composite services to be able to compensate it. Our first algorithm based on positive discrimination of paths, is very fast, though it can be imprecise (in several cases, it gives at least 10 times as many transitions to log compared to the absolute minimal size). Our second algorithm based on a heuristic trying to distinguish paths, is slower but still efficient (we don't need more than 15 seconds for one run over 1200 transitions), and usually gives much smaller compensable sets. Still, in one case, it seems to give inaccurate results. There are probably some more heuristics to apply to get a more accurate algorithm. Nevertheless, in mean value, it seems that the distinguishing algorithm gives results close to the absolute minimum (18% of transitions, while we get 20% for the absolute minimum, looking at hierarchical FSMs), so efforts to optimize it further would probably not be worth it but for very few theoretical cases, and would slow down the algorithm.

Visibility Perspectives

The work in this thesis is complete in the sense that we identified visibility as an important characteristic of hierarchical systems, and showed both how to determine and represent visibility. However, given the wide applicability of the visibility notion, there are still many interesting unexplored avenues, or issues touched upon briefly in this work. We give some insights into those issues in the sequel :

- With respect to the underlying infrastructure :
  - Non-hierarchical systems : We studied visibility for hierarchical systems. However, not all Web services compositions are hierarchical, and with the technique of overlaying a hierarchy on a non-hierarchical system (e.g., P2P overlay networks [CGM02]), it will be interesting to see if the notion of visibility and the identified properties of coherence and correlation, apply to non-hierarchical systems as well.
  - Dynamic structure : The proposed minimal visibility algorithms are clearly for static compositions. Given a hierarchical composition that keeps evolving with the discovery or termination of a component service, obtaining a minimal compensable set would need recomputation. However, any such recomputation is needed only at its level of the hierarchy (not below), plus *few* levels above (until the properties of a level are unchanged).
  - Multiple hierarchies : The P2P visibility model, and the search and update algorithms, in Chapter 2, have been defined for a single underlying hierarchy. The scenario becomes more interesting as soon as we allow for multiple overlapping hierarchies. For example, in Fig. 2.1, in addition to the hierarchical classification by type, the peers may also be chronologically (again, hierarchically) classified by their interest in decades, years and months. Given this, a query evaluation, after some initial search with respect to the type hierarchy and determining the year of production, might switch to a search by the chronological hierarchy.
- With respect to applicability : We have considered the application of visibility for transactional and security aspects in this work. We believe that it will be equally beneficial for discovery and monitoring of services, among other middleware aspects.
  - Discovery : Given a pool of existing services and a goal, discovery consists of finding the most optimum (e.g., cheapest) service capable of fulfilling the given goal. Clearly, the larger the pool the better. However, due to physical and security limitations, the discovery engine initially may not have visibility over all the existing services. Given this, it can only choose among the services visible to it, or we can envision a scenario where its visibility increases incrementally after each selection (maybe over services related to the selected service). Further, composite services are also eligible for discovery, and as such, need to be described and published. The capabilities and constraints of a composite service are basically a combination of its components' capabilities and constraints, and need to be consistent with them. Here, some component services may have

objections with respect to exposing their characteristics as part of the composite service's description due to security/confidentiality reasons, or we may not know the component services which are eventually going to be invoked during execution due to the inherent non-determinism (choice operators). Thus, we need some mechanism to choose the component services which should be (visible) exposed as part of the corresponding composite service's description. [Bis07] provides some pointers in this direction.

- Monitoring : For long running complex systems, their progress needs to be monitored. This includes sending status notifications, and in the event of a failure, notifying the entity responsible for handling that failure as soon as possible. To avoid redundancy and improve performance, such notifications should only be sent to the concerned entities, and not flooded to the whole system. The entities which need to be notified with respect to change/failure of an entity, can be captured as the *noticeability* of that entity. For example, let us consider the repair management system outlined in [BBH+05]. An instance of the repair management architecture constitutes a management domain, that is, a set of components under a single repair management authority. The components in turn consist of sub-components. The components and sub-components correspond to an abstraction of a physical computer and the software systems executing on that computer, respectively. As the repair management authority is responsible for repair of the components and (its children) sub-components in its domain, it should have visibility over them. Analogously, in the event of a component or sub-component failure, notifications need to be sent only to the entities in its SoN (in this case, the repair management authority of its domain). For more details of the repair mechanism, please refer to [BBH+05].

- From an algorithmic point of view :
  - Properties : In Chapter 3, we identified the dual properties of coherence and correlation (and their variants) which lead to practical visibility policies as well as a compact representation. We would like to look for other nice visibility properties that can be applied to general hierarchical and non-hierarchical systems. It will also be interesting to study efficient algorithms to check if the visibility assignment of a given system indeed satisfies coherence, correlation, or some other visibility property.
  - Constraints : By design, the proposed minimal visibility algorithms only consider a subset of the whole set of transitions. It is thus straightforward to add constraints, such as, a subset of transitions "can/cannot be logged". It is very useful since in practice, we have to take into account security/privacy issues. The algorithm would then answer the absolute minimal compensable set among those satisfying the constraints.
  - Incremental : Currently, we compute the minimal compensable set at the beginning, before execution begins. It will be interesting to try and do it incrementally as execution progresses, that is, the compensable set (the services which need to be visible) keeps evolving. Basically, after the execution of a transition or on reaching a specific state, it is possible that some transitions do not need

to be visible any longer. Starting with zero visible transitions, the visibility of a transition for compensability might be required, then not required, and then required again, at different stages of the execution. Given this, the minimal compensable set needs to be recomputed as execution progresses. [CT08] provides some pointers in this direction.

– Negotiation : In Chapter 3, we assumed the existence of a non-conflicting set of visibility requirements and restrictions to be modeled. However, in the real world, the requirements of one middleware aspect are very likely to conflict with the restrictions of another aspect. As such, we would need some negotiation mechanism [CWZL05] first to reach the non-conflicting stage.

Bibliography

[AAC+99] Serge Abiteboul, Bernd Amann, Sophie Cluet, Adi Eyal, Laurent Mignet, and Tova Milo. Active views for electronic commerce. *in: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB) (Morgan Kaufmann Publishers)*, pages 138–149, 1999.

[ABC+03] Serge Abiteboul, Angela Bonifati, Gregory Cobena, Ioana Manolescu, and Tova Milo. Dynamic xml documents with distribution and replication. *in: Proceedings of the ACM SIGMOD International Conference on Management of Data (ACM Press)*, pages 527–538, 2003.

[ABCM04] Serge Abiteboul, Omar Benjelloun, Bogdan Cautis, and Tova Milo. Active xml, security and access control. *in: Proceedings of the XIX Simpósio Brasileiro de Bancos de Dados (SBBD)*, pages 13–22, 2004.

[ABp] Activebpel bpel implementation. *http://www.activebpel.org*.

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. Web services: Concepts, architecture and applications. *ISBN: 3540440089 (Springer Verlag)*, 2004.

[AH00] Gustavo Alonso and Claus Hagen. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering 26(10)*, pages 943–958, 2000.

[AR03] Asif Akram and Omer F. Rana. Structuring peer-2-peer communities. *in: Proceedings of the 3rd International Conference on on Peer-to-Peer Computing (P2P) (IEEE Computer Scoiety Press)*, pages 194–195, 2003.

[AXM] Active xml (axml) systems. *http://www.activexml.net*.

[BBH+05] Sara Bouchenak, Fabienne Boyer, Daniel Hagimont, Sacha Krakowiak1, Adrian Mos, Noel de Palma, Vivien Quema, and Jean-Bernard Stefani. Architecture-based autonomous repair management: An application to j2ee clusters. *in: Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems (SRDS) (IEEE Computer Society Press)*, pages 13–24, 2005.

[BC94] Sanjiv Bavishi and Edwin K. P. Chong. Automated fault diagnosis using a discrete event systems framework. *in: Proceedings of the 9th IEEE International Symposium on Intelligent Control (IC) (IEEE Computer Society Press)*, pages 213–218, 1994.

[BCLM03] Daniela Berardi, Diego Calvanese, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. *in: Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC), Lecture Notes in Computer Science vol. 2910 (Springer Verlag)*, pages 621–630, 2003.

[BGG08] Debmalya Biswas, Thomas Gazagnaire, and Blaise Genest. Small logs for transactional services: Distinction is much more accurate than (positive) discrimination. *in: Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium (HASE) (IEEE Computer Society Press)*, 2008.

[Bis04] Debmalya Biswas. Compensation in the world of web services composition. *in: Proceedings of the 1st ICWS Workshop on Semantic Web Services and Web Process Composition (SWSWPC), Lecture Notes in Computer Science vol. 3387 (Springer Verlag)*, pages 69–80, 2004.

[Bis07] Debmalya Biswas. Web services discovery and constraints composition. *in: Proceedings of the 1st International Conference on Web Reasoning and Rule Systems (RR), Lecture Notes in Computer Science vol. 4524 (Springer Verlag)*, pages 73–87, 2007.

[Bis08] Debmalya Biswas. Active aml replication and recovery. *in: Proceedings of the 2nd International Conference on Complex Intelligent and Software Intensive Systems (CISIS) (IEEE Computer Society Press)*, pages 263–269, 2008.

[BK08] Debmalya Biswas and Il-Gon Kim. Active xml transactions. *Book Chapter: Service and Business Computing Solutions with XML: Applications for Quality Management and Best Processes (IGI Global)*, pages –, 2008.

[BKK05] Giorgio Busatto, Hans-Jorg Kreowski, and Sabine Kuske. Abstract hierarchical graph transformation. *Mathematical Structures in Computer Science 15 (Cambridge University Press)*, pages 773–819, 2005.

[BKR+99] Yuri Breitbart, Raghavan Komondoor, Rajeev Rastogi, S. Seshadri, and Abraham Silberschatz. Update propagation protocols for replicated databases. *in: Proceedings of the ACM SIGMOD International Conference on Management of Data (ACM Press)*, pages 97–108, 1999.

[Boc04] Laura Bocchi. Compositional nested long running transactions. *in: Proceedings of the 7th International Conference on Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science vol. 2984 (Springer Verlag)*, pages 194–208, 2004.

[BPE] Business process execution language for web services (bpel) specification. *http://www-106.ibm.com/developerworks/library/ws-bpel/*.

[BV04] Debmalya Biswas and Krishnamurthy Vidyasankar. A nested transaction model for ldap transactions. *in: Proceedings of the 1st International Conference on Distributed Computing and Internet Technology (ICDCIT), Lecture Notes in Computer Science vol. 3347 (Springer Verlag)*, pages 117–126, 2004.

[BV05] Debmalya Biswas and Krishnamurthy Vidyasankar. Monitoring for hierarchical web services compositions. *in: Proceedings of the 6th VLDB Workshop on Technologies for Electronic Services (TES), Lecture Notes in Computer Science vol. 3811 (Springer Verlag)*, pages 98–112, 2005.

[CD96] Qiming Chen and Umeshwar Dayal. A transactional nested process management system. *in: Proceedings of the 12th International Conference on 12th International Conference on Data Engineering (ICDE) (IEEE Computer Society Press)*, pages 566–573, 1996.

[CDFV88] Randy Cieslak, C. Desclaux, Ayman S. Fawaz, and Pravin Varaiya. Supervisory control of discrete event processes with partial observation. *IEEE Transactions on Automatic Control, 33(3)*, pages 249–260, 1988.

[CGM02] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. *Technical Report, Stanford University, USA*, 2002.

[CT08] Franck Cassez and Stavros Tripakis. Fault diagnosis with dynamic observers. *in: Proceedings of the 9th International Workshop on Discrete Event Systems (WoDES) (IEEE Computer Society Press)*, pages 212–217, 2008.

[CWZL05] Jian Cao, Jie Wang, Shensheng Zhang, and Minglu Li. A multi-agent negotiation based service composition method for on-demand service. *in: Proceedings of the International Conference on Services Computing (SCC) (IEEE Computer Society Press)*, pages 329–332, 2005.

[dag] Synthetic directed acyclic graph generation tool. *http://www.loria.fr/~suter/dags.html.*

[Dav78] Charles T. Davies. Data processing spheres of control. *IBM Systems Journal 17(2)*, pages 179–198, 1978.

[DS04] Khuzaima Daudjee and Kenneth Salem. Lazy database replication with ordering guarantees. *in: Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE) (IEEE Computer Society Press)*, pages 424–435, 2004.

[FDR99] Failure-divergence refinement (fdr) 2 user manual. *Formal Systems(Europe) Limited*, 1999.

[FF01] Robert E. Filman and Daniel P. Friedman. Aspect-oriented programming is quantification and obliviousness. *RIACS Technical Report, http://www.riacs.edu/research/technical_reports/TR_pdf/TR_01.12.pdf*, 2001.

[Fie05] Ludger Fiege. Visibility in event-based systems. *Ph.D. Thesis, Department of Computer Science, Darmstadt University of Technology, Darmstadt, Germany*, 2005.

[Gaz07] Thomas Gazagnaire. Langages de scénarios: Utiliser des ordres partiels pour modéliser, vérifier et superviser des systèmes parallèles et répartis. *PhD Thesis, IRISA, France*, 2007.

[GJ79] Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of np-completeness. *ISBN: 9780716710455 (W. H. Freeman and Company)*, 1979.

[GMS87] Hector Garcia-Molina and Kenneth Salem. Sagas. *ACM SIGMOD Record 16(3)*, pages 249–259, 1987.

[GR01] Chris Godsil and Gordon Royle. Algebraic graph theory. *ISBN: 0387952209 (Springer Verlag)*, 2001.

[Gri] Grid computing. *http://www.ogf.org/*.

[GRS94] Dimitrios Georgakopoulos, Marek Rusinkiewicz, and Amit P. Sheth. Using tickets to enforce the serializability of multidatabase transactions. *IEEE Transactions on Knowledge and Data Engineering, 6(1)*, pages 166–180, 1994.

[HH04] Michael P. Haustein and Theo Harder. Adjustable transaction isolation in xml database management systems. *in: Proceedings of the 2nd International XML Database Symposium (XSym), Lecture Notes in Computer Science vol. 3186 (Springer Verlag)*, pages 173–188, 2004.

[IMZI04] Hoh Peter In, Konstantinos A. Meintanis, Ming Zhang, and Eul Gyu Im. Kaliphimos: A community-based peer-to-peer group management scheme. *in: Proceedings of the 3rd International Conference on Advances in Information Systems (ADVIS), Lecture Notes in Computer Science vol. 3261 (Springer Verlag)*, pages 533–542, 2004.

[JCW02] Kuen-Fang Jea, Shih-Ying Chen, and Sheng-Hsien Wang. Concurrency control in xml document databases: Xpath locking protocol. *in: Proceedings of the 9th International Conference on Parallel and Distributed Systems (ICPADS) (IEEE Computer Society Press)*, pages 551–556, 2002.

[JKG03] Shengbing Jiang, Ratnesh Kumar, and Humberto E. Garcia. Optimal sensor selection for discrete-event systems with partial observation. *IEEE Transactions on Automatic Control, 48(3)*, pages 369–381, 2003.

[JPPMKA02] Ricardo Jiménez-Peris, Marta Patiño-Martínez, Bettina Kemme, and Gustavo Alonso. Improving the scalability of fault tolerant database clusters. *in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS) (IEEE Computer Society Press)*, pages 477–484, 2002.

[KG94] Ratnesh Kumar and Vijay K. Garg. Modeling and control of logical discrete event systems. *ISBN: 9780792395386 (Springer)*, 1994.

[KL70] Brian W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell Systems Technical Journal, 49(2)*, pages 291–307, 1970.

[KRD02] Mujtaba Khambatti, Kyung Ryu, and Partha Dasgupta. Peer-to-peer communities: Formation and discovery. *in: Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing Systems (PDCS) (Acta Press)*, pages 161–166, 2002.

[KRD03] Mujtaba Khambatti, Kyung Dong Ryu, and Partha Dasgupta. Push-pull gossiping for information sharing in peer-to-peer communities. *in: Proceedings of the IASTED International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) (ACTA Press)*, pages 1393–1399, 2003.

[LAP03] Alexander Lazovik, Marco Aiello, and Mike Papazoglou. Planning and monitoring the execution of web service requests. *in: Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC), Lecture Notes in Computer Science vol. 2910 (Springer Verlag)*, pages 335–350, 2003.

[Lif] Microsoft's mylifebits project. *http://research.microsoft.com/barc/mediapresence/MyLif*

[Lin94] Feng Lin. Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems (Springer Netherlands), 4(2)*, pages 197–212, 1994.

[Lom92] David B. Lomet. Mlr: A recovery method for multi-level systems. *in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD Record 21(2) (ACM Press)*, pages 185–194, 1992.

[Low97] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *in: Proceedings of the 10th Computer Security Foundations Workshop (CSFW) (IEEE Computer Society Press)*, pages 18–30, 1997.

[LTS98] Fu-Ren Lin, Gek Woo Tan, and Michael J. Shaw. Modeling supply-chain networks by a multi-agent system. *in: Proceedings of the 31st Annual Hawaii International Conference on System Science (HICSS) (IEEE Computer Society Press)*, pages 105–114, 1998.

[LW88] Feng Lin and W. Murray Wonham. On observability of discrete-event systems. *Information Sciences (Elsevier Science), 44(3)*, pages 173–198, 1988.

[Mah76] S. Maheshwari. Traversal marker placement problems are np-complete. *Research Report, Colorado Boulder University, USA*, 1976.

[MNBE08] Masakazu Maruoka, Alireza Goodarzi Nemati, Valbona Barolli, and Tomoya Enokido. Role-based access control in peer-to-peer (p2p) societies. *in: Proceedings of the 22nd International Conference on Advanced Information Networking and Applications (AINA) Workshops (IEEE Computer Society Press)*, pages 495–500, 2008.

[Mof98] Jonathan D. Moffett. Control principles and role hierarchies. *in: Proceedings of the 3rd ACM Workshop on Role-Based Access Control (RBAC) (ACM Press)*, pages 63–69, 1998.

[Mos81] T. E. B. Moss. Nested transactions: An approach to reliable distributed computing. *PhD Thesis, MIT Laboratory for Computer Science, USA*, 1981.

[Mos87] J. E. B. Moss. Log-based recovery for nested transactions. *in: Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, pages 427–432, 1987.

[OW90] Cuneyt M. Ozveren and Alan S. Wilsky. Observability of discrete event dynamical systems. *IEEE Transactions on Automatic Control, 35(7)*, pages 797–806, 1990.

[PABG87] Vassos Hadzilacos Philip A. Bernstein and Nathan Goodman. Concurrency control and recovery in database systems. *ISBN: 0201107155 (Addison-Wesley)*, 1987.

[RKK06] Kurt Rohloff, Samir Khuller, and Guy Kortsarz. Approximating the minimal sensor selection for supervisory control. *Discrete Event Dynamic Systems (Springer Netherlands), 16(1)*, pages 143–170, 2006.

[RS00] Arnon Rosenthal and Edward Sciore. Extending sql's grant and revoke operations, to limit and reactivate privileges. *in: Proceedings of the IFIP TC11/ WG11.3 14th Annual Working Conference on Database Security (DBSec) (Kluwer Academic)*, pages 209–220, 2000.

[RvS05] Kurt R. Rohloff and Jan H. van Schuppen. Approximating minimal communicated event sets for decentralized supervisory control. *in: Proceedings of the 16th IFAC World Congress (Elsevier Science)*, 2005.

[RW92] Karen Rudie and W. Murray Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control, 37(11)*, pages 1692–1708, 1992.

[SO00] Wasim Sadiq and Maria E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems 25(2) (Elsevier Science)*, pages 117–134, 2000.

[SSL+95] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinaamohideen, and Demosthenis Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control, 40(9)*, pages 1555–1575, 1995.

[TIA] Darpa's total information awareness system. *http://www.heritage.org/Research/HomelandSecurity/wm175.cfm.*

[UDD] Universal description, discovery and integration (uddi) specification. *http://www.uddi.org/specification.html.*

[vie] Database views. *http://en.wikipedia.org/wiki/View_(database).*

[VV04] Krishnamurthy Vidyasankar and Gottfried Vossen. Multi-level model for web service composition. *in: Proceedings of the 2nd International Conference on Web Services (ICWS) (IEEE Computer Society Press)*, pages 462–471, 2004.

[WDSS93] Gerhard Weikum, Andrew Deacon, Werner Schaad, and Hans-Jorg Schek. Open nested transactions in federated database systems. *IEEE Data Engineering Bulletin 16(2)*, pages 4–7, 1993.

[WFN04] Andreas Wombacher, Peter Fankhauser, and Erich Neuhold. Transforming bpel into annotated deterministic finite state automata for service discovery. *in: Proceedings of the 2nd International Conference on Web Services (ICWS) (IEEE Computer Society Press)*, pages 316–323, 2004.

[WS-] Web services security (ws-security) specification. *http://en.wikipedia.org/wiki/WS-Security*.

[WSD] Web services description language (wsdl) specification. *http://www.w3.org/2002/ws/desc/*.

[WST] Web services transactions specifications. *http://msdn2.microsoft.com/en-us/library/ms951262.aspx*.

[WV01] Gerhard Weikum and Gottfried Vossen. Transactional information systems: Theory, algorithms, and the practice of concurrency control. *ISBN: 1558605088 (Morgan Kaufmann Publishers)*, 2001.

[YG] Yahoo groups. *http://groups.yahoo.com/*.

[YG93] Stanley D. Young and Vijay K. Garg. Optimal sensor and actuator choices for discrete event systems. *in: Proceedings of the 31st Allerton Conference on Communication, Control, and Computing*, 1993.

[YL02] Tae-Sic Yoo and Stéphane Lafortune. Np-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Transactions on Automatic Control, 47(9)*, pages 1495–1499, 2002.

[YS01] Pinar Yolum and Munindar P. Singh. Commitment machines. *in: Proceedings of the 8th International Workshop on Intelligent Agents (ATAL), Lecture Notes in Computer Science vol. 2333 (Springer Verlag)*, pages 209–220, 2001.

Table des figures

Résumé

Les modèles hiérarchiques fournissent un élégant mécanisme d'analyse à différents niveaux d'abstraction. Ils sont habituellement construits dans une approche top-down ou bottom-up, niveau par niveau. En conséquence, par construction, la visibilité des entités dans un système hiérarchique est limitée aux niveaux adjacents (parent-enfant). Cette visibilité n'est souvent pas suffisante pour modèliser tous les scenarios de la vie réelle. D'autre part, une interaction arbitraire, sans aucune restriction, n'est pas une solution acceptable, en raison souvent de préoccupations en matière de sécurité. Dans cette thèse, nous nous adressons à deux sous-problèmes de la visibilité de systèmes hiérarchiques. Tout d'abord, nous considérons le problème de la définition d'un modèle de visibilité, compte tenu des exigences et des restrictions des différentes entités dans une hiérarchie. Nous présentons des modèles de visibilité à base de graphes pour deux types de systèmes hiérarchiques : les Communautés P2P et les services Web compositionnels. Deuxièmement, nous nous intéressons au problème orthogonal de déterminer la visibilité dans la hiérarchie pour qu'une propriété particulière soit satisfaite. Nous donnons des algorithmes pour calculer la visibilité minimale requise pour une hiérarchie de services Web compositionnels afin de fournir la propriété de l'atomicité des opérations. Cette visibilité minimale est calculée à la fois de manière absolue et de manière approchée.

Abstract

Hierarchical systems provide an elegant mechanism to analyze system functionality at different levels of abstraction. They are usually constructed in a top-down or bottom-up fashion level by level. As a result, by construction, visibility of entities in a hierarchical system is restricted to adjacent (parent-child) levels. Such restricted visibility is often not sufficient for real-life scenarios. On the other hand, allowing arbitrary interaction among the hierarchical entities, without any restrictions, is not an acceptable solution either due to security concerns. In this thesis, we address two sub-problems of the visibility issue in hierarchical systems. First, we consider the problem of defining a visibility model, given the visibility requirements and restrictions of the different entities in a hierarchy. We present graph based visibility models for two specific hierarchical systems : P2P Communities and Web services compositions. Second, we deal with the orthogonal problem of determining the visibility requirements of the given hierarchy such that a specific property holds. We give both absolute and approximate algorithms to determine the minimal visibility required by a hierarchical Web services composition to provide the property of transactional atomicity.