Distributed data management with access control : social Networks and Data of the Web

Alban Galland

Thèse de Doctorat en Informatique

École Doctorale d'Informatique de Paris-Sud (Paris 11)

Distributed Data Management with Access Control

Social Networks and Data of the Web

Gestion de Données Distribuées avec Contrôle d'Accès

Réseaux sociaux et données du Web

Alban GALLAND (INRIA Saclay & LSV ENS Cachan)

28 septembre 2011

Jury

Serge ABITEBOUL	DR INRIA Saclay	(directeur)
Bernd AMANN	Prof. Univ. Pierre et Marie Curie	(rapporteur)
Nicole BIDOIT	Prof. Univ. Paris Sud	
Maurizio LENZERINI	Prof. Univ. Rome	(rapporteur)
Philippe RIGAUX	Prof. CNAM	
Marie-Christine ROUSSET	Prof. Univ. Grenoble	

Résumé

La masse d'information disponible sur le Web s'accroit rapidement, sous l'afflux de données en provenance des utilisateurs et des compagnies. Ces données qu'ils souhaitent partager de façon controllée sur le réseau et qui sont réparties sur de nombreuses machines et systèmes différents, ne sont rapidement plus gérables directement par des moyens humains. Nous introduisons *WebdamExchange*, un nouveau modèle de bases de connaissances distribuées, qui comprend des assertions au sujet des données, du contrôle d'accés et de la distribution. Ces assertions peuvent être échangées avec d'autres pairs, répliquées, interrogées et mises à jour, en gardant la trace de leur origine. La base de connaissance permet aussi de guider de façon automatique sa propre gestion. *WebdamExchange* est basé sur *WebdamLog*, un nouveau langage de règles pour la gestion de données distribuées, qui associe formellement les règles déductives de Datalog avec négation et les règles actives de Datalog$^{\neg\neg}$. *WebdamLog* met l'accent sur la dynamicité et les interactions, caractéristiques du Web 2.0. Ce modèle procure à la fois un langage expressif pour la spécification de systèmes distribués complexes et un cadre formel pour l'étude de propriétés fondamentales de la distribution. Nous présentons aussi une implémentation de notre base de connaissance. Nous pensons que ces contributions forment une fondation solide pour surmonter les problèmes de gestion de données du Web, en particulier dans le cadre du contrôle d'accès.

Mots clefs

Distribution, Controle d'Accès, Réseaux Sociaux, Gestion de Données du Web, Datalog Distribué

Abstract

The amount of information on the Web is spreading very rapidly. Users as well as companies bring data to the network and are willing to share with others. They quickly reach a situation where their information is hosted on many machines they own and on a large number of autonomous systems where they have accounts. Management of all this information is rapidly becoming beyond human expertise. We introduce *WebdamExchange*, a novel distributed knowledge-base model that includes logical statements for specifying information, access control, secrets, distribution, and knowledge about other peers. These statements can be communicated, replicated, queried, and updated, while keeping track of time and provenance. The resulting knowledge guides distributed data management. *WebdamExchange* model is based on *WebdamLog*, a new rule-based language for distributed data management that combines in a formal setting deductive rules as in Datalog with negation, (to specify intensional data) and active rules as in Datalog$^{\neg\neg}$ (for updates and communications). The model provides a novel setting with a strong emphasis on dynamicity and interactions (in a Web 2.0 style). Because the model is powerful, it provides a clean basis for the specification of complex distributed applications. Because it is simple, it provides a formal framework for studying many facets of the problem such as distribution, concurrency, and expressivity in the context of distributed autonomous peers. We also discuss an implementation of a proof-of-concept system that handles all the components of the knowledge base and experiments with a lighter system designed for smartphones. We believe that these contributions are a good foundation to overcome the problems of Web data management, in particular with respect to access control.

Keywords

Distribution, Access Control, Social Network, Web Data Management, Distributed Datalog

Contents

Acknowledgement

I would like to thanks the people I worked with during these three years, my advisor Serge Abiteboul, my co-authors Sihem Amer-Yahia, Émilien Antoine, Meghyn Bienvenu, Pierre Bourhis, Kristian Lyngbaek, Amélie Marian, Bogdan Marinoiu, Neoklis Polyzotis, Marie-Christine Rousset and Pierre Senellart and the members of my teams Gemo, Leo, Dahu and *Webdam*.

More generally, I thank the members, researchers as supporting staff, of my laboratories, Institut National de Recherche en Informatique et Automatique de Saclay – Île-de-France, the Laboratoire de Recherche en Informatique of Université Paris-Sud 11 and the Laboratoire Spécification et Vérification of the École National Supérieure de Cachan. I thank as well the members of the DBWeb team, who welcomed me at Telecom ParisTech each time I wanted to come. I have also a special though to my previous teachers, in particular to my master professors in École Polytechnique, Télécom ParisTech and Université Pierre et Marie Curie (Paris 6)) and to my previous advisors, in particular Christian Boitet and Sihem Amer-Yahia. I thank my colleagues from the Conseil Général de l'Industrie, de l'Énergie et des Technologies, who help me seconding at INRIA. I also thank my family and friends for their support throughout this thesis preparation.

Last but not least, I thank the numerous proof readers of this thesis, in particular my reviewers, Bernd Amann and Maurizio Lenzerini, for their helpful comments.

Résumé en Français

Le volume d'informations présentes sur le Web ne cesse de s'accroitre. Les utilisateurs comme les compagnies veulent en effet partager leurs données, qui se trouvent distribuées sur les nombreuses machines qu'ils possèdent ou sur les comptes qu'ils possèdent dans des systèmes externes. En particulier, l'émergence du Web 2.0 et des réseaux sociaux a permis aux utilisateurs de partager encore plus de données privées. La gestion de cette information dépasse rapidement l'expertise humaine. Les informations manipulées par les utilisateurs ont de nombreuses facettes : elles concernent des données personnelles (photos, films, musique, mails), des données sociales (annotations, recommandation, liens sociaux), la localisation des données (marque-pages), les informations de contrôle d'accès (mots de passe, clés privées), les services Web (moteur de recherche, archives), la sémantique (ontologies), la croyance et la provenance. Les tâches exécutées par les utilisateurs sont très variées : recherches par mots clef, requêtes structurées, mise-à-jour, authentification, fouille de donnée et extraction de connaissances. Dans cette thèse, nous montrons que toute cette information devrait être modélisée comme un problème de gestion d'une base de connaissance distribuée. Nous soutenons aussi que Datalog et ses extensions est une base formelle sûre pour représenter ces informations et ces tâches. Ce travail fait partie du projet ERC Webdam [Web] sur les fondations de la gestion des données du Web. Le but de ce projet est de participer au développement de fondations formelles unifiées pour la gestion de données distribuées, le manque actuel de telles fondations ralentissant les progrès dans ce domaine.

Contributions

Les contributions de cette thèse sont les suivantes :

- Nous introduisons WebdamExchange, une nouvelle base de connaissances distribuée qui contient des assertions logiques pour représen-

ter les données, le contrôle d'accès, les secrets, la distribution et les connaissances sur les autres pairs. Ces assertions peuvent être communiquées, répliquées, interrogées et mises-à-jour, en traçant le temps et la provenance. La base de connaissance guide la gestion des données distribuées. La complexité de la gestion moderne et distribuée des données nous semble nécessiter un tel modèle, qui permet de représenter dans le même modèle formel des concepts jusqu'alors étudiés en isolation. Nous démontrons la généralité et la flexibilité de WebdamExchange, en montrant comment des protocoles de distribution et de contrôle d'accès très différents peuvent être représentés dans notre modèle.

- Nous présentons WebdamLog, un nouveau langage à base de règles pour la gestion de données distribuées qui combine dans un cadre formel les règles déductives de Datalog avec négation pour la définition des faits intensionnels et les règles actives de Datalog $\neg\neg$ pour les mises-à-jour et les communications. Le modèle met un accent fort sur la dynamique et les interactions typiques du Web 2.0. Ce modèle est à la fois suffisamment puissant pour spécifier des systèmes distribués complexes et suffisamment simple pour permettre une étude formelle de la distribution, de la concurrence et de l'expressivité dans un système de pairs autonomes.

- Nous mentionnons aussi une implémentation d'un démonstrateur qui gère les concepts essentiels de la base de connaissance ainsi que d'un démonstrateur plus simple pour téléphones.

Nous pensons que ces contributions forment une bonne base pour résoudre les problèmes fréquemment rencontrés dans l'échange de données sur le Web, en particulier pour les réseaux sociaux.

Motivation par l'exemple

Nous introduisons ici brièvement un exemple qui sera utilisé par la suite pour illustrer la thèse. Imaginons un groupe d'escaladeurs qui souhaitent organiser des sorties régulières à Fontainebleau. Alice, membre de ce groupe, pourrait avoir accès à une liste des rochers qu'elle a déjà grimpés, stockée sur son iPhone. Bob, un autre membre, pourrait avoir créé une carte Google pour localiser les circuits d'escalade. La liste des membres du groupe pourrait être disponible sur Facebook. La liste de tous les rochers avec leur cotation pourrait être stockée dans un fichier Excel sur un réseau

pair-à-pair. D'autres données pourraient être disponibles via des services Web, comme la météo ou le calcul de trajets. Pour organiser les sorties, Alice doit trouver où sont stockées ces données, quelles sont les informations d'authentification nécessaires pour y accéder, et intégrer ces données à sa propre base de connaissance. Bien sûr, toutes ces données pourraient être stockées sur un système unique, qui en assurerait la cohérence, la distribution et l'accès. Néanmoins, nous pensons personnellement qu'il serait préférable de décrire ces différentes données dans un formalisme unique, qui constituerait une base de connaissances distribuée. Avec notre modèle, les utilisateurs peuvent distribuer leurs données sur plusieurs hôtes, tout en gardant le contrôle sur leur accès. Ils peuvent aussi naviguer et intégrer ces données de façon transparente.

Les principales difficultés soulevées par ce scénario concernent le contrôle d'accès, la distribution et la manipulation des données. En effet, pour des données personnelles, le contrôle d'accès est fondamental et Alice et ses amis peuvent définir des politiques de sécurité très diverses. La cohabitation de ces politiques doit être permise pas le système, et le modèle doit permettre de vérifier leur sécurité globale. S'il n'est pas toujours possible d'empêcher certains comportements comme l'échange de données par des pairs corrompus, il est important de tracer la provenance des données pour repérer au plus vite ces comportements. La gestion de la distribution est aussi fondamentale sur le Web, ce dernier favorisant l'éclatement des données et la multiplication des protocoles d'accès. Enfin, il est important de fournir les moyens d'éditer, de naviguer et d'interroger les données si nécessaire. L'utilisateur doit pouvoir localiser les données auxquelles il a un droit d'accès, mais aussi déléguer des tâches à d'autres pairs s'il ne peut les exécuter lui-même.

Résumé de l'état de l'art

Cette thèse correspond à deux domaines importants de l'informatique, les systèmes de données distribués et le contrôle d'accès. Les systèmes distribués [ÖV99, AMR$^+$ar] sont des logiciels qui servent à coordonner les actions de plusieurs ordinateurs, à travers l'envoi de messages. Ils sont caractérisés par les notions de consistance, de fiabilité, de disponibilité, de passage à l'échelle et d'efficacité. Dans le cas des bases de données, le système consiste en un ensemble de plusieurs bases de données, logiquement liées, distribuées sur un réseau d'ordinateur. La distribution est transparente pour l'utilisateur : le résultat d'une requête ne dépend pas a priori du pair sur lequel elle a été posée. Sur le Web, la distribution est une

composante de base de l'organisation du système. Le développement d'un langage commun, XML, et des autres standards, a facilité l'expansion des échanges. Enfin, les systèmes pairs-à-pairs, structurés ou non, représentent l'aboutissement d'importants efforts de recherche en matière de distribution dans lesquels les nœuds ont des comportements extrêmement variés et flexibles.

Le contrôle d'accès est un moyen de restreindre l'accès à des ressources à un nombre limité d'utilisateurs ou a des conditions particulières. Il est important de pouvoir authentifier l'utilisateur (prouver son identité), autoriser (calculer les actions permises) et vérifier a posteriori que l'exécution était correcte. Le contrôle d'accès est globalement divisé en deux classes : le contrôle d'accès par capacité et le contrôle d'accès par listes. Dans le premier, une donnée supposée infalsifiable, l'accréditation, par exemple un mot de passe ou une clé de sécurité privée, est utilisée pour accéder à une ressource. Dans le second, les utilisateurs peuvent accéder à une ressource si leur identité est dans la liste correspondante. En ce qui concerne le contrôle d'accès distribué, la plupart des travaux [WABL94, MKKW99, JSS97, KFJ03, BFG07, GN08, MZZ+08, Aba09] se concentrent sur l'authentification des utilisateurs et sur les méthodes à base de capacité.

Le langage que nous présentons dans la section suivante est basé sur Datalog [AHV95]. Des versions distribuées de Datalog ont déjà été proposées [Hul89, NCW93, Hel10, GW10]. En général, un programme positif est distribué sur plusieurs pairs après une phase de compilation. Nous nous intéressons à un déploiement beaucoup plus dynamique, et nous introduisons en particulier la notion de délégation.

Un langage logique pour l'échange de données sur le Web

Ce travail a été réalisé en collaboration avec Serge Abiteboul, Meghyn Bienvenu, Marie-Christine Rousset et Emilien Antoine. Il est introduit dans [ABGR10] et présenté en détail dans [ABGA11].

La gestion d'information distribuée est un problème important, en particulier sur le Web. Des langages Datalog ont donc été proposés pour le modéliser. Nous introduisons ici un nouveau modèle, dans lequel des pairs autonomes échanges des messages et des règles (délégation). Nous étudions en particulier les conséquences sur l'expressivité de la délégation. Nous proposons aussi des restrictions du langage qui garantissent sa convergence.

Modèle

Considérons un pair *Alice-iPhone*, avec la relation *calendar* correspondant au calendrier de l'iPhone et la relation *Roc14members* correspondant à la liste des membres du club d'escalade Roc14. Voici des exemples de faits :

at Alice-iPhone:
calendar@Alice-iPhone(rockclimbing, 06/12/2011, Fontainebleau, Alice-iPhone).
Roc14members@Alice-iPhone(Bob, agenda, Bob-laptop).

La règle suivante ajoute les entrées relatives à l'escalade du calendrier d'Alice dans ceux des autres membres de Roc14 :

at Alice-iPhone:
$calendar@$peer(rockclimbing, $date, $place, Alice-iPhone) :-
calendar@Alice-iPhone(rockclimbing, $date, $place, Alice-iPhone),
Roc14members@Alice-iPhone($name, $calendar, $peer)

Il faut noter que les pairs et le nom des messages sont traités comme des données. La règle génère le nouveau fait suivant :

agenda@Bob-laptop(rockclimbing, 06/12/2011, Fontainebleau, Alice-iPhone)

Le fait décrit un message envoyé d'*Alice-iPhone* à *Bob-laptop*. Ce fait extensionnel est consommé par *Bob-laptop* lorsqu'il le lit. Comme dans les bases de données déductives, le modèle distingue entre faits extensionnels et faits intensionnels. Par exemple, la relation *Roc14members* peut être intensionnelle et définie ainsi :

at Alice-iPhone:
intensional Roc14members@Alice-iPhone(string, relation, peer)
Roc14members@Alice-iPhone($name, $relation, $peer) :-
contact@Alice-iPhone($name, $relation, $peer),
group@Alice-iPhone($name, Roc14)

La sémantique du système est basée sur une sémantique locale, standard et sur l'échange de faits et de règles. Intuitivement, un pair donné calcul un nouvel état depuis son état courant en consommant ses faits locaux et en déduisant à partir de ces faits et de la sémantique locale les faits qu'il doit envoyer aux autres et à lui-même, ainsi que les règles qu'il doit déléguer aux autres. Un exemple de délégation est le suivant. Considérons la règle suivante :

at Bob-laptop:
confirm@$peer(rockclimbing, $date, $place,Bob) :-
calendar@Bob-laptop(rockclimbing, $date, $place, $peer),
checkAvailability@Bob-iPhone($date);

L'effet de la règle, étant donné le fait généré à l'intention de Bob-laptop, est d'installer la règle suivante sur l'iPhone de Bob :

at Bob-iPhone:
confirm@Alice-iPhone(rockclimbing, 06/12/2011,Fontainebleau,Bob) :-
checkAvailability@Bob-iPhone(06/12/2011);

Lorsque l'iPhone de Bob exécute cette règle, en supposant que *confirm@Alice-iPhone* est extensionnel, il envoie le message suivant à Alice si *checkAvailability@ Bob-iPhone(06/12/2011)*:

confirm@Alice-iPhone(rockclimbing, 06/12/2011, Fontainebleau, Bob)

Si *confirm@Alice-iPhone* est intensionnel, c'est la règle suivante qui est envoyée

at Alice-iPhone:
confirm@Alice-iPhone(rockclimbing, 06/12/2011,Fontainebleau,Bob) :-

Sans rentrer dans les détails formels, il est intéressant d'étudier l'impact de la délégation sur l'expressivité du langage. En plus du langage général, note WL, on peut distinguer deux sous-langages. Le premier, SWL, restreint la délégation aux vues. Le second, SWL, interdit complètement la délégation. Enfin, il est intéressant de considérer les variantes autorisant les étiquetages temporels, notés WL^t, VWL^t et SWL^t respectivement. Les différences d'expressivité sont résumées sur la figure 1. Les inclusions sont strictes, à l'exception de celle de VWL^t dans VWL^t qui reste indéterminée.

Un autre point d'intérêt est la convergence du langage en fonction de l'ordre d'exécution des pairs. En règle générale, le résultat du calcul est a priori différent pour deux ordres d'exécution différents. Néanmoins, on peut isoler des cas monotones ou fortement stratifiés qui assurent la convergence, et ont une sémantique comparable à celle du cas où on centraliserait naturellement l'ensemble des règles et faits initiaux.

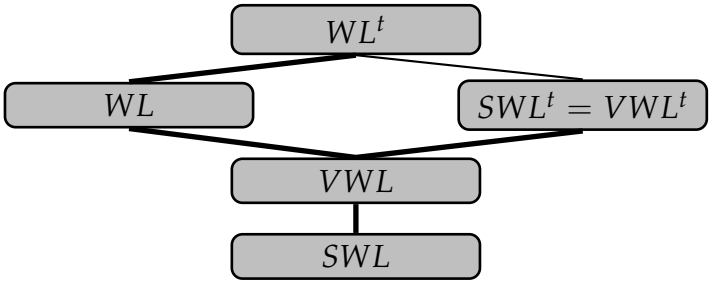

Figure 1: Expressivité des variantes de WL (les inclusions sont strictes quand l'arc est en gras)

Un modèle de donnée pour l'échange de données sur le Web

Ce travail a été réalisé en collaboration avec Serge Abiteboul, Neoklis Polyzotis et Amélie Marian. Il est présenté en détail dans [AGP11].

Sur le Web, le partage d'information se généralise, et les contraintes de gestion de données, en particulier de contrôle d'accès, s'accentuent. Le modèle que nous introduisons, WebdamExchange, permet de définir et partager l'information avec du contrôle d'accès dans un environnement distribué. Le modèle décrit une base de connaissance distribuée, qui contient des données et des connaissances sur les autres pairs, le contrôle d'accès, la localisation, la provenance des informations et en général toute sortes de connaissances utilisables directement par l'application, comme les ontologies. Elle contient aussi des règles qui définissent le comportement des pairs. Les principaux avantages de cette approche sont les suivants :

- **Large spectre :** le modèle permet de décrire des situations très différentes, allant du scénario centralisé au scénario le plus distribué, avec des pairs de confiance ou potentiellement malveillants, avec des informations en clair ou encryptées, ou des scénarios combinant ces différents cas

- **Modèle formel :** comme le modèle est formel, il peut être utilisé pour la vérification des systèmes ou pour le raisonnement automatique.

- **Contrôle de la qualité :** comme notre modèle prend en compte la provenance et le temps, ce qui permet d'améliorer le contrôle de la qualité de l'information.

La version du modèle présentée ici focalise sur le contrôle d'accès, représenté par des capacités (secrets et indices) et par des listes, ainsi que sur la distribution.

Modèle

WebdamExchange est bâtit sur WebdamLog. La notion de principal est étendue au principal virtuel, comme un utilisateur ou un groupe. Les relations sont aussi généralisées par un modèle semi-structuré. La notion élémentaire de WebdamExchange est l'assertion, un bloc de donné indépendant dans l'authenticité peut être vérifié. Par exemple, l'assertion :

Alice-iPhone *states* climbingSite@roc14={"id":"&cuvier", ...}
requester Alice *at* 18/12/2010

signifie que *Alice-iPhone* a crée localement une nouvelle version du document *climbingSite@roc14*. Dans le cadre du contrôle d'accès, cette assertion est annotée par une preuve d'authentification du principql *roc14*, qui démontre que *Alice-iPhone* et *Alice* peuvent écrire au nom de *roc14*. La provenance est aussi enregistrée via la mention du producteur Alice-iPhone et du demandeur Alice. Enfin, l'assertion est annotée par une étiquette temporelle. Le demandeur utilise une instruction pour déclencher la fabrication de l'assertion par le producteur. Cette instruction est authentifiée par le demandeur.

Le modèle comporte un jeu complet d'assertion et les instructions correspondantes pour les collections, les capacités (par exemple les clés publiques et privées), les listes de contrôle d'accès et la localisation. Ces assertions peuvent être échangées entre pairs, en les encapsulant dans des couches d'authentification qui capture la provenance. Enfin, des règles sur le modèle de celles de WebdamLog peuvent être utilisées à la fois comme connaissance échangée dans le système ainsi que pour décrire le système.

L'implémentation physique du contrôle d'accès est basé sur différents types d'assertion de capacités, qui représente soit les clés RSA, soit des mots de passes par exemple, permettant de décrire les différentes situations rencontrées sur le Web. De même, les assertions de localisation permettent de décrire les différents types de distribution.

Propriétés et scénarios

La correction (les messages sont bien formés), la sûreté (l'accès aux données est réservé aux ayant-droit), la complétude (l'accès à toutes les données est

possible) et la consistance (l'accès aux données ne dépend pas du pair) sont les propriétés fondamentales du droit d'accès. Garantri ces propriétés est difficile en général. Dans des scénarios particuliers comme @home (toutes les données sont stockées sur un pair de confiance), @friend (toutes les données sont stockées chez les ayant-droits), @host (toutes les données sont stockées encryptées sur un pair), @host-dht (toutes les données sont stockées encryptées sur une DHT), on peut néanmoins garantir certaines de ces propriétés sous certaines conditions.

Le système WebdamExchange

Ce travail a été réalisé en collaboration avec Serge Abiteboul, Neoklis Polyzotis, Amélie Marian, Emilien Antoine et Kristian Lyngbaek. Il est présenté en détail dans [AGP11] et [AGL+11].

Nous avons implémenté un système WebdamExchange, comme preuve de concept de notre modèle. Ce système supporte le modèle de donnée de WebdamExchange, avec différents types de capacités (RSA, URL, mot de passe). Les pairs principaux du système sont programmés en Java et communique via des services Web. Ils peuvent aussi interagir avec le Web, en particulier avec des sites comme Facebook ou des systèmes comme Pastry. Ils peuvent aussi communiquer avec des clients logiciels légers, qui permettent à une application de réseau social pour les escaladeurs ou un iPhone de communiquer avec le système.

La démonstration de ce prototype est basé sur le scénario présenté précédemment. Il permet à des escaladeurs d'échanger des données sur le Web sans se soucier de l'hétérogéinité du contrôle dáccés et de la distribution de leur données.

Conclusion

Dans cette thèse, nous avons présenté successivement un langage pour la gestion de données distribuées, avec une sémantique claire permettant la délégation de tâche, un modèle de donnés pour une base de connaissance intégrant des informations de distribution et de contrôle d'accès, et un système basé sur les contributions précédentes.

Il reste néanmoins de nombreux points ouverts, en particulier en ce qui concerne la non-monotonie, l'intégration de données, la gestion de la croyance, l'apprentissage automatique ou les données intensionnelles. Globalement, ce travail illustre qu'il est devenu indispensable de fournir un

moyen de gérer les données hétérogènes, qui soit à la fois efficace et flexible. Le travail présenté contribue ainsi à permettre au programmeur de se concentrer sur le développement de nouvelles applications et fonctionnalités en s'appuyant sur un modèle robuste et expressif.

Chapter 1

Introduction

The amount of information on the Web is spreading very rapidly. Users as well as companies bring data to the network and are willing to share with others. They quickly reach a situation where their information is hosted on many machines they own and on a large number of autonomous systems where they have accounts. Management of all this information is rapidly becoming beyond human expertise. In particular, the emergence of Web 2.0 and social network applications has enabled more and more users to share sensitive information over the Web. The information they manipulate has many facets: personal data (e.g., pictures, movies, music, emails), social data (e.g., annotations, recommendations, contacts), localization information (e.g., bookmarks), access information (e.g., login, keys), Web services (e.g., legacy data, search engines), access rights, ontologies, beliefs, time and provenance information, etc. The tasks they perform are very diverse: search, query, update, authentication, data extraction, etc. We argue in this thesis that all this should be viewed in the holistic context of the management of a distributed knowledge base. Furthermore, we also propose that datalog (and some extensions) forms the sound formal basis for representing such information and supporting these tasks. This work is part of the ERC project *Webdam* [Web] on the foundations of Web data management. The goal of the project is to participate in the development of unified formal foundations for distributed data management, since the current lack of such foundations is hindering progress in Web data management.

Contributions

The contributions of the thesis are as follows:

1. We introduce *WebdamExchange*, a novel distributed knowledge-base model that includes logical statements for specifying information, access control, secrets, distribution, and knowledge about other peers. These statements can be communicated, replicated, queried, and updated, while keeping track of time and provenance. The resulting knowledge guides distributed data management. We argue that the complexity of modern distributed data management requires the support of such a model capturing many facets of this management, that are typically considered in isolation. We demonstrate the generality and flexibility of the *WebdamExchange* model by showing that very different and common schemes of access control and distribution can be specified in the model.

2. We present *WebdamLog*, a new rule-based language for distributed data management that combines in a formal setting deductive rules as in Datalog with negation, (to specify intensional data) and active rules as in Datalog$^{\neg\neg}$ (for updates and communications). The model provides a novel setting with a strong emphasis on dynamicity and interactions (in a Web 2.0 style). Because the model is powerful, it provides a clean basis for the specification of complex distributed applications. Because it is simple, it provides a formal framework for studying many facets of the problem such as distribution, concurrency, and expressivity in the context of distributed autonomous peers.

3. We also mention the implementation of a proof-of-concept system that handles all the components of the knowledge base and some experiments with a lighter system designed for smartphones.

We believe that these contributions form a good foundation to overcome problems commonly encountered for Web data exchange, in particular for Social Networks.

WebdamExchange

Of course, there have already been lots of works on the specification and sharing of information in a distributed environment. See Chapter 3. In particular, ontologies are often considered to handle the heterogeneity of vocabulary and data organization of different participants. Although ontology statements can naturally be handled in our knowledge-base and fit nicely with our rule-based language, we are mostly concerned with

different kinds of heterogeneity: that of the policies adopted for distribution and access control. A major challenge was to abstract away many details of the existing technology while preserving the essence of these two aspects. To prove that we succeeded, we demonstrate the generality and the flexibility of the *WebdamExchange* model. More precisely we discuss how very different and commonly found schemes can be captured in the model using fundamental pieces of knowledge it includes.

Since such different schemes can all be described in the *WebdamExchange* model, our work opens the way for addressing applications that combine them as well as other natural ways of managing distributed information in arbitrarily rich ways, which is the reality of today's Web. The use of a formally defined knowledge base simplifies the verification of desirable properties of applications (e.g., prevention of information leaks) and the monitoring of the global information system. Also, because the model includes time and provenance information, the quality of data can better be controlled, and diagnoses are easier to perform in case of malfunction.

WebdamLog

The management of our knowledge base, in particular the specification of the different policies, is a challenging problem. Because of the complexity of the management of modern distributed information, there has been a recent trend towards using high-level Datalog-style rules to specify applications in a Web setting. *WebdamLog* is in this spirit. It is a new model for distributed computation where peers exchange messages (i.e., logical facts) as well as rules. The model is specially well adapted to the *WebdamExchange* model, that is deeply rooted in this formal ground.

The model provides a well founded support for reactions to changes in evolving environments using *delegation*, a special mechanism we introduce to install rules on other peers. In particular, it generalizes *remote materialized view* and allows a peer to delegate work to other peers, in an ActiveXML style. The model therefore opens a new avenue for the specification of data exchanges between autonomous peers, that we illustrate with different examples. We also study the impact of delegation and time on expressivity. Finally we study two sub-languages that guarantee some form of convergence.

Organization

The thesis is organized as follows. We first motivate further this work with an illustrative example in Chapter 2. We propose a summary of the state of the art in Chapter 3. In Chapter 4, we present *WebdamLog*, a rule-based language to support the distributed knowledge base. In Chapter 5, we present *WebdamExchange*, a distributed knowledge base model to handle heterogeneous access control and distribution policies. In Chapter 6, we also discuss an implementation of the *WebdamExchange* system. We finally briefly review some others research works we achieved during our thesis in Chapter 7. We conclude in Chapter 8.

Chapter 2

Motivating Example

In this chapter, we introduce a motivating example. This example is used as the Ariadne's thread of this thesis. Let us imagine a group of rock-climbers who want to organize regular outings in the Fontainebleau forest (in France, close to Paris).

Today's world

Some of these rock-climbers maintain data on the bouldering area in the forest. The group may have a blog where members describe the last outings. Alice may have a list of the rocks she has already climbed available through an application on her iPhone. Bob may have created some Google maps that describe access to the bouldering areas. Some data may also be available on a peer to peer network, for example an Excel file with the list of all the rocks and their difficulties. Other data are delivered through Web-services, for example weather forecasts for a given GPS position.

Suppose Alice wants to organize outings in Fontainebleau. She connects to the Web from her smartphone. She first looks in a social network server, such as Facebook, and finds which members of her rock-climbing club of Paris 14th district, Roc14, live near Fontainebleau and publish Web pages or documents on rock-climbing. She then wants to perform some full text "search" on these data. Since the data is distributed in a very heterogeneous way, this simple task therefore induces a large number of knowledge management tasks: localizing resources, finding whether she has access to them, using or obtaining some secrets (password or encryption keys) to access them, accessing data and indexes, and integrating this external data to her own knowledge base.

Alice of course is also facing problems of data extraction and data

integration. There are numerous works on these topics, and we rather focus on a less obvious but as difficult problem: how does Alice access this distributed data, that reside in heterogenous system? Indeed, part of the data is managed by a blog application, part is on Alice's iPhone, part is on a P2P network, that all have different kinds of distribution protocols. Furthermore, some of the data may have restricted access. For example, only members of the group, identified by an account with a login and a password, are allowed to post on the group's blog. Moreover, distribution and access control can not be considered totally separately. For example, Bob may also control access to his maps by distributing the private url of the maps to his friends only. Observe that Web users already see data location and secret keys as information, e.g., Delicious or Mozilla Sync.

One may argue than a unique trusted peer, such as Facebook, should be in charge of all the data. We personally believe that a better solution would be to describe these different data as statements in the same universal knowledge base and perform some distributed reasoning on this knowledge base to help support data management tasks. Using our model, called *WebdamExchange*, the users are able to use different systems to host their data, with different access control policies. They use the *WebdamExchange* model to transparently navigate and integrate their information while keeping total control on their data.

Ideal world

Let us now describe the motivating example further, by abandoning the current real world of the Web for an ideal world.

Suppose Alice uses the *WebdamExchange* system on a regular basis. The data of Alice is distributed using different schemes (See Figure 2.1). Alice hosts some picture on her laptop which is a real *WebdamExchange* peer. She hosts her list of contacts on her smartphone, that communicates with the *WebdamExchange* system using a client application installed on the smartphone. Of course, the data on her smartphone are usually not accessible from the Web and her laptop is not running all the time, so she also replicates her data on an untrusted DHT, SomeDHT. She also has a favorite social network Website, Facebook, and stores her profile information on this trusted peer, using a wrapper.

Some of the friends of Alice are also part of the *WebdamExchange* system, and their data are distributed using similar schemes. Since they are interested in some of her data, that they frequently use, they replicate it locally. Indeed, this unstructured P2P distribution is natural since people

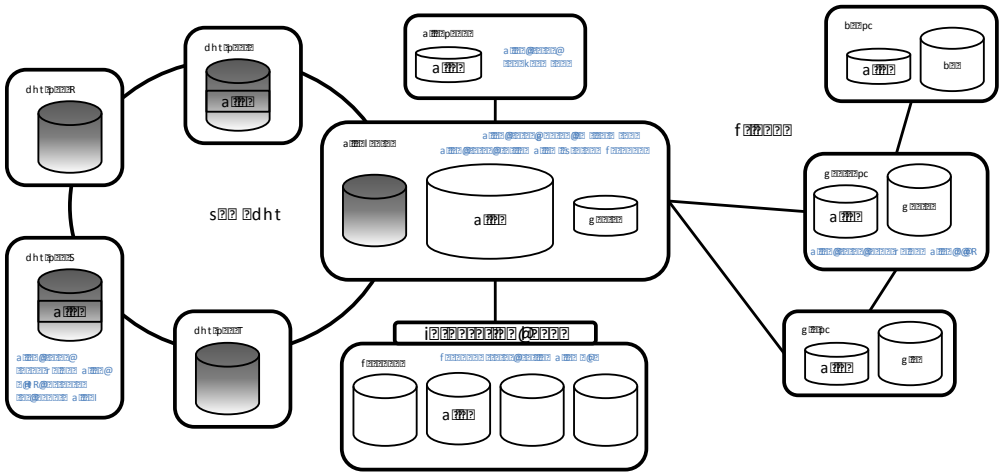

Figure 2.1: The distribution of Alice's data

prefer to store data they care about and to interact mostly with friends.

Some of the friends of Alice still use old-fashionned systems, like Blogs or Facebook, to share their data. Their data are available for the *WebdamExchange* system using special wrappers, and they access the *WebdamExchange* system using a client Web GUI or by letting *WebdamExchange* update data of their favourite systems.

Recall that Alice wants to organize outings in Fontainebleau, with members of Roc14, without worrying about issues such as access control and distribution. The goal of *WebdamExchange* is to automatically manage these problems. In particular, Alice is able to query the whole system (even the part accessed using wrappers) as a unified view. The system automatically is in charge of the execution of the query, by finding the localization of the data and the needed credentials, and checking access rights. In particular, some peers automatically send subtasks to other peers, using a rule based language called *WebdamLog*.

Alice also wants to be able to deploy easily specific applications. For example, she wants to create a collaborative calendar where users may propose outing dates. This collaborative calendar uses rules that compute the best outing dates. It checks availability of the users without disclosing their calendar and composes carpooling groups. Alice needs to delegate part of the work of the application, for example to enforce that the calendars are not disclosed. *WebdamExchange* will clearly ease the writing of such

application, since it is based on *WebdamLog*, a rule-based distributed language with delegation.

Challenges

The scenario described above raises several difficulties. We are particularly interested by the following ones.

Access control The management of access control is the most critical requirement. In order to protect her data, Alice specifies different kinds of access rights, in particular read, write, append, remove, own rights. The system should not permit illegal operations on the data. Unfortunately, it is not possible to prevent people from misbehaving outside of the system, e.g., a user may illegally send confidential data to another user. But we want to detect illegal operations resulting from such kinds of behavior whenever it is possible. To do so, the system needs to keep a full trace of the provenance, and support distributed monitoring.

Distribution As already discussed, an important problem on the Web is the wide dispersion of the data. In particular, the data is spread between different peers, using different distribution schemes. In our example, Alice uses different schemes at the same time. She also uses peer that are outside of the *WebdamExchange* realm. A real system may even be more complex. Alice may have data on several social network Websites (Facebook, Linked-In, MySpace). She may have several personal computers (at home, at work) and several mobile devices (smartphone, tablet). In such a highly distributed setting, the system has to provide a unified view of the access control and the localization of the data. It also has to automatize as much of the work as possible.

Data manipulation In this setting, providing a way to edit, navigate and query data is clearly necessary. The user should be able to localize all the data she has access to and to ask global structured queries to the system. The user also needs to delegate some tasks she cannot perform herself. A rule-based language such as *WebdamLog* is a sound foundation for such a need.

There is already a large amount of work on access control and distributed data management. In the next chapter, we discuss the state of the

art insisting on the shortcomings that *WebdamExchange* and *WebdamLog* fix at least in part.

Chapter 3

State of the Art

This thesis is mostly related to two important domains of computer science, distributed information systems and access control. Indeed, we are interested in building a distributed knowledge-base system that is aware of distribution and access control policies. We discuss next these two domains.

3.1 Distributed Information Systems

Distributed information systems are now a well developed area of computer science, covered by a large number of reviews and books. See, e.g., [ÖV99, AMR$^+$ar]. In the next discussion, we discuss the most essential notions for this thesis. We first present general properties of distributed systems, then review successively database, Web data and peer-to-peer distributions.

3.1.1 Distributed systems

[AMR$^+$ar] defines a distributed system as some software that serves to coordinate the actions of several computers. This coordination is achieved by exchanging messages, i.e., pieces of data conveying information. The system relies on a network that connects the computers and handles the routing of messages.

Distributed systems are characterized by the following desirable properties:

- *consistency* [DHJ$^+$07] denotes the ability of a distributed system to give the same answer to a client regardless of the server it is connected to.

- *reliability* [Bir05] denotes the ability of a distributed system to deliver its services even when one or several of its software or hardware components fail,
- *availability* denotes the ability of a distributed system to limit as much as possible the latency due to the replacement of a faulty component.
- *scalability* [MMSW07] denotes the ability of a distributed system to continuously evolve in order to support a growing amount of tasks and data. In general, one is interested by linear scalability, i.e., a growing of the system resources proportional to that of the tasks and data.
- *efficiency* denotes the ability of a distributed system to minimize the response time (when the first item is delivered) and to maximize the throughput (the number of items delivered by unit of time)

One is typically facing a trade-off between these properties. In particular, the CAP theorem [GL02] states that a distributed system cannot provides simultaneously consistency, availability and arbitrary message loss. This last result is of particular importance for us, since we aims at providing some precise security results for our system, but also consistency guarantees.

3.1.2 Distributed Databases

[ÖV99] defines a distributed database as a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users. It provides a shared structure among the data, and an access via a common interface. Distributed DBMSs are intended to provide data independence, network transparency, replication transparency and fragmentation transparency. Indeed, distributed DBMSs improve reliability by replicating components, thereby eliminating single points of failure, while letting the user ignore distribution issues.

[ÖV99] describes the architecture of a distributed DBMS by characterizing the autonomy of local systems (tight integration, semi-autonomy and total isolation), their distribution (no distribution, client-server or peer-to-peer) and their heterogeneity (homogeneous or heterogeneous system).

An important component of the design is the placement of data, that can be investigated along three orthogonal dimensions [LM75]: level of

sharing (no sharing, data sharing, data and program sharing), access pattern (static or dynamic) and level of knowledge on access pattern (partial or complete information). The main issues for placement of data is the type of fragmentation (horizontal [CNP82] or vertical [NCWD84, SW85]) and allocation [DF82, CL82] of these fragments to sites in the network. The main difficulties are to define a replication policy that enforces reliability with a good efficiency, to manage failure of nodes and recovery, and to synchronize transactions.

Since we are mostly interested by systems where servers are highly independent, we refer the reader to [ÖV99] for more details on distributed databases.

3.1.3 Data on the Web

With the development of *Internet* [RFC74] and *HTML* [W3C99a], the *Web* [BLC90] rapidly became an essential way of data distribution. This position was further strengthen by the development of *XML* [W3C08a], that highly eases exchange and integration of data. The World Wide Web Consortium, that is in charge of promoting and developing XML usage, proposed a wide range of standards for typing [W3C04], querying [W3C10] or transforming [W3C99b] XML. There is now a large number of books discussing aspects of Web's data. See, e.g., [AMR$^+$ar, ABS00].

As his founder Tim Berners-Lee foresaw, the Web is also developing a layer of semantics on top of XML, using *ontology* languages such as OWL [W3C09] to facilitate data integration. More formal analysis of these languages can be found in [AH08, AvH08]. Integration also benefits of the large amount of work on *mediation*. See [AMR$^+$ar] for a survey.

Finally, the development of *Web-services* gave a basic infrastructure for distributed Web data management. This infrastructure is based on XML standards such as SOAP [W3C07a], WSDL [W3C07b] and UDDI [OAS04] and does not aims at providing complex functionality. Nevertheless, additional standard are used to express services *workflows*, e.g., BPEL [OAS07] and *orchestration* of services, e.g WSCL [W3C02a]. Some models such as *ActiveXML* [ABM08] aim at providing a clean model for intensional data (e.g., obtained by service calls) on the Web and distributed data intensive applications. Rules are used in ActiveXML, e.g., in [ASV09], but they are different of the one used in *WebdamLog*, that we will present further, because the data are XML trees.

To summarize, the Web is now a standard way of sharing and managing data. Our work, as part of the *Webdam* Project [Web], aims at providing

better foundations for two kinds of data management problems, distribution and access control. Our model is therefore as close as possible of the Web organization and data and our implementation largely relies on the standard models and tools afore-mentioned.

3.1.4 Peer-to-Peer Systems

A *peer to peer* (P2P) network (See, e.g., the surveys in [TS04, Wal03]) is a large network of nodes, called *peers*, that are both clients and servers and that agree to cooperate in order to achieve a particular task. It is a particular kind of distributed systems which assumes that the organization of the nodes is loose and flexible. Indeed, the peers are assumed to be highly autonomous, choosing when they participate to the network and how much resource (CPU, memory, ...) they provide to the system. It is also often assumed they use an *overlay network*, i.e., a graph of connections laid over a physical infrastructure, e.g., the Internet.

A general search technique on this kind of networks is *flooding*: a peer disseminates its request to all its friends, that flood in turn their own friends. One may also use other forms of *gossipping*, for example by choosing randomly only one friend to propagate the request. This kind of P2P networks are called *unstructured*. There are more structured ways for searching for information in the network (*structured P2P networks*), such as distributed hash tables or distributed search trees. We are particularly interested in this thesis by describing these different distribution policies in a unified framework. So we briefly discuss next the main structured P2P network technologies.

Distributed Hash Tables The hash table is a particular kind of index structure, that associates a key to a location in constant time. In addition to the search operation, the index supports insertion and deletion of new entries. A main difficulty of the hash table technology is to choose the trade-off between space and collision. Indeed, two keys may get the same location and the collision has to be managed, usually by chaining this data. When too much collisions occur, the hash table is usually dynamically redistributed. One efficient way to do so is linear hashing [Lit80, LNS96], that allows a linear growth of the table. The mapping of a hash table to a cluster of machine is relatively straightforward, and is largely improved by methods such as consistent hashing [KLL$^+$97, DHJ$^+$07]. In particular, some XML databases have been successfully deployed on top of distributed hash table [AMP05, AMP$^+$08].

Distributed search trees The main problem of hash tables is that they do not support range queries or nearest-neighbors searches. This is a well-known limitation that justifies the coexistence, in centralized systems, of hash tables and tree indexes. This kind of structure can also be distributed by carefully placing parts of the index on the peers [LNS94, KW94], for example using routing tables [JOV05] or special kind of meta-data tablets [CDG+08].

3.2 Access control

3.2.1 Principles of access control

Access control provides the means to control access to *resources* in a given computer-based information system. Access control allows some authority to decide who can do what on some particular pieces of information. The entities that can perform actions in the system are called *principals*. Access control includes *authentication* (who is the principal), *authorization* (what can be done by the principal) and *audit* (correctness of what has been done by the principal). It may also include measures such as physical devices, including biometric scans, digital signatures, encryption, social barriers...

Access control models used by current systems tend to fall into one of two classes: those based on *capabilities* and those based on *access control lists* (ACLs). In a capability-based model, holding an unforgeable reference or capability to a resource provides access to the resource. For example, holding a decryption key is the condition to access the content of a document. Access is granted to another party by transmitting such a capability over a secure channel. To revoke access, the capability used to protect the resource has to be changed, since it is not possible to un-send a capability. In an ACL-based model, a principal's access to a resource depends on whether its identity is on a list associated with the object. Access is granted and revoked by editing the list, which may have its own ACL. We are interested by both kinds of models and the *WebdamExchange* model captures them both.

A *security policy* is a definition of what it means to be secure for a system. The security policy addresses constraints on functions and flow among them and constraints on access to resources by principals, in particular by programs, external systems and adversaries. These formal policy models can be categorized into the core security principles of: confidentiality (the data is only accessed by principals who have access), integrity (the data can fully be accessed by principals who have access) and availability (the data

can always be accessed by principals who have access). The protection of a system is the enforcement of a security policy by organizational policies or security mechanisms. Given a security policy, a particular implementation can be secure or insecure.

For operating systems In *operating systems*, principals are usually users, groups of users or processes (or, in most evolved cases, users using a process). The access control is usually defined with ACLs, the simple case of Unix providing read, write and execute right to a user, a group and other principals respectively. It corresponds to the rather limited `chmod` shell command. More developed languages have been proposed to define more refined policies, e.g., [HRU76]. It is important to note that for some of these languages, the security of a system given a policy may be undecidable.

For databases A possible approach to access control, specific to *databases*, is based on views, used as protection mechanisms. See, e.g., [FSW81, HKM78]. Also, the data manipulation language (such as SQL) may be directly used to define the privileges (insert, delete, update and read) of users and groups. The authorization are stored as particular tables or views of the database.

With respect to authorization control in classical DBMS, distributed DBMSs have to provide remote user authentication, management of distributed authorization rules and handling of views and user groups. See, e.g., [WL82]

3.2.2 Access control for distributed systems

Distributed file systems is another major field of research for access control. Early works [WABL94, MKKW99] considered primarily authentication of users and servers in a distributed system. In [WABL94], a logic is used to validate chains of authentication in a distributed setting without considering data. In [MKKW99], self certification of pathnames is used to authenticate servers, with the remaining of key management handled outside of the file system, under the responsibility of agents.

Another important branch of work is the numerous logical access control policy languages for distributed systems proposed recently, e.g., [JSS97, KFJ03, BFG07, GN08, MZZ+08, Aba09]. The work of [BFG07] and [Aba09] are probably the closest to our work. [BFG07] proposes a declarative authorization language where policies and credentials are expressed using predicates defined by logical clauses. Access requests are mapped to logical

authorization queries, and access is granted if the query succeeds against the current database of clauses. [Aba09] uses also a logical description of the policy, with statements that are semantically mapped to logical clauses as well. *WebdamExchange* language is highly inspired from this work, in particular borrowing denotation and syntax. The original language of [Aba09] introduces interesting new features and formalism for authentication, as principal composition and delegation. These notion are mostly orthogonal with access control authorization. So we choose to concentrate first on integration of distribution in the language and gave up the most advanced features. Nevertheless, the original language can be mapped to Datalog, so we are confident in the fact that *WebdamExchange* on top of *WebdamLog* can be easily upgraded with these authentication features.

Based on the idea of separating key management and data management, a large numbers of systems have been proposed to share file in a P2P network. See, e.g., [KBC+00, RD01, REG+03]. The focus of these works are generally on distribution and caching, limiting security issues to encryption and external key management. For example, the system described in [RD01] uses the PASTRY routing substrate, replica diversion and caching to balance query load. In [REG+03], a distributed routing index not constraining data placement is used. Redundancy, cryptography and monitoring are used to build the storage system on an untrusted infrastructure. By contrast with these approaches, *WebdamExchange* covers both key and data management and different kinds of data distribution.

3.2.3 Access control for XML

There is also a large amount of work on defining an access control language for XML, e.g., [DdVPS02, Bry05, MTKH06, FM07]. Extending the issues encountered with standard databases, the problem is mostly to properly take in account the specific granularity of XML. Indeed, one may want to give access to sub-parts of XML trees. A typical example is the following. Suppose given an XML document with some "store" element, containing "book" and "employee" elements. Suppose each book has a "price", "description" and number of book in "stock" elements. A client should be able to access the book prices and descriptions, without seeing the employees or the stock. It corresponds to granting access to the sub-trees rooted at "/store/book", while preserving the paths to these nodes, but not the sub-trees rooted at "/store/book/stock". The languages are then usually based on access paths, with different kinds of choices regarding how to grant or revoke access to the node itself, the path to the node, and the sub-tree, and

how to manage conflicts in policy.

Based on works on enforcing access control on distributed systems using cryptography [CGI+99, RR02], the work presented in [MS02, MS03, AW08] proposes a way to enforce some particular kind of access control policies for XML by encrypting parts of the document and by inserting special meta-data containing encryption of keys.

In *WebdamExchange*, the granularity for access control is the document. This is to simplify, since one could clearly consider other granularities.

3.2.4 Access control policies

In this section, we are mentioning interesting aspects of specific access control policies. We are particularly interested in access control policies related to personal data, since users have a huge need for privacy and control over these data, that is far from fulfilled by current systems.

Hippocratic database Hippocratic databases [AKSX02] are systems that respect a certain number of principles, in particular purpose specification, limited collection, i.e., data accessed should be limited to the minimum necessary for accomplishing the specified purpose, and limited retention, i.e., data should be retained only as long as necessary for the purpose fulfillment. It has been designed with personal data management in mind. Some proposals have been made to formalize [LAE+04] and implement [JSAV09] such databases. Hippocratic databases are of particular interest to illustrate how complex the definition of a security policy can be, and how far current approaches are to meet the needs.

ActiveXML-based There have been some preliminary works on managing personal information using ActiveXML, e.g., [SHLX03, ABCM04, AAB+04, Kim10] These works illustrate, on the example of an electronic patient record, how ActiveXML can be used to automatically manage distributed information with respect to security and access control. *WebdamExchange* follows this line of work, that aims to addressing in a uniform way distributed query processing and security. In [ABCM04], access control is enforced by special service calls and a simple query rewriting mechanism. One of the important novelty of *WebdamExchange* compared to that work is that it allows the system to use standard cryptography methods as well, in order to enforce access control.

Social networks *WebdamExchange* was initially motivated by the idea of implementing a social network in P2P. To the best of our knowledge, there are few works on the topic, but the topic clearly became more active recently. For example, [KGG+06] proposed a distributed identity management with access control based on the social network of users. In particular, it uses the standard Friend-Of-A-Friend (FOAF) [BM10] representation of the social network. An access control policy model based on the social network and trust has also been proposed by [AVM07]. These two works illustrate well specific needs of social network, in particular with respect to transitive closure on graphs. On another direction, [FAZ09] proposes a language for social network access control and uses it to analyze Facebook security policy.

Finally, [BSVD09] proposes an implementation of a social network based on a distributed hash table and partially addresses privacy. In general, such approaches, that focus on particular kind of policies, are clearly complementary to ours.

3.2.5 Other problems related to access control

There is a large number of problems related to access control. One is protection of anonymity, i.e., to forbid an opponent to guess who edited or read a data. See, e.g., [CKGS98, CSWH01].

There is also an important line of works about security in statistical database. See, e.g., [YC77, Bec80, FJ02]. In particular, an adversary may try to guess some data he has not access to, by using a set of queries and some external knowledge. There are two main methods to avoid that problem. One is to monitor queries and refuse to answer a query if it will provide forbidden information (interface control). Another one is to slightly modify the database or the answer to the query (data swapping and multidimensional transformation) to have as little impact as possible on the aggregated answers but to give no information on single rows.

3.3 Distributed Datalog

To support *WebdamExchange*, we will use the *WebdamLog* language. Note that this language participates in the renewed interest in datalog. See, e.g., [Dat10]. Datalog has been the subject of a large amount of work in the database community. Some of it is reviewed in [AHV95]. To our knowledge, the first attempts to distribute Datalog on different peers are [Hul89]

and [NCW93]. The first distributes a positive Datalog program on different machines after a compilation phase. The second adapts classical transformations of positive programs based on semi-joins to minimize distribution cost. Perhaps the work closest to the *WebdamLog* model is [AAHM05] that adapts query-subquery optimization [Vie86] to a variant of positive distributed Datalog. We are also interested by negation, in particular by stratified negation [CH85], and by active rules in the style of Datalog$^{\neg\neg}$ [AV91]

We believe that the most interesting usage of Datalog-style rules for distributed data management came recently from the Berkeley and U. Penn groups. They used distributed versions of Datalog to implement Web routers [LHSR05], DHT [LCH$^+$05] and Map-Reduce [ACC$^+$10] rather efficiently. By demonstrating what could be efficiently achieved with this approach, these works were essential motivations for our own. The most elaborate variant of distributed Datalog used in these works is presented in [LHSR05, CCHM08] and formally specified in [NR09, PRS09]. In these papers, the semantics is operational and based on a distribution of the program before the execution. In view of issues with this model, a new model was recently introduced in [Hel10], based on an explicit time constructor. We found the semantics of negation together with the use of time in that model rather unnatural. In particular, time is used as an abstract logical notion to control execution steps and the future may have influence on the past. As a consequence, we found it difficult to understand what applications are doing as well as to prove results on their language. However, we have been influenced by this line of work, and other recent works such as [GW10].

Chapter 4

A rule based language for Web data exchanges

This work has been carried out in collaboration with Serge Abiteboul, Meghyn Bienvenu, Marie-Christine Rousset and Émilien Antoine. It is introduced in [ABGR10] and presented in [ABGA11].

The management of modern distributed information, notably on the Web, is a challenging problem. Because of its complexity, there has recently been a trend towards using high-level Datalog-style rules to specify such applications. We introduce here a model for distributed computation where peers exchange messages (i.e., logical facts) as well as rules. The model provides a novel setting with a strong emphasis on dynamicity and interactions (in a Web 2.0 style). Because the model is powerful, it provides a clean basis for the specification of complex distributed applications. Because it is simple, it provides a formal framework for studying many facets of the problem such as distribution, concurrency, and expressivity in the context of distributed autonomous peers.

As mentionned in the previous chapter, there has been renewed interest in studying languages in the Datalog family for a wide range of applications ranging from program analysis, to security and privacy protocols, to natural language processing, or multiplayer games. For references, see [Hel10] and the proceedings of the Datalog 2.0 workshop [Dat10]. Here, we are concerned with using rule-based languages for the management of data in distributed settings, as in Web applications [ABM04, ASV09, FMS09] and [ABGR10], networking [LCG+06, LMO+08, GW10] or distributed systems [LCG+09]. The arguments in favor of using Datalog-style specifications for complex distributed applications are the familiar ones. See, e.g., [Hel10].

A main contribution of this work is a new model for distributed data management that combines in a formal setting deductive rules as in Datalog with negation [CH85] (to specify intensional data) and active rules as in Datalog$^{\neg\neg}$ [AV91] (for updates and communications). There have already been a number of proposals for combining active and deductive features in a rule-based language; see [LLM98, Hel10] and our discussion of related work. However, there is yet to be a consensus on the most appropriate such language. We therefore believe that there is a need to continue investigating novel language features adapted to modern data management and to formally study the properties of the resulting new models.

The language we introduce, called *WebdamLog*, is tailored to facilitate the specification of data exchange between autonomous peers, which is essential to the applications we have in mind. Towards that goal, the new feature we introduce is *delegation*, that is, the possibility of installing a rule at another peer. In its simplest form, delegation is essentially a *remote materialized view*. In its general form, it allows peers to exchange rules, i.e., knowledge beyond simple facts, and thereby provides the means for a peer to delegate work to other peers, in Active XML style [ABM08]. We show using examples that because of delegation, the model is particularly well suited for distributed applications, providing support for reactions to changes in evolving environments.

A key contribution is a study of the impact of delegation on expressivity. We show that view delegation (delegation in its simplest form, allowing only the specification of materialized views) strictly augments the power of the language. We also prove that full delegation further increases it. These results demonstrate the power of exchanging rules in addition to facts.

A message sent from peer p, received at peer q, that starts some task at q, introduces a kind of synchronization between the two peers. Thus, time implicitly plays an important role in the model. We show that when explicit time is allowed (each peer having its local time), view delegation no longer increases the expressive power of the language.

Because of their asynchronous nature, distributed applications in *WebdamLog* are nondeterministic in general. To validate our semantics for deductive rules, we study two kinds of systems that guarantee a form of convergence (even in presence of certain updates). These are positive systems (positive rules and persistence of extensional facts) and strongly-stratified systems (allowing a particular kind of stratified negation [CH85] for restricted deductive rules and fixed extensional facts). We also show that both types of systems essentially behave like the corresponding centralized systems.

Organization

The chapter is organized as follows. We introduce the model in Section 4.1, first by means of examples and then formally. In the following section, we discuss some key features of the model and illustrate them with more examples. In Section 4.3, we compare the expressivity of different variants of our model. In Section 4.4, we discuss the convergence of *WebdamLog* systems and compare the semantics to the "centralized semantics", for the positive and strongly-stratified restrictions of the language. In Section 4.5, we mention an implementation and optimization techniques. The final section concludes with directions for future work.

4.1 The model

In this section, we first illustrate the model with examples, then formalize it. More examples and a discussion of key issues will be provided in the next section.

4.1.1 Informal presentation

We introduce with a first example the main concepts of the model: the notions of *fact* that captures both *local tuples* and *messages* between peers, of *extensional* and *intensional* data, and of *(WebdamLog)) rule*.

Consider a particular peer, namely *Alice-iPhone*, with the relation *calendar* that gives the calendar entry that Alice entered from her iPhone and the relation *Roc14members* that gives the list of member of the Roc14 climbing group and how to send them calendar invitation (on which servers, with which messages). Examples of facts are:

at Alice-iPhone:
calendar@Alice-iPhone(rockclimbing, 06/12/2011, Fontainebleau, Alice-iPhone).
Roc14members@Alice-iPhone(Bob, agenda, Bob-laptop).

The following rule, called [*Send-Invitation*] , is used to include rockclimbing entries from Alice's agenda into the agendas of other members of the group, and in particular into Bob's agenda:

at Alice-iPhone:
$calendar@$peer(rockclimbing, $date, $place, Alice-iPhone) :-
calendar@Alice-iPhone(rockclimbing, $date, $place, Alice-iPhone),
Roc14members@Alice-iPhone($name, $calendar, $peer)

Observe that peer and message names are treated as data. The two previous facts represent pieces of local knowledge of *Alice-iPhone*. Now consider the new fact generated by the rule:

agenda@Bob-laptop(rockclimbing, 06/12/2011, Fontainebleau,
Alice-iPhone)

This fact describes a message that is sent from *Alice-iPhone* to *Bob-laptop*.

As in deductive databases, the model distinguishes between extensional relations that are defined by a finite set of ground facts and intensional relations that are defined by rules. So for instance, the relation *Roc14members* on Alice-iPhone may be intensional and defined as follows:

at Alice-iPhone:
intensional Roc14members@Alice-iPhone(string, relation, peer)
Roc14members@Alice-iPhone($name, $relation, $peer) :-
contact@Alice-iPhone($name, $relation, $peer),
group@Alice-iPhone($name, Roc14)

Observe that it is defined using extensional relations.

As usual, intensional knowledge is defined by rules such as the previous one, that we call *deductive* rules. Other rules such as the [*Send-Invitation*] rule, that we call *active*, produce extensional facts. Such an extensional fact *m@p* is received by the peer *p* (e.g., Bob-laptop and Alice's iPhone). During its next phase of local processing, this peer will consume these facts and produce new ones. By default, any processed fact disappears. Facts can be made persistent using persistence rules, illustrated next on the relation *calendar@Alice-iPhone*:

at Alice-iPhone:
calendar@Alice-iPhone($name, $date, $place, $peer) :-
calendar@Alice-iPhone($name, $date, $place, $peer),
¬ del.calendar@Alice-iPhone($name, $date, $place, $peer)

The rules state that a fact persists unless there is explicitly a deletion message (e.g., *del.calendar*).

Delegation by example

In the model, the semantics of the global system is defined based on local semantics and the exchange of messages and rules. Intuitively, a given peer chooses how to move to another state based on its local state (a set of

personal facts and messages received from other peers) and its program. A move consists in (1) consuming the local facts, (2) deriving new local facts, which define the next state, (3) deriving nonlocal facts, i.e., messages sent to other peers, and (4) modifying their programs via "delegations".

The derivation of local facts and messages sent to other peers are both standard and were illustrated in the previous example. The notion of delegation is novel and is illustrated next. Consider the following rule, installed at peer *Bob-laptop*:

at Bob-laptop:
confirm@$peer(rockclimbing, $date, $place,Bob) :-
calendar@Bob-laptop(rockclimbing, $date, $place, $peer),
checkAvailability@Bob-iPhone($date);

where *calendar@Bob-laptop*, *checkAvailability@Bob-iPhone* and *confirm@Alice-iPhone* are all extensional. Its semantics is as follows. Suppose that *agenda@Bob-laptop(rockclimbing, 06/12/2011, Fontainebleau, Alice-iPhone)* holds, then the effect of this rule is to install at *Bob-iPhone* the following rule:

at Bob-iPhone:
confirm@Alice-iPhone(rockclimbing, 06/12/2011,Fontainebleau,Bob) :-
checkAvailability@Bob-iPhone(06/12/2011);

The action of installing a rule at some other peer is called *delegation*. When *Bob-iPhone* runs, if *checkAvailability@Bob-iPhone(06/12/2011)* holds, it will send the message *confirm@Alice-iPhone(rockclimbing, 06/12/2011, Fontainebleau, Bob)* to *Alice-iPhone*.

Now suppose instead that *confirm@Alice-iPhone* is intensional. When *Bob-iPhone* runs, if *checkAvailability@Bob-iPhone(06/12/2011)* holds, the effect of this rule is to install at *Alice-iPhone* the following rule:

at Alice-iPhone:
confirm@Alice-iPhone(rockclimbing, 06/12/2011,Fontainebleau,Bob) :-

The intuition for the delegation from *Bob-laptop* to *Bob-iPhone* is that there is some knowledge from *Bob-iPhone* that is needed in order to realize the task specified by this particular rule. So, to perform that task, *Bob-laptop* delegates the remainder of the rule to *Bob-iPhone*. The delegation from *Bob-iPhone* to *Alice-iPhone* is somewhat different. Peer *Bob-iPhone* knows that *confirm@Alice-iPhone* (an intensional fact) holds until some change occurs. As *Alice-iPhone* may need this fact for his own computation, *Bob-iPhone* will pass this information to *Alice-iPhone* in the form of a rule (since as a fact, it would be consumed).

We next formalize the model illustrated by the previous example.

4.1.2 Formal model

Alphabets

We assume the existence of two infinite disjoint alphabets of sorted *constants*: *peer* and *relation*. We also consider the alphabet of *data* that includes in addition to *peer* and *relation*, infinitely many other constants of different sorts (notably, *integer*, *string*, *bitstream*, etc.). It is because *data* includes *peer* and *relation* that we may write facts such as those in the *birthday* relation. Similarly we have corresponding alphabets of sorted *variables*. An identifier starting by the symbol $ implicitly denotes a variable. A *term* is a variable or a constant.

A *schema* is an expression $(\Pi, \mathcal{E}, \mathcal{I}, \sigma)$ where Π is a (possibly infinite) set of peer IDs; $\mathcal{E}$ and $\mathcal{I}$ are disjoint sets, respectively, of *extensional* and *intensional* names of the form $m@p$ for some relation name m and some peer p; and the typing function σ defines for each $m@p$ in $\mathcal{E} \cup \mathcal{I}$ the arity and sorts of its components. Note that because $\mathcal{I} \cap \mathcal{E} = \varnothing$, no m is both intensional and extensional in the same p. Considering Π to be infinite reflects the assumption that the set of peers is dynamic and of unbounded size just like it is the case on the Web.

Facts and rules

Given a relation $m@p$, a (ground) (p-)*fact* is an expression $m@p(\overline{u})$ where $\overline{u}$ is a vector of data elements of the proper type, i.e., correct arity and correct sort for each component. For a set K of facts and a peer p, $K[p]$ is the set of p-facts in K. The notion of fact is central to the model. It will be the basis for both stored knowledge and communication. For instance, in the peer p, if we derive the extensional fact $r@p(1,2)$, this is a fact p knows. On the other hand, if we derive the extensional fact $s@q(2,3)$, this is a message that p sends to q.

A *(WebdamLog) rule* is an expression of the form

$$M_{n+1}@Q_{n+1}(\overline{U}_{n+1}) \text{ :- } (\neg)M_1@Q_1(\overline{U}_1)...(\neg)M_n@Q_n(\overline{U}_n)$$

where each M_i is a relation term, each Q_i is a peer term and each $\overline{U}_i$ is a vector of data terms. We also allow in the body of the rules, atoms of the form $X = Y$ or $X \neq Y$ where X, Y are terms.

We require a rule to be *safe*, i.e.,

1. For each i, if Q_i is a peer variable, it must be previously bound, i.e., it must appear in $\overline{U}_j$ for some positive literal $M_j@Q_j(\overline{U}_j)$, $j < i$.

2. Each variable occuring in a literal $\neg M_i@Q_i(\overline{U}_i)$ must be previously bound to a positive literal.

3. Each variable in the head must be positively bound in the body.

Remark 4.1 (Unguarded peer)**.** Observe that we treat differently peer and relation names. By (1), a peer variable has to be previously positively bound. We insist on (1) so that we control explicitly to whom a peer sends a message or delegates a rule.

Note also that because of (1), the ordering of literals is relevant. One could define a variation of the language, namely *peer-unguarded WebdamLog* by not imposing Constraint (1) and considering all orderings of the body literals (with the negative ones seen implicitly after all the others). When deriving new facts, we simply consider first the positive literals and never consider a literal if its peer is not instantiated.

We say that a rule is *deductive* if the head relation is intensional. Otherwise, it is *active*. Rules live in peers. We say that a rule in a peer p is *local* if all Q_i in all body relations are from p. It is *fully local* if the head relation is also from p. We will see that the following four classes of rules play different roles:

Local deduction Fully local deductive rules are used to derive intensional facts *locally*.

Update Local active rules are used for sending messages, i.e., facts, that modify the databases of the peers that receive them.

View delegation The local but not fully local deductive rules provide some form of view materialization. For instance, this rule results in providing at q a view of some data from p:

$$at\ p:\ r@q(\overline{U})\ \text{:-}\ (\neg)r_1@p(\overline{U_1}),...(\neg)r_n@p(\overline{U_n})$$

General delegation The remaining rules allow a peer to install arbitrary rules at other peers.

Peer and relation variables provide considerable flexibility for designing applications. However, observe that because of them, it may be unclear whether a rule is (fully) local or not, deductive or active. Using atoms such as $Q = p$, $Q \neq p$ for some constant peers and similarly for relations, one could remove the ambiguity and distinguish the nature of the rule. We omit the formal details. Note that in a real system, one can wait until a rule

is (partially) instantiated at runtime to find what its nature is, and decide what should be done with it.

The semantics of *WebdamLog* is based on autonomous local computations of the peers. We consider this first, then look at the global semantics of *WebdamLog* systems.

Local computation

A local computation happens at a particular peer. Based on its set of facts and set of rules, the peer does the following: (1) some local deduction of intensional facts, (2) the derivation of extensional facts that either define its next state or are sent as messages, and (3) the delegation of rules to other peers.

(Local deduction) For local deduction, we want to rely on the semantics of standard Datalog languages. However, because of possible relation variables, *WebdamLog* rules are not strictly speaking proper Datalog$^\neg$ rules, since the relation names of atoms may include variables. So, to specify local deduction, we proceed as follows. We start by *grounding* the peer and relation variables appearing in the rules. More precisely, for each rule

$$M_{n+1}@Q_{n+1}(\overline{U}_{n+1}) \text{ :- } (\neg)M_1@Q_1(\overline{U}_1)...(\neg)M_n@Q_1(\overline{U}_n)$$

of peer p, we consider the set of rules obtained by instantiating relation variables M_i with relation constants and peer variables Q_i with peer constants. To ensure finiteness, we only use constants from the active domain of the peer, that is, that appear in some fact or rule in the peer state. We can now deal with pairs $m@p$ of relation and peer constants as normal relation symbols in Datalog. Since for local deduction, we are only interested in fully local deductive rules, we will remove rules with a relation $m@q$ for $q \neq p$ or an extensional relation in the head. We must also remove rules that violate the arity or sort constraints of σ. The remaining rules are all fully local deductive rules which belong to standard Datalog.

Now, given a set I of facts and a set P_d of fully local deductive rules (defined as in the previous paragraph), we denote by $P_d^*(I)$ the set of facts inferred from I using P_d with a standard Datalog semantics. For instance, in absence of negation, the semantics is, as in classical Datalog, the least model containing I and satisfying P_d. When considering negation, one can use any standard semantics of Datalog with negation, say well-founded [Prz90] or stable [GL88]. For results in Section 4.4.2, we will use a variant of stratified negation semantics [CH85].

(Updates) Given a set K of facts and a set P_a of local active rules, the set $P_a(K)$ of *active consequences* is the set of *extensional* facts $v(A)$ such that for some rule A :- Θ of P_a and some valuation v, $v(\Theta)$ holds in K, and $v(A), v(\Theta)$ obey the typing and sort constraints of σ. This is the set of *immediate consequences*. Note that it does not necessarily contain all facts in K.

Observe that for deductive rules, we typically use a fixpoint (based on the particular semantics that is used), whereas for active rules, we use the immediate consequence operator that is explicitly procedural.

(Delegation) Given a set K of facts and a set P of (active and deductive) rules in some peer p, the *delegation* $\gamma_{pq}(P, K)$ of peer p to $q \neq p$ is defined as follows.

If for some deductive rule $M@Q(\overline{U})$:- Θ in P, there exists a valuation v such that $v\Theta$ holds in K, $v(Q) = q$, and the typing constraints in σ are respected, then

$$vM@vQ(v\overline{U}) \text{ :-}$$

is in $\gamma_{pq}(P, K)$.

If for some active or deductive rule

$$A \text{ :- } \Theta_0, (\neg)M@Q(\overline{U}), \Theta_1$$

in P (where Θ_0, Θ_1 are sequences of possibly negated atoms), there exists a valuation v satisfying σ such that $v\Theta_0$ contains only p-facts, $v\Theta_0$ holds in K, and $vQ = q(\neq p)$, then

$$vA \text{ :- } (\neg)M@vQ(v\overline{U}), v\Theta_1$$

is in $\gamma_{pq}(P, K)$.

Nothing else appears in $\gamma_{pq}(P, K)$.

Observe that we do not produce facts that are improperly typed. In practice, a peer p may not have complete knowledge of the types of some peer q's relations. Then p may "derive" an improperly typed fact. This fact will be sent and rejected by q. From a formal viewpoint, it is simply assumed that the fact has not even been produced. Similarly, a peer may delegate an improperly typed rule, but that rule will never produce any facts, and so can safely be ignored.

We are now ready to specify the semantics of *WebdamLog* systems.

States and runs

A *(WebdamLog) state* of the schema $(\Pi, \mathcal{E}, \mathcal{I}, \sigma)$ is a triple $(I, \Gamma, \tilde{\Gamma})$ where for each $p \in \Pi$, $I(p)$ is a finite set of extensional p-facts at p, $\Gamma(p)$ is the finite set of rules at p, and $\tilde{\Gamma}(p, q)$ $(p \neq q)$ is the set of rules that p delegated to q. For $p \in \Pi$, the (p-)*move* from $(I, \Gamma, \tilde{\Gamma})$ to $(I', \Gamma', \tilde{\Gamma}')$ (corresponding to the *firing* of peer p) is defined as follows. Let P_p be $\Gamma(p) \cup (\cup_q \tilde{\Gamma}(q, p))$, P_{pd} be the set of fully local deductive rules in P_p and P_{pa} the set of local active rules in it. Then the next state is defined as follows:

- (Local deduction) Let $K = P_{pd}^*(I(p))$.
- (Updates) $I'(p) = P_{pa}(K)[p]$; and
  (external activation) $I'(q) = I(q) \cup P_{pa}(K)[q]$ for each $q \neq p$.
- (Delegations) $\tilde{\Gamma}'(p, q) = \gamma_{pq}(P_p, K)$ for each q $\neq$ p; and
  $\tilde{\Gamma}'(p', q') = \tilde{\Gamma}(p', q')$ otherwise.

A (*WebdamLog*) *system* is a state $(I, \Gamma, \tilde{\Gamma})$ where $\tilde{\Gamma}(p, q) = \varnothing$. We will speak of the system (I, Γ) (since $\tilde{\Gamma}$ is empty). A sequence of moves is *fair* if each peer p is invoked infinitely many times. A *run* of a system (I, Γ) is a fair sequence of moves starting from (I, Γ).

Observe that $I(p)$ is finite for each peer and that it remains so during a run, even if the number of peers is infinite. Note also that deletions are implicit: a fact is deleted if it is not derived for the next state. We recall that facts can be made persistent using persistence rules of the form

$$r@p(\overline{U}) \text{ :- } r@p(\overline{U}), \neg del.r@p(\overline{U})$$

In the following, such a rule for relation $r@p$ will be denoted *persistent* $r@p$.

Remark 4.2 (Fact and rules)**.** It is important to observe a difference between the semantics of facts and rules. Observe that, if we visit twice peer p in a row, the fact-messages that p sends to q accumulate at q. On the other hand, the new set of delegations replaces the previous such set. Moreover, when we visit q, the messages of q are consumed whereas the delegations stay until they are replaced. These subtle differences are important to capture different facets of distributed computing, e.g., for capturing materialized views or for providing the expected semantics to extensional / intensional data.

4.2 Discussion

In this section, we present examples that illustrate the interest of our model for distributed data management, and make key observations about different aspects of the model.

We first consider two serious criticisms that could be adressed to the model, namely too much synchronization and too little local control. We show how both issues can be resolved.

4.2.1 Too much synchronization

Observe that moves capture some form of asynchronicity and parallelism. The peer that fires is randomly chosen and does (atomically) some processing. However, there is still some form of synchronization, that may be undesired. When we process peer p, messages from p to some peer q are instantaneously available in q. This is impossible to guarantee in practice. In a standard manner, when a more precise modeling is desired, one can introduce a peer acting as the network between p and q. Instead of going instantaneously from p to q, the message goes instantaneously from p to $network_{pq}$, waits there until $network_{pq}$ is fired, then goes instantaneously to q, and similarly for delegations. This captures more realistically what happens in practice, and does not require changing the model.

4.2.2 Too little local control

In the model we have defined, nothing prevents a peer p from modifying another peer q's relations or accessing q's data using delegation. In realistic settings, one would want a peer to be able to hold private information, which cannot be modified or accessed by another peer without its permission. This can be easily accomplished by extending the model with *local relations*. These relations can only appear in p's own facts and rules (i.e., $I(p)$ and $\Gamma(p)$), but not in any rules delegated to p (in practice, this means p would simply ignore any delegations using one of its private relations).

To illustrate, suppose that we want to control the access to a relation $r@p$ of peer p. We create for this purpose two local relations $read@p(\$r, \$q)$ and $write@p(\$r, \$q)$ that store who can read/write in p's relations. Note that the *read* and *write* relations are local, i.e., only p can specify the access rights in p. Relations $r@p$ and $del.r@p$ must also be local so that p control access to them. To obtain relation $r@p$, a peer q sends a message $get@p(r, q)$. The following rule controls whether q will receive the data it requested:

```
at p: send@$q($r,$x) :- get@p($r,$q), read@p($r,$q),
                        $r@p($x)
```

Insertions in $r@p$ (or deletions using $del.r@p$) are treated similarly. Access control is at the center of the work around *WebdamExchange*, described in Chapter 5.

We next consider two subtleties of delegation.

4.2.3 Delegation and complexity

Consider the rule:

at p: m@q() :- m_1@p($q,$x), m_2@$q($x)

If there are 1000 distinct tuples $(p_i, 0)$ such that $m_1@p(p_i, 0)$ holds, then we have to install rules in 1000 distinct peers. So delegation is inherently transforming data complexity into program complexity.

4.2.4 Peer life and delegation

It is very simple in the model to consider that peers are born, die or hibernate. We simply have to insist that p can be fired (p-move) only if p is alive and not hibernating. We can assume that messages and delegations to dead peers are simply lost and that for hibernating ones, they are buffered somewhere in the network. A subtlety is that (with this variant of the model), if a peer dies without cleanly terminating, delegations from this peer are still valid. In practice, the system may realize that a particular peer is no longer present and terminate its delegations.

We conclude this section with three examples that illustrate different aspects of the language, communications, persistence services, and rule updates.

4.2.5 Multicasting

We can simulate channels, i.e., m-n communications with the following rules:

```
at q: persistent channelsubscribe@q
      channel@$p($m,q,$s) :- channelsubscribe@q($p,$m),
            $m@q($s)
```

The rules at peer q allows him to support channels. A peer p can subscribe to receiving all the messages from the channel m hosted by q by sending $channelsubscribe@q(p, m)$ to q. Then, whenever someone sends a message $m@q(s)$, p will receive $channel@p(m, q, s)$.

4.2.6 Database server replication

The following rule allows a database server to replicate relations from many peers:

```
intensional export@db(relation,peer)
at db: persistent tobeexported@db
       export@db($r,$p,$x) :- tobeexported@db($r, $p),
            $r@$p($x)
```

If a peer p wants his relation $r@p$ to be stored at db, then p simply needs to send db the message $tobeexported@db(r, p)$. Now, $export@db(r, p, \$x)$ is a copy of $r@p(\$x)$.

4.2.7 Rule updates and rule deployment

Observe that (to simplify) we assumed that the set of rules in a run is fixed, i.e., $\Gamma(p)$ is fixed for each p. It is straightforward to extend the model to support addition or deletion of rules. Furthermore, one might want to be able to control whether a particular rule is deployed on a particular peer. To illustrate this point, consider the two rules:

```
at p: persistent server@p
      f@$p($u) :- server@p($p), f1@$p($u1),...,fn@$p($un)
```

Sending the message $server@p(q)$ results in installing

```
at q: f@q($u) :- f1@q($u1),...,fn@q($un)
```

Note that if we send the message $del.server@p(q)$, the rule is removed.

4.3 Expressivity

In this section, we study the expressive power of *WebdamLog* and of different languages that are obtained by allowing or restricting delegations. We also consider the expressive power of timestamps. More precisely, we consider the following languages for rules:

- WL (*WebdamLog*): the general language.
- VWL (views WL): the language obtained by restricting delegations to only view delegations.
- SWL (simple WL): the language obtained by disallowing all kinds of delegations.

At the core of view delegation, we find the maintenance of materialized views. To maintain views, we will see that timestamps turn out to be useful. More precisely, for time, we assume that each peer has a local predicate called *time* (with $time(t)$ specifying that the current move started at local time t.) The predicate $<$ is used to compare timestamps. Note that each peer has its separate clock, so the comparison of timestamps of distinct peers is meaningless. To prevent time from being a source of nondeterminism, for t_1, t_2 two times at different peers, we have: $t_1 \not< t_2$ and $t_2 \not< t_1$. The languages obtained by extending the previous languages with timestamps are denoted as follows: WL^t, VWL^t, SWL^t.

4.3.1 Traces and simulations

To formally compare the expressivity of the above languages, we need to introduce the auxiliary notions of trace and simulation.

Let $r = (I_1, \Gamma_1, \widetilde{\Gamma}_1), ...(I_n, \Gamma_n, \tilde{\Gamma}_n), ...$ be a run. Let M be a set of predicates and I a set of facts. Then $\Pi_M(I)$ is the set of facts in I with predicates in M. The *M-trace* of the run r for a set M of predicates is the subsequence of $\pi_M(I_{i_1}), ..., \pi_M(I_{i_n})...$ obtained by starting from $\pi_M(I_1), ..., \pi_M(I_n)...$ and removing all repetitions, i.e., deleting the $(k+1)$th element of the sequence if it is identical to the kth, until the sequence does not contain two identical consecutive elements. Given an initial state S and a set of predicates M, we denote by $M\text{-}trace(S)$ the set of M-traces of runs from S. In some sense, it is what can be observed from S when only facts over M are visible.

Let α be a set of peers. An initial state $S = (I, \Gamma)$ can be *α-simulated* by an initial state $S' = (I, \Gamma')$ if $\Gamma(p) = \Gamma'(p)$ for all $p \in \alpha$ and S and S' have the same M-traces, where M is the set of relations of S. In other words, from the point of view of what is visible from S, S' behaves exactly like S. The set of peers α is meant to capture the part of the system (one or more peers) that we want to keep strictly identical.

Now, we say that a language L can be *simulated* by a language L', denoted $L \prec L'$, if there exists a translation τ from programs in L to programs in L' such that for each initial state (I, Γ) (with programs in L) and for each α, $(I, \overline{\tau}(\Gamma))$ α-simulates (I, Γ) where $\overline{\tau}$ is defined by: for each peer p,

- if $p \in \alpha$, $\overline{\tau}(\Gamma(p)) = \Gamma(p)$.
- otherwise, $\overline{\tau}(\Gamma(p)) = \tau(\Gamma(p))$.

Clearly, in the previous definition, the peers in α are not part of the simulation, they behave exactly as originally. In some sense, they should not even be aware that something has changed.

4.3.2 Expressivity results

The expressive power of the different languages are compared in Figure 4.1. The containments are strict except for that of VWL^t inside WL^t where the issue remains open.

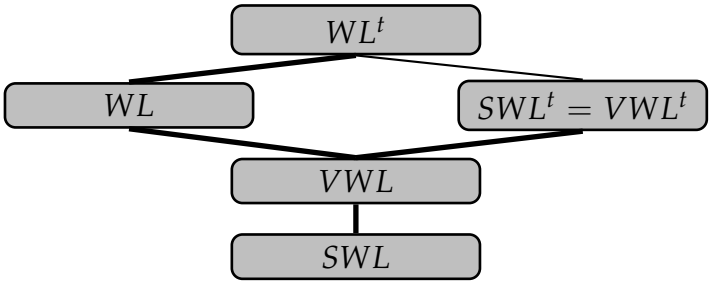

Figure 4.1: Expressive power of the rule languages (the inclusion is strict when the arc is in bold)

Our first result states that view delegation cannot be simulated by simple rules.

Theorem 4.3 (No views in SWL)**.** *$VWL \not\leq SWL$.*

Proof. Intuitively, the difficulty is that the system may visit an arbitrary number of times the same peer p before visiting another peer q. Then q sees all the messages from p at the same time and ignores in which order they were received.

Formally, consider a VWL system (I, Γ) consisting of three peers p_α, p, q. There are two facts that hold in the initial state: $true@p_\alpha()$, $true@p()$.

The set of active rules $\Gamma(p_\alpha)$ maintain the peer p_α in a permanent flip-flop between two modes:

$$
\begin{aligned}
at\ p_\alpha : r@p() &\text{ :- } true@p_\alpha() \\
false@p_\alpha() &\text{ :- } true@p_\alpha() \\
del.r@p() &\text{ :- } false@p_\alpha() \\
true@p_\alpha() &\text{ :- } false@p_\alpha()
\end{aligned}
$$

Note that p_α keeps inserting then deleting the same proposition in p, namely $r@p()$. Peer p uses the following four rules:

$at\ p : r@p() \text{ :- } r@p(), \neg del.r@p()$
$\quad true@p() \text{ :- } false@p()$
$\quad false@p() \text{ :- } true@p()$
$\quad s@q() \text{ :- } r@p()$

The first active rule maintains relation $r@p$. The next two active rules maintain p in a flip-flop between two modes. The last rule is a view delegation rule. It is because of this latter rule that the system is in VWL but not in SWL.

Finally peer q has one active rule:

$at\ q : true@q() \text{ :- } s@q()$

Suppose for a contradiction that there is a p_α-simulation of this system in SWL, via some program translation function τ. As the set of peers is finite (namely 3), the initial state $(I, \tau(\Gamma))$ is finite. Thus, it includes a finite set of relation names and constants. This means that there is a finite number of distinct messages that can be sent during a run of this system. Now let r_1 be any run of $(I, \tau(\Gamma))$ such that the initial segment of activated peers is as follows: p_α, then p, then p_α, then p, etc., n times (for n to be fixed later in the proof), and then q. Let $I, I_1, I_2, ..., I_{2n-1}, I_{2n}, I'$ be the trace of r_1. Because of the two flip-flops, the trace has this size and it is clear from it which peer has been activated at each step.

Consider a second run r_2 which is defined like r_1 except that this time we visit p_α and p, $n+1$ times, then q. Let $I, I_2, I_3..., I_{2n-1}, I_{2n}, I_{2n+1}, I_{2n+2}, I''$ be the trace of r_2.

Observe that while p and p_α are being activated, q is simply accumulating messages. Recall that the set of messages that q may accumulate is finite. Thus we can choose n large enough so that $I_{2n+2}(q) = I_{2n}(q)$. Suppose that $I'(q)$ contains $true@q$. Then because the set of messages at q is the same in the second run, $I''(q)$ also contains $true@q$, a contradiction because the last iteration in p_α, p must have removed $r@p$. A similar contradiction occurs if $true@q$ is not produced. Thus such a simulation does not exist. □

Next we separate VWL and WL.

Theorem 4.4 (No general delegations in VWL)**.**
WL $\not\prec$ VWL.

Proof. (sketch) Intuitively, peer q will use a general delegation to ask peer p to do something that is beyond the capability of the rules in p. This is

not trivial because p may perform very complex operations with arbitrarily many complex rules. However, it turns out that there is a limit to what p can do. To prove it, we use the fact that with formulas using a bounded number k of variables, one cannot check whether a graph has a clique of size $k+1$ (when an ordering of the nodes is not available).

Formally, consider a WL system (I, Γ) that consists of three peers p_α, p, q. Intuitively, peer p_α sends a sequence of updates to a graph that is originally empty and is stored at p. To do that, p_α has a persistent relation that stores a sequence of updates. More precisely, p_α has a set of tuples of the form: $upd@p_\alpha(i, o, a, b)$ where i in [0,m] for some m and there is a single tuple for each i, o in { ins, del }, and a, b are data elements in a very large fixed set Σ (the identifiers of the graph g.) Peer p_α also has a persistent relation *next* containing the tuples: $[0,1], ...[m-1, m]$. Finally, p_α has the fact $now@p_\alpha(0)$ in its initial state. The program of p_α consists of the following active rules:

$$\begin{aligned} at\ p_\alpha : g@p(\$x, \$y) \text{ :- } & now@p_\alpha(\$i), upd@p_\alpha(\$i, ins, \$x, \$y) \\ del.g@p(\$x, \$y) \text{ :- } & now@p_\alpha(\$i), upd@p_\alpha(\$i, del, \$x, \$y) \\ now@p_\alpha(\$j) \text{ :- } & now@p_\alpha(\$i), next(\$i, \$j) \end{aligned}$$

Now p has the following active rule for maintaining the graph g :

$$at\ p : g@p(\$x, \$y) \text{ :- } g@p(\$x, \$y), \neg del.g@p(\$x, \$y)$$

Finally, peer q has a rule delegation to p:

$$at\ q : clique@q() \text{ :- } \wedge_{1 \leqslant i,j \leqslant n} g@p(\$x_i, \$x_j), \$x_i \neq \$x_j$$

which essentially requests p to send a message if there exists an n-clique in $g@p$. Peer q also has a flip-flop rule:

$$\begin{aligned} at\ q : true@q() \text{ :- } & false@q() \\ false@q() \text{ :- } & true@q() \end{aligned}$$

Originally $true@q()$ holds.

Suppose for a contradiction that there is a p_α-simulation of this system in *VWL*. Consider the run of (I, Γ) beginning with a very long sequence $q(p_\alpha)^* p(p_\alpha)^* ... p$ where each time p is called, the graph oscillates between "there is a clique" and "there isn't". Note that the first time q is called, it installs the delegation.

Let k be the number of variables and constants that appear in a rule in $\tau(\Gamma(p))$. As the rules in p have less than k symbols, they can only evaluate formulas in FO^k. Choose $n > k$, so that formulas in FO^k cannot check for the presence of an n-clique in a graph. Choose also the set of mode identifiers Σ large enough. (Recall that the translation for the rules of p is independent from the program of q and p_α.) So, it is not possible for p to

evaluate whether there is a clique. So q has to be called before each *clique* message to check the existence of a clique. Note that it is possible to do so: p pretends it has not been called and waits until q is called; then q sends a secret message to p to tell p whether there is a clique.

This is "almost" a simulation except that q has a bounded memory that depends essentially on Σ. Now consider a very long sequence of the WL system that never calls q. If the sequence is long enough, its simulation in VWL will visit twice the same state. Then by pumping, one can construct an infinite run of the VWL simulating system such that the flip-flop of q is never activated. This corresponds to a simulation of an unfair run of the WL system, a contradiction. Thus there can be no VWL simulation of the above WL system. □

We now consider timestamps. The next result compares the expressive power of WL and WL^t.

Theorem 4.5 (Timestamps)**.** *For a finite number of peers,*

1. *WL is in* PSPACE*;*
2. *SWL^t over a single peer can simulate any arbitrary Turing machine;*
3. *Thus, SWL$^t \not\leq$ WL and (a fortiori) WL$^t \not\leq$ WL.*

Proof. (sketch) For (1.), consider a fixed schema over a finite number of peers. Let (I, Γ) be an initial instance of size $n = |I| + |\Gamma|$. Let $(I_i, \Gamma, \widetilde{\Gamma}_i)$ be an instance that is reached during the computation. Because the schema is fixed, the number of facts that can be derived is bounded by a polynomial in n, and each fact is also of bounded size. So, $|I_i|$ can be bounded by a polynomial in n. Similarly, the size of $\widetilde{\Gamma}_i$ can be bounded by a polynomial in n, since a rule that is delegated is essentially determined by an instantiation of an original rule and a position in it. Thus we can represent $(I_i, \Gamma, \widetilde{\Gamma}_i)$ in polynomial space in n. Hence, WL is in PSPACE.

Now consider (2.). Let M be a Turing Machine. We can assume without loss of generality that it is deterministic and that it has a tape that is infinite only in one direction. The SWL^t system that simulates it is as follows. Its initial instance encodes the initial state of M. More precisely, it has a relation *input*, with initial value

$$\{\ input(0,1,a_1),\ input(1,2,a_2),\ \ldots\ input(n-1,n,a_n)\ \}$$

where $a_1a_2...a_n$ is the input of M. It also has a relation *tape* that is originally empty.

First, the SWL^t system copies the input on its tape using the timestamps $t_0, t_1, t_2...$ to identify tape cells. More precisely, it constructs,

$$\{tape(t_0,t_1,a_1,s_0),tape(t_1,t_2,a_2,\perp),...,tape(t_{n-1},t_n,a_n,\perp)\}$$

where s_0 is the start state of M. Using rules from SWL^t, it is straightforward to simulate moves of M. The only subtlety is that at each step of the iteration, the tape is augmented so that there is no risk of reaching its limit. The fact that the cells are denoted with timestamps guarantees that no two cells will have the same ID.

Now, given the encoding of a word w, one can simulate the computation of TM on w. Thus (2), so (3). □

Note that the converse of (1) holds: any PSPACE query over an ordered database can be computed in SWL (hence WL) with a single peer. This can be shown by proving how to simulate in SWL with a single peer, the language Datalog$^{\neg\neg}$ that can express all PSPACE queries on ordered databases [AV91].

Next we see how to use timestamps to simulate view maintenance.

Theorem 4.6 (Views with timestamps)**.** *VWL$^t \approx$ SWL^t.*

Proof. (sketch) We illustrate with an example the simulation of view delegation by a program with timestamps.

Consider a VWL system with an extensional relation $s@q$ and the deductive rule at p: $r@p(\overline{U})$:- $s@q(\overline{U})$ that specifies that $r@p$ is a view of $s@q$. The simulation of the view delegation in SWL^t is as follows.

at q : *persistent past@q*
$aux@p(\overline{U},\$t)$:- $s@q(\overline{U})$, $time@q(\$t)$
$past@q(\$t)$:- $time@q(\$t)$
$obsolete@p(\$t)$:- $past@q(\$t)$

at p : *intensional r@p*
persistent aux@p, obsolete@p
$r@p(\overline{U})$:- $aux@p(\overline{U},\$t)$, $\neg\, obsolete@p(\$t)$

Then the value of $r@p$ is that of $s@q$ when q was last visited, i.e., $r@p$ is a copy of $s@q$ at the last visit of q.

The above simulation is straightforwardly generalized to arbitary VWL systems, from which we obtain the desired VWL$^t \approx$ SWL^t. □

It is still open whether WL$^t \not\prec$ VWL^t.

4.4 Convergence of *WebdamLog*

Systems that converge to a unique state independently of the order of computation, i.e., some form of Church-Rosser property, are of particular interest. In this section, we consider two kinds of such systems: the positive and the strongly-stratified *WebdamLog* systems. Indeed, we show that such systems continue to converge even in presence of insertions of facts or rules. Finally, we show that for these two classes of systems, the distributed semantics can be seen as mimicking the centralized semantics.

4.4.1 Positive *WebdamLog*

Clearly, negation may explain why a system does not converge. However, the following example shows that even in absence of negation, convergence is not guaranteed because the order of arrival of messages matters:

Example 4.7. Consider the rules:

```
at q:  extensional r1@q, r2@q, r@q
       persistent r@q
       r@q() :- r1@q(), r2@q()
at q1: r1@q() :-
at q2: r2@q() :-
```

If we process the peers according to the order $q1, q, q2, q, q1, \ldots$, then $r@q$ is never derived. If we consider instead the order $q1, q2, q, q1, q2, q, \ldots$, then $r@q$ is derived and remains forever. The absence of convergence here is in fact a desired feature of the model: the extensional relations model events, so their arrival times matter.

On the other hand, note that, as we will see, if in the example $r1@q$ and $r2@q$ were intensional, the system would converge.

We now introduce the restricted systems we study in this section. A *WebdamLog* state or system is *positive* if the following holds:

1. Each of its rules is positive (no negation); and
2. Each extensional relation $m@p$ is made persistent with a rule of the form $m@p(\overline{U})$:- $m@p(\overline{U})$.

We will see that because of these restrictions, the states in runs of positive systems are monotonically increasing. For positive systems with a finite number of peers, there are only finitely many possible states, so monotonicity implies that runs converge after a finite number of steps. We

will also show convergence for positive systems with infinitely many peers, except that in this case, we may converge only in the limit. This motivates the following somewhat complex definition of convergence.

A run $S_0, S_1, S_2, \ldots$ *converges* to a possibly infinite state $S^* = (I^*, \Gamma^*, \widetilde{\Gamma}^*)$ if for each finite $S' \subseteq S^*$, there exists $k_{S'}$ such that for all $k > k_{S'}$, $S' \subseteq S_k$ and if for each finite $S' \not\subseteq S^*$, there is $k_{S'}$ such as for all $k > k_{S'}$, $S' \not\subseteq S_k$. We say a system S *converges* if all its runs converge to the same state.

The following theorem states the convergence of (possibly infinite) positive systems.

Theorem 4.8 (Convergence)**.** *All positive WebdamLog systems converge.*

Lemma 4.9. *Suppose $I_1(p^*) \subseteq I_2(p^*)$, $\Gamma_1(p^*) = \Gamma_2(p^*)$ and $\widetilde{\Gamma}_1(q, p^*) \subseteq \widetilde{\Gamma}_2(q, p^*) \forall q \neq p^*$. Let $P_{a,i}$ (resp. $P_{d,i}$) be the set of local active (resp. fully local deductive) rules in $\Gamma_i(p^*) \cup \cup_{q \neq p^*} \widetilde{\Gamma}_i(q, p^*)$. Then if there is no negation in the rules, we have $P_{a,1}(K_1) \subseteq P_{a,2}(K_2)$ and*

$$\gamma_1(p^*, q)(P_{a,1}, K_1) \subseteq \gamma_2(p^*, q)(P_{a,2}, K_2) \forall q \neq p^*$$

where $K_i = P^*_{d,i}(I_i(p^*)))$.

Proof. (of Lemma 4.9) Since $\Gamma_1(p^*) = \Gamma_2(p^*)$ and $\widetilde{\Gamma}_1(q, p^*) \subseteq \widetilde{\Gamma}_2(q, p^*)$ for all $q \neq p^*$, it follows that $P_{a,1} \subseteq P_{a,2}$ and $P_{d,1} \subseteq P_{d,2}$. Together with $I_1(p^*) \subseteq I_2(p^*)$, and in absence of negation, we obtain $P_{a,1}(P^*_{d,1}(I_1(p^*))) \subseteq P_{a,2}(P^*_{d,2}(I_2(p^*)))$. Likewise, $\gamma_{p^*q}(P_{a,1}, P^*_{d,1}(I_1(p^*))) \subseteq \gamma_{p^*q}(P_{a,2}, P^*_{d,2}(I_2(p^*)))$. □

Proof. (of Theorem 4.8) In fact, we will prove that the result is true for a simple update I', Γ', since the result is then easy to generalize. Consider a positive *WebdamLog* system $(I_0, \Gamma_0, \widetilde{\Gamma}_0)$. Let $r = (I_0, \Gamma_0, \widetilde{\Gamma}_0)(I_1, \Gamma_1, \widetilde{\Gamma}_1)$ $(I_2, \Gamma_2, \widetilde{\Gamma}_2) \ldots$ be a run for this system. It follows from the definition of moves that $\Gamma_i = \Gamma_j$ for all $i, j \geqslant 0$ and that delegated rules are sub-rules of these sets so have no negation. So $(I_i, \Gamma_i, \widetilde{\Gamma}_i)$ is positive for every $i \geqslant 0$. We show by induction on i that $I_i(p) \subseteq I_{i+1}(p)$ and $\widetilde{\Gamma}_i(p, q) \subseteq \widetilde{\Gamma}_{i+1}(p, q)$ for all i and all peers p, q, i.e., the states in the run increase monotonically. Using this property, it is easy to show that r converges to the (possibly infinite) state $(I^*, \Gamma_0, \widetilde{\Gamma}^*)$ where $I^*(p) = \cup_i I_i(p)$ and $\widetilde{\Gamma}^*(p, q) = \cup_i \widetilde{\Gamma}_i(p, q)$. The base case ($i = 0$) for our induction is straightforward. If the first move is a p^*-move, then by the definition of move, we have $I_0(q) \subseteq I_1(q)$ for all $q \neq p^*$. For peer p^*, we use the fact that $I_0(p)$ contains only extensional p-facts and that $\Gamma_0(p)$ contains persistence rules for all extensional relations of p.

We thus obtain $I_0(p^*) \subseteq I_1(p^*)$. As for delegations, we have $\widetilde{\Gamma}_0(p,q) = \varnothing$ for all p,q (since $(I_0,\Gamma_0,\widetilde{\Gamma}_0)$ is initial), hence $\widetilde{\Gamma}_0(p,q) \subseteq \widetilde{\Gamma}_1(p,q)$ for all peers p,q. Suppose next that the claim holds for all $i < k$. Let p^* be the peer whose move takes $(I_k,\Gamma_k,\widetilde{\Gamma}_k)$ to $(I_{k+1},\Gamma_{k+1},\widetilde{\Gamma}_{k+1})$. Using the same argument as in the base case, we obtain $I_k(p) \subseteq I_{k+1}(p)$ for all peers p. According to the definition of moves, $\widetilde{\Gamma}_k(p,q) = \widetilde{\Gamma}_{k+1}(p,q)$ whenever $p \neq p^*$. Thus, the only interesting case is when $p = p^*$ and $\widetilde{\Gamma}_k(p^*,q) \neq \varnothing$. In this case, we must have visited peer p^* previously. Let j be such that the last p^*-move took $(I_j,\Gamma_j,\widetilde{\Gamma}_j)$ to $(I_{j+1},\Gamma_{j+1},\widetilde{\Gamma}_{j+1})$. Since our last visit to p^* was at timepoint j, $\widetilde{\Gamma}_{j+1}(p^*,q) = \widetilde{\Gamma}_k(p^*,q)$. By repeatedly applying the IH, we obtain $I_j(p) \subseteq I_k(p)$ and $\widetilde{\Gamma}_j(p,q) \subseteq \widetilde{\Gamma}_k(p,q)$ for all peers p,q. In particular, we have $I_j(p^*) \subseteq I_k(p^*)$, $\Gamma_j(p^*) = \Gamma_k(p^*)$, and $\widetilde{\Gamma}_j(p^*,q) \subseteq \widetilde{\Gamma}_k(p^*,q)$. Applying Lemma 4.9, we get $\widetilde{\Gamma}_{j+1}(p^*,q) \subseteq \widetilde{\Gamma}_{k+1}(p^*,q)$, which yields the desired $\widetilde{\Gamma}_k(p^*,q) \subseteq \widetilde{\Gamma}_{k+1}(p^*,q)$, and completes our proof of the monotonicity claim.

Now consider two runs $r_1 = (I_{0,1},\Gamma_{0,1},\widetilde{\Gamma}_{0,1})(I_{1,1},\Gamma_{1,1},\widetilde{\Gamma}_{1,1})(I_{2,1},\Gamma_{2,1},\widetilde{\Gamma}_{2,1})\ldots$ and $r_2 = (I_{0,2},\Gamma_{0,2},\widetilde{\Gamma}_{0,2})(I_{1,2},\Gamma_{1,2},\widetilde{\Gamma}_{1,2})\ (I_{2,2},\Gamma_{2,2},\widetilde{\Gamma}_{2,2})\ldots$ for the system which converge respectively to $(I_1^*,\Gamma_1^*,\widetilde{\Gamma}_1^*)$ and $(I_2^*,\Gamma_2^*,\widetilde{\Gamma}_2^*)$. We will prove by induction on $i \geqslant 0$ that for every state $(I_{i,1},\Gamma_{i,1},\widetilde{\Gamma}_{i,1})$ of r_1, there is $j \geqslant 0$ such that $I_{i,1}(p) \subseteq I_{j,2}(p)$ and $\widetilde{\Gamma}_{i,1}(p,q) \subseteq \widetilde{\Gamma}_{j,2}(p,q)$ for all peers p,q. This, together with monotonicity property in the previous paragraph, yields the desired $(I_1^*,\Gamma_1^*,\widetilde{\Gamma}_1^*) = (I_2^*,\Gamma_2^*,\widetilde{\Gamma}_2^*)$. The base case ($i = 0$) is trivial since $(I_{0,1},\Gamma_{0,1},\widetilde{\Gamma}_{0,1}) = (I_{0,2},\Gamma_{0,2},\widetilde{\Gamma}_{0,2})$ (as they are both runs for the same system). For the induction step, suppose the claim holds for $i \leqslant k$, and consider $(I_{k+1,1},\Gamma_{k+1,1},\widetilde{\Gamma}_{k+1,1})$. Let p^* be the peer whose move takes $(I_{k,1},\Gamma_{k,1},\widetilde{\Gamma}_{k,1})$ to $(I_{k+1,1},\Gamma_{k+1,1},\widetilde{\Gamma}_{k+1,1})$. By the IH, we can find j such that $I_{k,1}(p) \subseteq I_{j,2}(p)$ and $\widetilde{\Gamma}_{k,1}(p,q) \subseteq \widetilde{\Gamma}_{j,2}(p,q)$ for all p,q. As r_2 is a fair run, we can find $l \geqslant j$ such as $(I_{l+1,2},\Gamma_{l+1,2})$ results from a p^*-move. Since states are monotonically increasing in r_2, $I_{k,1}(p) \subseteq I_{j,2}(p) \subseteq I_{l,2}(p)$ and $\widetilde{\Gamma}_{k,1}(p,q) \subseteq \widetilde{\Gamma}_{j,2}(p,q) \subseteq \widetilde{\Gamma}_{l,2}(p,q)$ for all p,q. Using Lemma 4.9, $I_{k+1,1}(p^*) \subseteq I_{l+1,2}(p)$ and $\widetilde{\Gamma}_{k+1,1}(p,q) \subseteq \widetilde{\Gamma}_{l+1,2}(p,q)$ for all peers p,q. □

The previous theorem is still true if one allows the peers to insert facts and rules. One can show that the system will reach a stable state that does not depend on the points of insertion.

Theorem 4.10 (Updates)**.** *Given two positive WebdamLog systems (I,Γ) and (I′,Γ′), for any run of the system (I,Γ), if for a given step, I′ is added to the current set of facts and Γ′ to the current set of rules, then the modified run converges to*

the convergence state of $(I \cup I', \Gamma \cup \Gamma')$.

Proof. Let $(I_{0,1}, \Gamma_{0,1}, \widetilde{\Gamma}_{0,1}), (I_{1,1}, \Gamma_{1,1}, \widetilde{\Gamma}_{1,1})...$ be a run of (I,Γ); k a point of insertion; $(I_{k,1'}, \Gamma_{k,1'}, \widetilde{\Gamma}_{k,1'})$ the state $(I_{k,1} \cup I', \Gamma_{k,1} \cup \Gamma', \widetilde{\Gamma}_{k,1})$; and $r_1 = (I_{0,1}, \Gamma_{0,1}, \widetilde{\Gamma}_{0,1}), (I_{1,1}, \Gamma_{1,1}, \widetilde{\Gamma}_{1,1})...(I_{k-1,1}, \Gamma_{k-1,1}, \widetilde{\Gamma}_{k-1,1}), (I_{k,1'}, \Gamma_{k,1'}, \widetilde{\Gamma}_{k,1'}), (I_{k+1,1'}, \Gamma_{k+1,1'}, \widetilde{\Gamma}_{k+1,1'})...$ the modified run of the system. For ease of reference, we will denote by $(I_{i,1'}, \Gamma_{i,1'}, \widetilde{\Gamma}_{i,1'})$ any state $i \geqslant 0$ of this run. We show (i) that there is a run $r_2 = (I_{0,2}, \Gamma_{0,2}, \widetilde{\Gamma}_{0,2}), (I_{1,2}, \Gamma_{1,2}, \widetilde{\Gamma}_{1,2})...$ of the system $(I \cup I', \Gamma \cup \Gamma')$ such that for each $i \geqslant 0$, $I_{i,1'} \subseteq I_{i,2}$, $\Gamma_{i,1'} \subseteq \Gamma_{i,2}$ and $\widetilde{\Gamma}_{i,1'} \subseteq \widetilde{\Gamma}_{i,2}$, and (ii) that there is a run $r_3 = (I_{0,3}, \Gamma_{0,3}, \widetilde{\Gamma}_{0,3}), (I_{1,3}, \Gamma_{1,3}, \widetilde{\Gamma}_{1,3})...$ of the system $(I \cup I', \Gamma \cup \Gamma')$ such that for each $i \geqslant 0$, $I_{i,3} \subseteq I_{i+k,1'}$, $\Gamma_{i,3} \subseteq \Gamma_{i+k,1'}$ and $\widetilde{\Gamma}_{i,3} \subseteq \widetilde{\Gamma}_{i+k,1'}$. This is sufficient to prove the result since r_2 and r_3 are both runs of the same positive system, and thus must converge (by Theorem 4.8) to the same state. Since the states of r_1 are sandwiched between those of r_2 and r_3, convergence of both r_2 and r_3 to a single state implies convergence of r_1 to this same state.

Let us consider the first assertion. We select a run of the system $(I \cup I', \Gamma \cup \Gamma')$ with exactly the same sequence of peers as the modified run r_1. For $i = 0$, the desired inclusions clearly hold. Now suppose $i > 0$. Suppose $I_{i-1,1'} \subseteq I_{i-1,2}$, $\Gamma_{i-1,1'} \subseteq \Gamma_{i-1,2}$ and $\widetilde{\Gamma}_{i-1,1'} \subseteq \widetilde{\Gamma}_{i-1,2}$. Using Lemma 4.9, if $i \neq k$, we have the desired inclusions for timepoint i. If $i = k$, we have, using Lemma 4.9, $I_{k,1} \subseteq I_{k,2}$, $\Gamma_{k,1} \subseteq \Gamma_{k,2}$ and $\widetilde{\Gamma}_{k,1} \subseteq \widetilde{\Gamma}_{k,2}$. Since $I' \subseteq I_{0,2}$ and $\Gamma' \subseteq \Gamma_{0,2}$, and since the run of (I',Γ') is monotonic (by Theorem 4.8), $I' \subseteq I_{k,2}$ and $\Gamma' \subseteq \Gamma_{k,2}$. Finally, since $I_{k,1'} = I_{k,1} \cup I'$, $\Gamma_{k,1'} = \Gamma_{k,1} \cup \Gamma'$ and $\widetilde{\Gamma}_{k,1'} = \widetilde{\Gamma}_{k,1}$, we have the result for $i = k$.

Now consider the second assertion. We choose a run r_3 of the system $(I \cup I', \Gamma \cup \Gamma')$ with exactly the same sequence of peers as the sub-run r_1 started from the timepoint k, i.e., if peer p moves at timepoint $i + k$ in r_1, then it is p who moves at timepoint i in r_3. It is clear that desired inclusions hold for $i = 0$, since the runs of (I,Γ) are monotonic. Let $i > 0$. Suppose $I_{i-1,3} \subseteq I_{i+k-1,1'}$, $\Gamma_{i-1,3} \subseteq \Gamma_{i+k-1,1'}$ and $\widetilde{\Gamma}_{i-1,3} \subseteq \widetilde{\Gamma}_{i+k-1,1'}$. Using Lemma 4.9, we obtain directly the desired inclusions for i. □

The previous theorem is straightforwardly extended to a series of updates. However, as illustrated by the following example, a more liberal definition of updates which also allows deletion of facts or rules in a system would compromise convergence.

Example 4.11. Consider the system defined as follows:

at p: extensional@p, intensional r@p

```
        r@q() :- r@p()
        r@p() :- s@p()
        s@p() :- s@p()
        s@p().
  at q: intensional r@q
        r@p() :- r@q()
```

This system converges to a state where $I^*(p) = \{s@p()\}$, $\widetilde{\Gamma}^*(p,q) = \{r@q()\text{:-}\}$, $\widetilde{\Gamma}^*(q,p) = \{r@p()\text{:-}\}$ Then removing the fact $s@p()$ or the rule $r@p()$:- $s@p()$ after the convergence will not change $\widetilde{\Gamma}$ whereas $\widetilde{\Gamma}$ would be empty were the fact or the rule removed before beginning a run.

The previous example illustrates the difficulty of managing non-monotony. If we remove a fact or a rule, we need to remove as well all facts or rules that were deduced using this fact. This could be achieved using view maintenance techniques. We leave this for future work.

To further ground our semantics, we show that for positive systems, our semantics correspond to the standard centralized Datalog semantics.

Centralized semantics

In the positive case, we can compare with a "centralized" semantics, in which all facts and rules are combined into a single Datalog program. Such a comparison would not make sense in the general case since our semantics too closely depends on the order in which peers fire.

We associate to a positive *WebdamLog* state (I,Γ) the set $\cup_p(I(p) \cup \Gamma(p))$ composed of the facts and rules of all peers. We can transform this set of facts and rules into a standard Datalog program by first instantiating the variable relations in the rules (as was done for local computation) and then removing those rules that violate the typing constraints in σ. We denote by $c(I,\Gamma)$ the Datalog program thus obtained.

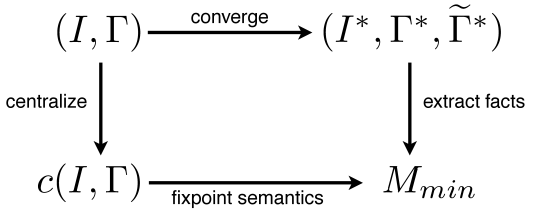

Figure 4.2: Link with centralized semantics

The following theorem (illustrated by Figure 4.2) demonstrates the equivalence, for the class of positive systems, of our distributed semantics and the traditional fixpoint semantics of Datalog. The result deals only with systems with finitely many peers to avoid having to extend Datalog to infinitely many relations.

Theorem 4.12. *Let (I, Γ) be a positive system with a finite number of peers that converges to $(I^*, \Gamma^*, \widetilde{\Gamma}^*)$, and let M_{min} be the unique minimal model of the Datalog program $c(I, \Gamma)$. Then*

$$M_{min} = \cup_p P^*_{p,d}(I^*(p))$$

where $P_{p,d}$ is the set of fully local deductive rules in $\widetilde{\Gamma}^*(p) \cup \cup_q \Gamma^*(q, p)$.

Proof. Let $S_0 = (I_0, \Gamma_0, \widetilde{\Gamma}_0)$ be a positive initial state with a finite number of peers which converges to the (finite) state $S_\infty = (I_\infty, \Gamma_\infty, \widetilde{\Gamma}_\infty)$. Let M_{min} be the unique minimal model of the Datalog program $c(I_0, \Gamma_0)$. Given a run $(I_0, \Gamma_0, \widetilde{\Gamma}_0), (I_1, \Gamma_1, \widetilde{\Gamma}_1), (I_2, \Gamma_2, \widetilde{\Gamma}_2) \ldots$, we use $P_{p,d,i}$ (resp. $P_{p,a,i}$) to refer to the set of fully local deductive (resp. local active) rules in $\Gamma_i(p) \cup \cup_q \widetilde{\Gamma}_i(q, p)$. For ease of reference, we denote by F_i the set of facts $\cup_p P^*_{p,d,i}(I_i(p))$. Our aim is to show that $M_{min} = F_\infty$.

First direction: $F_\infty \subseteq M_{min}$
Consider the run $r = (I_0, \Gamma_0, \widetilde{\Gamma}_0), (I_1, \Gamma_1, \widetilde{\Gamma}_1), (I_2, \Gamma_2, \widetilde{\Gamma}_2) \ldots$. Let p_i be the peer whose move takes the state $(I_i, \Gamma_i, \widetilde{\Gamma}_i)$ to $(I_{i+1}, \Gamma_{i+1}, \widetilde{\Gamma}_{i+1})$. We will show by induction on i that (a) $P^*_{p_i,d,i}(I_i(p_i)) \subseteq M_{min}$, (b) $P_{p_i,a,i}(P^*_{p_i,d,i}(I_i(p_i))) \subseteq M_{min}$, and (c) $M_{min} \models \widetilde{\Gamma}_{i+1}(p_i, q)$ for all $q \neq p_i$. Because of the monotonicity of states in r (cf. proof of Theorem 4.8), it follows from (a) and our definition of the sets F_i that $F_\infty \subseteq M_{min}$. Consider first the base case ($i = 0$). For (a), we note that $I_0(p_0) \cup \Gamma_0(p_0) \subseteq c(I_0, \Gamma_0)$ and $\cup_q \widetilde{\Gamma}_0(q, p_0) = \varnothing$ (since (I_0, Γ_0) is an initial state). We can thus deduce that $P^*_{p_0,d,0}(I_0(p_0)) \subseteq M_{min}$. For (b), we use (a) and the fact that $P_{p_0,a,0} \subseteq \Gamma_0(p_0)$ (as there are no delegations in the first time step). For (c), we first note that rules in $\widetilde{\Gamma}_1(p_0, q)$, are known to be of one of two types. The first type of rules are of the form

$$vA \text{ :- } vM@vQ(v\overline{U}), v\Theta_1$$

where A :- $\Theta_0, M@Q(\overline{U}), \Theta_1$ is a rule in $P_{p_0,a,0}$ and v is a valuation such that $v\Theta_0$ holds in $P^*_{p_0,d,0}(I_0(p_0))$ and $vQ = q(\neq p_0)$. In this case, the fact that $P^*_{p_0,d,0}(I_0(p_0)) \subseteq M_{min}$ ensures that $v\Theta_0$ holds in M_{min}. Since we also have $P_{p_0,a,0} \subseteq c(I_0, \Gamma_0)$, all rules in $P_{p_0,a,0}$ must holds in M_{min}, which means the partially instantiated rule vA :- $vM@vQ(v\overline{U}), v\Theta_1$ must also be satisfied by

M_{min}. All other rules in $\widetilde{\Gamma}_1(q, p_0)$ are of the form vA :- where A :- Θ is a rule in $P_{p_0,a,0}$ and v is a valuation such that $v\Theta$ holds in $P^*_{p_0,d,0}(I_0(p_0))$ and $vA = r@q(\overline{u})$ for some $r \in \mathcal{I}$. Again, the fact that $P^*_{p_0,d,0}(I_0(p_0)) \subseteq M_{min}$ means that $v\Theta$ holds in M_{min}, and the fact that $P_{p_0,a,0} \subseteq c(I_0, \Gamma_0)$ means that vA :- must hold in the minimal model M_{min}.

For the induction step, suppose our claim holds for $i \leqslant k$. Let j be such that $p_j = p_{k+1}$ and $p_{j'} \neq p_{k+1}$ for all $j < j' < k+1$, or 0 in the case where p_j has never been visited. Then it follows from our definition of moves and runs that

$$I_{k+1}(p_{k+1}) \subseteq I_j(p) \ \cup \bigcup_{j<l<k+1} P_{p_l,a,l}(P^*_{p_l,d,l}(I_l(p_l)))$$

It follows then from part (b) of the IH applied to timepoints $j, j+1, \ldots, k$ that $I_{k+1}(p_{k+1}) \subseteq M_{min}$. Part (c) of the IH applied to the timepoints in which a peer $q \neq p_{k+1}$ was last visited gives us $M_{min} \models \cup_q \widetilde{\Gamma}_{k+1}(q, p_{k+1})$. Together with the fact that $\Gamma_{k+1}(p_{k+1}) = \Gamma_0(p_{k+1}) \subseteq c(I_0, \Gamma_0)$, we obtain

$$M_{min} \models P_{p_{k+1},a,k+1} \cup P_{p_{k+1},a,k+1}$$

Parts (a) and (b) of our claim follow directly. Now for part (c), consider some rule in $\widetilde{\Gamma}_{k+2}(p_{k+1}, q)$. First consider the case where the rule is of the form

$$vA \text{ :- } vM@vQ(v\overline{U}), v\Theta_1$$

where A :- $\Theta_0, M@Q(\overline{U}), \Theta_1$ is a rule in $P_{p_{k+1},a,k+1}$ and v is a valuation such that $v\Theta_0$ holds in $P^*_{p_{k+1},d,k+1}(I_{k+1}(p_{k+1}))$ and $vQ = q(\neq p_{k+1})$. We know $P^*_{p_{k+1},d,k+1}(I_{k+1}(p_{k+1})) \subseteq M_{min}$ from part (a), so $v\Theta_0$ must hold in M_{min}. This together with the fact (from above) that $M_{min} \models P_{p_{k+1},a,k+1}$ means the partially instantiated rule vA :- $vM@vQ(v\overline{U}), v\Theta_1$ must also be satisfied by M_{min}. Suppose instead our rule is of the form vA :- where A :- Θ is a rule in $P_{p_{k+1},a,k+1}$ and v is a valuation such that $v\Theta$ holds in $P^*_{p_{k+1},d,k+1}(I_{k+1}(p_{k+1}))$ and $vA = r@q(\overline{u})$ for some $r \in \mathcal{I}$. We again utilize the fact that $P^*_{p_{k+1},d,k+1}(I_{k+1}(p_{k+1})) \subseteq M_{min}$ and $M_{min} \models P_{p_{k+1},a,k+1}$, which give $v\Theta \subseteq M_{min}$ and hence $M_{min} \models vA$:-.

Second direction: $M_{min} \subseteq F_\infty$
We proceed by induction on the depth of proof trees for facts in M_{min}. The base case is when the proof tree of a fact $r@p(\overline{u}) \in M_{min}$ has depth 0, i.e., it appears explicitly in $c(I_0, \Gamma_0)$. There are two possibilities: either $r@p(\overline{u}) \in I_0(p)$ or the rule $r@p(\overline{u})$:- appears in some $\Gamma_0(q)$. In the former case,

monotonicity (cf. proof of Theorem 4.8) ensures that $r@p(\overline{u}) \in I_\infty(p) \subseteq F_\infty$. In the latter case, if $r@p$ is extensional, then $r@p(\overline{u})$ will be sent to p the first time q is visited and will remain at p by monotonicity. If $r@p$ is an intensional relation name and $q = p$, then $r@p(\overline{u})$:- belongs to $P_{p,d,\infty}$. If $q \neq p$, then $r@p(\overline{u})$:- will be delegated to p every time q is visited, and hence will belong to $\widetilde{\Gamma}_\infty(q,p)$, and hence to $P_{p,d,\infty}$. In all cases, we obtain $r@p(\overline{u}) \in \cup_p P^*_{p,d,\infty}(I_\infty(p)) = F_\infty$.

For the induction step, suppose that all facts in M_{min} with proof trees of depth at most k appear in F_∞. Consider some fact $r@p^*(\overline{u})$ with a proof tree of depth $k+1$. Then there must exist some rule

$$\alpha = M_{n+1}@Q_{n+1}(\overline{U}_{n+1}) \text{ :- } M_1@Q_1(\overline{U}_1)...M_n@Q_n(\overline{U}_n)$$

in $\cup_p \Gamma(p)$ and some valuation v such that

$$r@p^*(\overline{u}) = vM_{n+1}@vQ_{n+1}(v\overline{U}_{n+1})$$

and for all $1 \leqslant j \leqslant n$, the fact

$$s_j@q_j(\overline{t_j}) = vM_j@vQ_j(\overline{U}_j)$$

possesses a proof tree of depth at most k. Consider some run $r = (I_0, \Gamma_0, \widetilde{\Gamma}_0)$, $(I_1, \Gamma_1, \widetilde{\Gamma}_1), (I_2, \Gamma_2, \widetilde{\Gamma}_2) \ldots$ of (I_0, Γ_0). Applying the IH, we obtain $s_j@q_j(\overline{t_j}) \in F_\infty$ for all $1 \leqslant j \leqslant n$. It follows that we can find some index m such that $s_j@q_j(\overline{t_j}) \in F_m$ for all $1 \leqslant j \leqslant n$. Because all runs of (I_0, Γ_0) converge to the same state, we can assume without loss of generality that it is a q_j-move which takes the state $(I_{m+j-1}, \Gamma_{m+j-1}, \widetilde{\Gamma}_{m+j-1})$ in r to the state $(I_{m+j}, \Gamma_{m+j}, \widetilde{\Gamma}_{m+j})$, for all $1 \leqslant j \leqslant n$. We aim to show that $r@p^*(\overline{u}) \in F_{m+n}$, hence $r@p^*(\overline{u}) \in F_\infty$. We first remark that for all peers p, the set $P^*_{p,d,m}(I_m(p))$ can only consist of p-facts. This is because $I_0(p)$ contains only p-facts (by definition), only p-facts are added to $I_i(p)$ (by definition of moves), and $P_{p,d,m}$ consists of only deductive rules in p, i.e., rules using intensional p-relations. It follows then that $s_j@q_j(\overline{t_j}) \in P^*_{q_j,d,m}(I_m(q_j))$ for all $1 \leqslant j \leqslant n$. The safety condition implies that the term Q_1 equals a peer constant q_1. We can suppose that at timepoint m, $\alpha \in \Gamma_m(q_1)$.

Then it is q_1's move. If α is fully local deductive for q_1, then p^* and all of the q_j must be equal to q_1. This means that $s_j@q_j(\overline{t_j}) \in P^*_{q_1,d,m}(I_m(q_1))$ for all j, and so $r@p^*(\overline{u}) \in P^*_{q_1,d,m}(I_m(q_1))$. Thus, $r@p^*(\overline{u}) \in F_m$, and by monotonicity of states, $r@p^*(\overline{u}) \in F_{m+n}$. Next consider the more interesting case where α is not a fully local deductive rule for q_1. Let l be the maximal index such that $q_j = q_1$ for all $1 \leqslant j \leqslant l$. Then we have $s_j@q_j(\overline{t_j}) \in$

$P^*_{q_1,d,m}(I_m(q_1))$ for all $1 \leqslant j \leqslant l$. If $l = n$, then $r@p^*(\overline{u}) \in I_{m+1}(p^*)$, and so again, by monotony, $r@p^*(\overline{u}) \in F_{m+n}$. If instead we have $l < n$, then delegation comes into play. Specifically, let v' be the minimal sub-valuation of v such that $v'M_j@v'Q_j(v'\overline{U_j}) = s_j@q_j(t_j)$ for all $1 \leqslant j \leqslant l$. Note that by the safety condition, Q_{l+1} must now be instantiated to q_l. It follows that the rule α'

$$v'M_{n+1}@v'Q_{n+1}(v'\overline{U}_{n+1}) \text{ :-}$$
$$v'M_l@v'Q_l(v'\overline{U_l})...v'M_n@v'Q_n(v'\overline{U}_n)$$

must belong to $\widetilde{\Gamma}_{m+1}(q_1, q_l)$. By monotony, $\alpha' \in \widetilde{\Gamma}_{m+l-1}(q_1, q_l)$, and $s_j@q_j(\overline{t_j})$ $\in F_{m+l-1}$ We can thus repeat the same procedure to q_l when at timepoint $m + l - 1$ it is its turn to move. We will either finish (in which case the fact $r@p^*(\overline{u})$ is derived and preserved) or continue via delegations to the next peer, and so forth, until the final peer is treated and the fact $r@p^*(\overline{u})$ has been produced. We thus find the desired $r@p^*(\overline{u}) \in F_{m+n}$. □

4.4.2 Strongly-stratified *WebdamLog*

With negation, convergence is not guaranteed in the general case as illustrated by the following example.

Example 4.13. Consider the program that is stratified in the sense of Datalog with stratified negation:

```
intensional s@p, r@p, r@q
at p: r@q() :- r@p()
      r@p() :- ¬s@p()
at q: r@p() :- r@q()
      s@p() :-
```

Any run of this system that begins with p converges to a state where p delegates $r@q()$:- to q and q delegates $r@p()$:- and $s@p()$:- to p. On the other hand, runs that begin with q converge to a state where p delegates nothing to q and q delegates $s@p()$:- to p.

As already mentioned for the non-monotone updates in the previous subsection, one may adapt methods of view maintenance to solve the problem. We develop in this section an alternative in which syntactic restrictions prohibit circles of wrong deductions, without having to deal with the complexity of view maintenance in presence of belief revision. Note that most of the examples of the paper belong to (or are easily adapted to) this restricted class.

A *stratification* σ' is an assignment of numbers to relations, i.e., to pairs $r@p$. If $\sigma'(r@p) = i$, we say that $r@p$ is in the ith stratum. The stratification is *strong* if for each i, all the relations in the ith stratum refer to the same peer. Given a strong stratification σ', an instantiated rule is σ'-stratified if all relation names of positive body atoms appear in a stratum smaller or equal to that of the head relation and all relation names of negative terms belong to a strictly smaller stratum. Note that a stratification for Example 4.13 would not be strong because $r@p$ and $r@q$ have to be in the same stratum, although they belong to different peers.

In our setting, we see a strong stratification σ' of $\mathcal{I}$ as an extra component of the system's schema. The strong stratification works much like the typing constraint σ in that it tells us whether a particular rule instantiation is legal. Specifically, a peer is only allowed to use instantiated rules which are σ'-stratified. Observe that our use of stratification is in the spirit of classical Datalog with stratified negation, namely preventing cycling through negation. However, the way stratification is enforced is somewhat different. In the centralized context, one analyzes the program and checks for the existence of a stratification. In the distributed case, this is not possible because no one has access to the entire program. Also, the use of relation and peer variables makes such a computation even less conceivable. So, instead, one assumes that a stratification is imposed and the computation is such that it prevents deriving facts with rule instantiations that would violate the strong stratification.

There is a subtlety with strong stratification arising from general delegation. Indeed, we will see that the result does not hold for WL. So the next result deals simply with view delegation, i.e., the language VWL. One of the advantages of VWL is that at the time a rule is delegated, it is possible to check that it does not violate the strong stratification. We consider systems with finitely many peers, where the extensional facts are constant and only the intensional delegations vary. Formally, a *WebdamLog* system is said to be *strongly-stratified* if for some strong stratification σ':

1. its local computation is constrained by the stratification σ'.

2. Each extensional relation $m@p$ is made persistent with a rule of the form $m@p(\overline{U})$:- $m@p(\overline{U})$ and these are the only active rules in the system[1]. We say the system is *purely intensional*.

Observe that, by Condition (2), the set of extensional facts is constant whereas it was increasing for positive systems. So Condition (2) here is

[1]Technically speaking, if we want to use variable or peer relations in the rule heads, then we must forbid instantiations which yield extensional relations in the heads.

more restrictive than for positive systems. Thus, strictly speaking the two classes are incomparable. Clearly, it would be interesting to consider classes that would include both.

We are now ready to present our results, following the same logic as in the previous section.

Theorem 4.14 (Convergence)**.** *All strongly-stratified VWL systems over a finite number of peers converge.*

Proof. Let us first remark that deductive rules in SWL can only be of two types: fully local deductive or local deductive. This means that the only types of rules that can be delegated to a peer p are fully instantiated bodyless rules of the form $r@p(\overline{u})$:- . The general idea of the proof is as follows. Given a run, we will prove that for each stratum, there is a state after which the stratum has converged. A similar argument will prove that the limit is the same for each run.

Consider a σ'-stratified system (I_0, Γ_0) with rules in VWL and a finite number of peers. Let $r = (I_0, \Gamma_0, \varnothing)(I_1, \Gamma_1, \widetilde{\Gamma}_1)(I_2, \Gamma_2, \widetilde{\Gamma}_2)\ldots$ be a run of this system. For simplicity, in what follows, we use $P_{p,d,i}$ to refer to the set of fully local deductive rules in $\Gamma_i(p) \cup \cup_q \widetilde{\Gamma}_i(q, p)$.

First, we can show by induction that for all $i \geqslant 0$, every state $(I_i, \Gamma_i, \widetilde{\Gamma}_i)$ is intensional, $I_i = I_0$, and $\Gamma_i = \Gamma_0$. The base case $i = 0$ is immediate. For the induction step, suppose we have the result for $i < k$ and consider state $(I_k, \Gamma_k, \widetilde{\Gamma}_k)$ resulting from a p-move. From the IH, we know that $(I_{k-1}, \Gamma_{k-1}, \widetilde{\Gamma}_{k-1})$ is intensional, and so the only active rules in $\Gamma_{k-1}(p)$ and $\cup_q \widetilde{\Gamma}_{k-1}(q, p)$ are persistence rules for p's extensional predicates. We also have $\Gamma_k = \Gamma_0$ from the definition of runs. In particular, this means that $\Gamma_k(p)$ contains persistence rules for each of p's extensional predicates. This means that p copies its extensional facts ($I_k(p) = I_{k-1}(p)$) and does not send any extensional facts to other peers ($I_k(q) = I_{k-1}(q)$ for $q \neq p$). We thus have $I_k = I_{k-1} = I_0$. Finally, we note that $(I_{k-1}, \Gamma_{k-1}, \widetilde{\Gamma}_{k-1})$ contains no other active rules besides persistence rules, which means that all delegations will involve deductive rules.

Given the strong stratification σ', let us prove that for each stratum i, there is a timepoint $t_i \geqslant 0$ such that after each $t \geqslant t'$, the restriction of $\widetilde{\Gamma}_t$ to rules with head in strata less or equal to i is the same as the one of $\widetilde{\Gamma}_{t_i}$. Let us start with the first stratum, call it 0. Suppose that p^* is the peer associated with this stratum. Let t_0 be the first occurrence of a p^*-move after visiting all the other peers. Such a timepoint must exist since the number of peers is finite (this is assumed in the statement of the theorem) and the run is fair. We claim that t_0 has the desired properties. Consider some

timepoint $t \geqslant t_0$ in which it's peer q's turn to move and some delegation appearing in $\widetilde{\Gamma}_{t+1}(q, p^*)$. We remark that because we only have VWL rules, the delegation must be of the form $r@p^*(\overline{u})$:-. To produce this delegation, there must be a rule in $\Gamma_t(q) = \Gamma_0(q)$ of the following form

$$M_{n+1}@Q(\overline{U}_{n+1}) :-$$
$$(\neg)M_1@q(\overline{U}_1), (\neg)M_2@q(\overline{U}_2), ...(\neg)M_n@q(\overline{U}_n)$$

and some valuation v satisfying the typing σ and stratification σ' such that: $vM_{n+1}@vQ(v\overline{U}_{n+1}) = r@p^*(\overline{u})$, each positive body fact $vM_i@q(v\overline{U}_i)$ belongs to $P^*_{q,d,t}(I_t(q))$, and each negated body fact $\neg vM_i@q(v\overline{U}_i)$ is such that $vM_i@q(v\overline{U}_i)$ is not in $P^*_{q,d,t}(I_t(q))$. We note however that because v satisfies the strong stratification, we are at peer $q \neq p$, and the head relation $r@p$ is in the lowest stratum, all relations $vM_i@q$ must be extensional. As the extensional facts of each peer are the same at each timepoint (see above), it follows that this delegation is produced at each and every visit to q, and in particular the very first visit to q, which occurs before t_0. Thus, this delegation already appears in $\widetilde{\Gamma}_{t_0}(q, p^*)$. A very similar argument shows that every delegation concerning stratum 0 which appears in $\widetilde{\Gamma}_{t_0}(q, p^*)$ also appears in $\widetilde{\Gamma}_t(q, p^*)$ for all $t \geqslant t_0$.

Now let us consider higher strata. Suppose our claim holds for strata up to and including k. This means we can find a timepoint t_k such that for all $t \geqslant t_k$, the restriction of $\widetilde{\Gamma}_t$ to rules with head in strata less or equal to k is the same as the one of $\widetilde{\Gamma}_{t_k}$. Again, we use p^* to refer to the peer associated with the stratum of interest (here $k+1$). Set t_{k+1} equal to the timepoint after t_k in which we first visit p^* after having visited all other peers at least once since timepoint t_k. Consider some timepoint $t \geqslant t_0$ in which q moves and produces some delegation in $\widetilde{\Gamma}_{t+1}(q, p^*)$. Again, because we only have VWL rules, we know this delegation must be of the form $r@p^*(\overline{u})$:-. To produce it, there must be a rule in $\Gamma_t(q) = \Gamma_0(q)$ of the following form

$$M_{n+1}@Q(\overline{U}_{n+1}) :-$$
$$(\neg)M_1@q(\overline{U}_1), (\neg)M_2@q(\overline{U}_2), ...(\neg)M_n@q(\overline{U}_n)$$

and some valuation v satisfying the typing σ and stratification σ' such that: $vM_{n+1}@vQ(v\overline{U}_{n+1}) = r@p^*(\overline{u})$, each positive body fact $vM_i@q(v\overline{U}_i)$ belongs to $P^*_{q,d,t}(I_t(q))$, and each negated body fact $\neg vM_i@q(v\overline{U}_i)$ is such that $vM_i@q(v\overline{U}_i)$ is not in $P^*_{q,d,t}(I_t(q))$. Because v satisfies the strong stratification, we are at peer $q \neq p$, and the head relation $r@p$ is in the lowest stratum, we know all body facts $vM_i@q(v\overline{U}_i)$ must either be extensional

or intensional but in a lower stratum ($\leqslant k$). We have already seen that extensional facts are fixed throughout the run. Since $t \geqslant t_{k+1} > t_k$, we know that all delegations for strata less than or equal to k are fixed and equal to those found at timepoint t_k. It follows that this delegation is produced at each and every visit to q following timepoint t_k, and hence in the visit to q between timepoints t_k and t_{k+1}. Thus, this delegation already appears in $\widetilde{\Gamma}_{t_{k+1}}(q, p^*)$. We can similarly show that all delegations stratum $k+1$ delegations in $\widetilde{\Gamma}_{t_{k+1}}(q, p^*)$ are also found in $\widetilde{\Gamma}_t(q, p^*)$ for all $t \geqslant t_{k+1}$.

We now prove that all systems converge to the same limit. In fact, we can straightforwardly extend the previous proof by adding to the claim that each stratum $k+1$ converges to the same value on all runs. In the base case, we use the fact that the extensional facts are the same in all runs. This means delegations for the first stratum will be the same for all runs. For later strata, we use the fact that the delegations at level $k+1$ are fully determined by the delegations in previous strata. □

This result does not hold if we allow general delegation instead of view delegation. This is because with general delegation, a peer p may delegate a partially instantiated rule to q. As the relation and peer terms of the rule may contain variables, peer p may not be able to decide whether the rule is σ'-stratified, and neither will q (or later peers) as they do not know which relations p used to launch the delegation. So enforcement of the stratification is not straightforward. This is illustrated by the following example.

Example 4.15. Consider the following program:

```
intensional m@p, s@q, r@q
at p:  m@p($x) :- m@p($x), r@q($x)
       m@p($x) :- r@q($x), ¬s@q()
at p′: s@q() :-
at q:  r@q(a) :-
```

Consider a run that starts by firing p, q, then p. Then the rule $m@p(a)$:- is delegated by q to p and will remain forever. Now, consider a run that starts by firing p'. Then q will know $s@q()$:-. from the beginning and will never delegate $m@p(a)$:-.

Convergence also holds for strongly-stratified VWL systems in the presence of insertions as well as deletions.

Theorem 4.16 (Update)**.** *Let (I,Γ) be a VWL system with strong stratification σ' over a finite number of peers. Consider (I^+,I^-, Γ^+,Γ^-) where I^+, I^- are sets of extensional facts and Γ^+,Γ^- are sets of deductive rules. For each run of the*

system (I,Γ), if for some k a given state (I_k,Γ_k, $\widetilde{\Gamma}_k$) is replaced by ($I_k \cup I^+ \setminus I^-$, $\Gamma_k \cup \Gamma^+ \setminus \Gamma^-, \widetilde{\Gamma}_k$), then the modified run converges to the convergence state of the σ'-stratified system ($I \cup I^+ \setminus I^-$, $\Gamma \cup \Gamma^+ \setminus \Gamma^-$).

Proof. First, it is straightforward to show that ($I \cup I^+ \setminus I^-$,$\Gamma \cup \Gamma^+ \setminus \Gamma^-$) respects the constraints of intensional states. Let us recall from the proof of Theorem 4.14 that until the insertion point k, $I_k = I$ and $\Gamma_k = \Gamma$. So at the end of the timepoint k, the state is indeed ($I \cup I^+$ I^-, $\Gamma \cup \Gamma^+ \setminus \Gamma^-, \widetilde{\Gamma}_k$). Then observe that the proof never used the fact that $\widetilde{\Gamma}$ was initially empty, except to prove that the initial state was intensional. So the proof applies as it is and gives the desired result. □

This theorem can obviously be generalized to any sequence of updates. The final theorem of this section shows that the set of facts computed by a σ'-stratified system corresponds to the set of facts in the minimal model of a centralized version of the system. As in the previous section, we associate a σ'-stratified *WebdamLog* system (I, Γ) with the set $\cup_p (I(p) \cup \Gamma(p))$ composed of the facts and rules of all peers. We then transform this set of facts and rules into a standard Datalog program by instantiating the variable predicates in the rules and removing rules which violate the typing constraints σ or the strong stratification σ'. We use $c_s(I, \Gamma)$ to refer to the resulting Datalog program.

Theorem 4.17 (Centralized)**.** *Let (I, Γ) be a σ'-stratified system with a finite number of peers and rules in SWL, which converges to $(I^*, \Gamma^*, \widetilde{\Gamma}^*)$, and let M_{min} be the unique minimal model of the Datalog program $c_s(I, \Gamma)$. Then*

$$M_{min} = \cup_p P^*_{p,d}(I^*(p))$$

where $P^*_{p,d}$ is the set of fully local deductive rules in $\widetilde{\Gamma}^*(p) \cup \cup_q \Gamma^*(q, p)$.

Proof. Let $S_0 = (I_0, \Gamma_0, \widetilde{\Gamma}_0)$ be a strongly stratified VWL system (with strong stratification σ') which converges to the finite state $S_\infty = (I_\infty, \Gamma_\infty, \widetilde{\Gamma}_\infty)$. As the rules in the Datalog program $c_s(I_0, \Gamma_0)$ are stratified with respect to σ' (by construction), we can be sure that there is a unique minimal model of $c_s(I_0, \Gamma_0)$. We use M_{min} to denote this minimal model. Given a run $(I_0, \Gamma_0, \widetilde{\Gamma}_0), (I_1, \Gamma_1, \widetilde{\Gamma}_1), (I_2, \Gamma_2, \widetilde{\Gamma}_2) \ldots$ of our system, we use $P_{p,d,i}$ to refer to the set of fully local deductive rules in $\Gamma_i(p) \cup \cup_q \widetilde{\Gamma}_i(q, p)$. For ease of reference, we denote by F_i the set of facts $\cup_p P^*_{p,d,i}(I_i(p))$. Our aim is to show that $M_{min} = F_\infty$.

We first note that the desired equality holds if we consider only extensional facts. This is because the only rules with extensional heads in Γ_0 are

extensional persistence rules. Thus, the extensional facts in F_∞ are precisely the original extensional facts $\cup_p I_0(p)$. The Datalog program $c_s(I_0, \Gamma_0)$ will contain these extensional facts, and will not contain any rules to create new extensional facts, so the extensional facts in M_{min} will be exactly $\cup_p I_0(p)$.

It thus remains to show the equality for intensional facts. The proof will proceed by induction on the strata of facts. In what follows, we will use the integers $0, 1, 2, \ldots$ to label the strata, with 0 being the lowest stratum. Also, given a set S of facts, we denote by $S[i]$ the set of facts whose relations belong to strata lower than or equal to i.

Base Case: $M_{min}[0] = F_\infty[0]$

First direction ($F_\infty[0] \subseteq M_{min}[0]$). Let us consider some intensional fact $r@p(\overline{u})$ from stratum 0 which belongs to F_∞, and hence more precisely to $P^*_{p,d,\infty}(I_\infty(p))$. We know that the set $P_{p,d,\infty}$ consists of fully local deductive rules from $\Gamma_\infty(p) = \Gamma_0(p)$ and delegated body-less rules $\cup_q \widetilde{\Gamma}_\infty(q, p)$. Moreover, we have seen in the proof of Theorem 4.14 that each body-less delegation with head relation in stratum 0 from a peer q results from evaluating the extensional q-facts present in the initial state using the instantiation of a local rule in Γ_q which respects σ and σ'. As the extensional q-facts in M_{min} are precisely those found in the initial state, and all well-typed rules from $\Gamma_0(q)$ respecting σ' can be found in $c_s(I_0, \Gamma_0)$, it follows that the delegated rule is entailed by M_{min}. Thus, all (well-typed and properly stratified) instantiations of rules in $P_{p,d,\infty}$ with heads of stratum 0 are entailed by M_{min}, and so are all extensional facts in $I_\infty(p)$. It follows that the fact $r@p(\overline{u})$ must belong to M_{min}.

Second direction ($M_{min}[0] \subseteq F_\infty[0]$). Consider some intensional fact $r@p(\overline{u})$ from stratum 0 which belongs to M_{min}. The proof proceeds by induction on the depth of the proof tree of $r@p(\overline{u})$. The base case is when $r@p(\overline{u})$ has depth 0, i.e., it appears explicitly in $c_s(I_0, \Gamma_0)$. There are two possibilities: either $r@p(\overline{u}) \in I_0(p)$ or the rule $r@p(\overline{u})$:- appears in some $\Gamma_0(q)$. In the former case, we know from the proof of Theorem 4.14 that $I_\infty = I_0$, so we must have $r@p(\overline{u}) \in F_\infty$. In the latter case, as we are in an intensional system, the rule $r@p(\overline{u})$:- must be deductive. Either this rule appears in $\Gamma_0(p)$ (hence $\Gamma_\infty(p)$) or it will be delegated to p by another peer q at every visit to q, and thus will appear in $\widetilde{\Gamma}_\infty(q, p)$. In both cases, the rule must belong to $P_{p,d,i}$, hence $r@p(\overline{u}) \in P^*_{p,d,i}(I_i(p)) \subseteq F_\infty$. Now suppose the proof tree of fact $r@p(\overline{u})$ has depth $d + 1$, and we already have the result for facts of stratum 0 with proof trees of depth at most d. Let β be the rule

in $c_s(I_0, \Gamma_0)$ which was used for the last step of the proof of $r@p(\overline{u})$. As (I_0, Γ_0) is an intensional VWL system, it follows that all rules in (I_0, Γ_0) are of one of two types: persistence rules for extensional predicates, or local deductive rules. Thus, the rule β must be of the form

$$vM_{n+1}@vQ(v\overline{U}_{n+1}) \text{ :-}$$
$$(\neg)vM_1@q(v\overline{U}_1), (\neg)vM_2@q(v\overline{U}_2), ...(\neg)vM_n@q(v\overline{U}_n)$$

for some rule ρ

$$M_{n+1}@Q(\overline{U}_{n+1}) \text{ :-}$$
$$(\neg)M_1@q(\overline{U}_1), (\neg)M_2@q(\overline{U}_2), ...(\neg)M_n@q(\overline{U}_n)$$

in $\Gamma_0(q)$ and some valuation v which respects the typing constraints σ and the strong stratification σ', and is such that $vM_{n+1}@vQ = r@p$. Note in particular that this means that each of the (ground) relations $vM_j@q$ must be extensional or belong to the same stratum (0) as $r@p$. If there are any facts from the stratum 0 in the body, then they must use a relation with peer p, and so we would have $q = p$ (since only local deductive rules are permitted). Otherwise, if $q \neq p$, then only extensional relations may be used in the body. Also note that all atoms in the body which belong to stratum 0 must not be negated. We know that the rule β was used to derive the fact $r@p(\overline{u})$. This means that there must be a second valuation v' such that $v'vM_{n+1}@v'vQ(v'v\overline{U}_{n+1}) = r@p(\overline{u})$ and each literal $(\neg)vM_i@q(v'v\overline{U}_i)$ is either extensional and satisfied by the set of extensional facts or a positive atom of stratum 0 which has a proof tree of depth at most k. As F_∞ and M_{min} agree on all extensional facts, all extensional literals $(\neg)vM_i@q(v'v\overline{U}_i)$ are satisfied by $P^*_{q,d,\infty}(I_\infty(q))$. For the remaining body atoms, we use the IH to infer that each atom $vM_i@q(v'v\overline{U}_i)$ of stratum 0 belongs to F_∞, and more specifically to $P^*_{q,d,\infty}(I_\infty(q))$. If $q = p$, then we can use the rule ρ in $\Gamma_\infty(p) = \Gamma_0(p)$ together with the valuation $v'' = v'v$ and the facts $vM_i@p(v'v\overline{U}_i) \in P^*_{p,d,\infty}(I_\infty(p))$ to obtain $r@p(\overline{u}) \in P^*_{p,d,\infty}(I_\infty(p))$. If $q \neq p$, then we know from above that each $vM_i@q(v'v\overline{U}_i)$ must be an extensional fact and it must belong to $P^*_{q,d,\infty}(I_\infty(q))$. It follows that q must delegate the rule $r@p(\overline{u})$:- to p. The fact that the run has converged to $(I_\infty, \Gamma_\infty, \widetilde{\Gamma}_\infty)$ means that this delegation must appear in $\widetilde{\Gamma}_\infty$). It follows that $r@p(\overline{u})$:- belongs to $P_{p,d,i}$, hence $r@p(\overline{u}) \in P^*_{p,d,i}(I_i(p)) \subseteq F_\infty$.

Induction Step: show $M_{min}[k+1] = F_\infty[k+1]$ assuming $M_{min}[k] = F_\infty[k]$

First direction ($F_\infty[k+1] \subseteq M_{min}[k+1]$). We suppose that $F_\infty[k] \subseteq M_{min}[k]$. Let us consider some intensional fact $r@p(\overline{u})$ from stratum $k+1$ which belongs to F_∞, and hence to $P^*_{p,d,\infty}(I_\infty(p))$. We know that the set $P_{p,d,\infty}$ consists of fully local deductive rules from $\Gamma_\infty(p) = \Gamma_0(p)$ and delegated body-less rules from $\cup_q \widetilde{\Gamma}_\infty(q,p)$. As for the delegated rules, note that if $s@p(\overline{w})$:- appears in $\widetilde{\Gamma}_\infty(q,p)$, there must exist a rule in $\Gamma_\infty(q) = \Gamma_0(q)$ of the form

$$M_{n+1}@Q(\overline{U}_{n+1}) :-$$
$$(\neg)M_1@q(\overline{U}_1), (\neg)M_2@q(\overline{U}_2), ...(\neg)M_n@q(\overline{U}_n)$$

and a valuation v satisfying the typing σ and strong stratification σ' such that: $vM_{n+1}@vQ(v\overline{U}_{n+1}) = s@p(\overline{w})$, each fact $vM_i@q(v\overline{U}_i)$ appearing positively in the body belongs to $P^*_{q,d,\infty}(I_\infty(q))$ (and hence to F_∞), and each negated fact $\neg vM_i@q(v\overline{U}_i)$ in the body does not appear in $P^*_{q,d,\infty}(I_\infty(q))$ (nor a fortiori in F_∞). Because v respects the strong stratification σ', and $q \neq p$, we know that every relation $vM_i@q$ is either extensional or must belong to a stratum k or less. From the IH, we know that M_{min} and F_∞ agree on all intensional facts appearing in strata up to and including k, and we have seen earlier in the proof that the same is true for extensional facts. It follows that each fact $vM_i@q(v\overline{U}_i)$ appearing positively in the body belongs to M_{min}, and each negated fact $\neg vM_i@q(v\overline{U}_i)$ in the body does not appear in M_{min}. Moreover, we know that the instantiated rule used to produce the delegation is entailed by M_{min}. Thus, we have that M_{min} entails the delegation $s@p(\overline{w})$:-. Thus, all (well-typed and properly stratified) instantiations of rules in $P_{p,d,\infty}$ whose head relations are in strata at $k+1$ are entailed by M_{min}. Moreover, we know that only (well-typed and stratified) instantiations of rules in $P_{p,d,\infty}$ with head relations in stratum $k+1$ or lower are used in the production of $r@p(\overline{u})$. Finally, we know that all extensional p-facts in $I_\infty(p) = I_0(p)$ belong to M_{min}. It follows that the fact $r@p(\overline{u})$ belongs to M_{min}.

Second direction ($M_{min}[k+1] \subseteq F_\infty[k+1]$). Consider some intensional fact $r@p(\overline{u}) \in M_{min}$ from the stratum $k+1$. As σ' provides a stratification of $c_s(I_0, \Gamma_0)$, it is possible to find a proof tree for $r@p(\overline{u})$ whose leaves use only (negations of) facts in M_{min} belonging to strata $\leqslant k$. We will thus again proceed by induction on the depth of such a proof tree. The base case is when the proof tree for $r@p(\overline{u})$ has depth 0, i.e., it appears explicitly in $c_s(I_0, \Gamma_0)$. We can then proceed as in the base case for stratum 0. Suppose next that we have already shown the result for intensional facts in M_{min}

belonging to stratum $k+1$ and having proof trees from facts in strata $\leqslant k$ of depth at most d. Consider $r@p(\overline{u}) \in M_{min}$ from the stratum $k+1$ with a proof tree of depth $d+1$. Let β be the rule in $c_s(I_0, \Gamma_0)$ which was used for the last step of the proof. As we saw earlier, β must be of the form

$$vM_{n+1}@vQ(v\overline{U}_{n+1}) \text{ :-}$$
$$(\neg)vM_1@q(v\overline{U}_1), (\neg)vM_2@q(v\overline{U}_2), ...(\neg)vM_n@q(v\overline{U}_n)$$

for some rule ρ

$$M_{n+1}@Q(\overline{U}_{n+1}) \text{ :-}$$
$$(\neg)M_1@q(\overline{U}_1), (\neg)M_2@q(\overline{U}_2), ...(\neg)M_n@q(\overline{U}_n)$$

in $\Gamma_0(q)$ and some valuation v which respects the typing constraints σ and the strong stratification σ' and such that $vM_{n+1}@vQ = r@p$. It follows that each (ground) relation $vM_i@q$ is either extensional or an intensional relation which belongs to a stratum lower than or equal to $k+1$. We also know that β was used to derive the fact $r@p(\overline{u})$, which implies the existence of a second valuation v' such that $v'vM_{n+1}@v'vQ(v'v\overline{U}_{n+1}) = r@p(\overline{u})$ and each literal $(\neg)vM_i@q(v'v\overline{U}_i)$ is either (i) a (possibly negated) extensional fact which is satisfied by M_{min}, (ii) a (possibly negated) intensional fact from some stratum $\leqslant k$ which holds in M_{min}, or (iii) a non-negated intensional fact from stratum $k+1$ with a proof tree of depth at most k. We know from earlier in the proof that F_∞ and M_{min} agree on extensional facts. This means that every non-negated extensional fact $vM_i@q(v'v\overline{U}_i)$ belongs to F_∞ (more precisely $P^*_{q,d,\infty}(I_\infty(q))$) and every negated extensional fact $\neg vM_i@q(v'v\overline{U}_i)$ does not belong to $P^*_{q,d,\infty}(I_\infty(q))$.

For intensional facts from lower strata (k or less), we use the induction hypothesis (from the initial induction over strata) to obtain $F_\infty[k] = M_{min}[k]$. From this we can deduce that an intensional fact $vM_i@q(v'v\overline{U}_i)$ of stratum $\leqslant k$ which appears positively in the body of our rule must belong to F_∞ (or more specifically $P^*_{q,d,\infty}(I_\infty(q))$), and if it appears negatively in the rule, then it will not belong to $P^*_{q,d,\infty}(I_\infty(q))$.

Finally, if we have a non-negated intensional fact $vM_i@q(v'v\overline{U}_i)$ from stratum $k+1$ with a proof tree of depth at most k, then using the (local) IH, we obtain $vM_i@q(v'v\overline{U}_i) \in F_\infty$, and hence $vM_i@q(v'v\overline{U}_i) \in P^*_{q,d,\infty}(I_\infty(q))$. If we are in the case where $p = q$, then we can use the rule ρ in $\Gamma_\infty(p) = \Gamma_0(p)$ together with the valuation $v'' = v'v$ and the facts $vM_i@p(v'v\overline{U}_i) \in P^*_{p,d,\infty}(I_\infty(p))$ to obtain $r@p(\overline{u}) \in P^*_{p,d,\infty}(I_\infty(p) \subseteq F_\infty)$. If $q \neq p$, then because we respect the strong stratification, we know that each $vM_i@q(v'v\overline{U}_i)$

must be either an extensional fact or an intensional fact from a stratum $\leqslant k$. In both cases, we have shown above that $vM_i@q(v'v\overline{U}_i)$ belongs to $P^*_{q,d,\infty}(I_\infty(q))$ when $vM_i@q(v'v\overline{U}_i)$ appears positively in the rule, and $vM_i@q(v'v\overline{U}_i)$ does not belong to $P^*_{q,d,\infty}(I_\infty(q))$ when it is appears negatively. Thus, the body of the rule is satisfied by $P^*_{q,d,\infty}(I_\infty(q))$. It follows that q must delegate the rule $r@p(\overline{u})$:- to p. The fact that the run has converged to $(I_\infty, \Gamma_\infty, \widetilde{\Gamma}_\infty)$ means that this delegation must appear in $\widetilde{\Gamma}_\infty$). It follows that $r@p(\overline{u})$:- belongs to $P_{p,d,i}$, hence $r@p(\overline{u}) \in P^*_{p,d,i}(I_i(p)) \subseteq F_\infty$. □

4.5 Optimization

To make the approach feasible, we have to rely intensively on some known optimization techniques. We briefly mention them next and see how they fit in the *WebdamLog* picture.

4.5.1 Differential technique

Consider a peer p who has the rule $s@q(x, y)$:- $r@p(x, y)$ with $s@q$ an extensional relation. Suppose that $r@p$ is a very large relation that changes infrequently. Each time we visit p we have to send to q the current version of $r@p$, say a set K_n of tuples. This is a clear waste of communication resources. It is preferable to send the symmetric difference of $r@p$, i.e., send a set of updates Δ with the semantics that $K_n = \Delta(K_{n-1})$, since q already knows K_{n-1}. If $s@q$ is intensional, we face a similar issue; it is preferable to send the new set of delegation rules as Δ rather than sending the entire set.

4.5.2 Seed-based delegation

Consider again the rule:

at p: m@q() :- m_1@p($x), m_2@p'($x)

Now suppose that $m_1@p(a_i)$ holds for $i = [1..1000]$. We need to install 1000 rules. However, in this particular case, we can install a single rule at p' and send many facts:

at p': m@q() :- $seed_{r,1,p}$@p'($x), m_2@p'($x)
at p': $seed_{r,1,p}$@p'(a_i). (for each i)

Note that it now becomes natural to use a differential technique to maintain delegation. In particular, if the delegation from p to q does not change,

there is no need to send anything. If it does, one needs only to send the delta on $seed_{r,1,p}@p'$. Observe that we have replaced the task of installing and uninstalling delegation rules by that of sending insertion and deletion messages in a persistent extensional (seed) relation that controls a rule.

4.5.3 Query-subquery and delegation

Consider the following example of a rule in Bob's iPhone, where photosAlice@Bob-iPhone is intensional:

```
at Bob-iPhone: photosAlice@Bob-iPhone($X,$Y) :-
                   photos@picasa(Alice,$X,$Y)
```

This rule says that to find the photos of Alice, one needs to ask Picasa. The formal semantics says that we install the following [*Upload*] rule at Picasa:

```
at Picasa: photosAlice@Bob-iPhone($X,$Y) :-
               photos@picasa(Alice,$X,$Y)
```

which will result in uploading in Bob-iPhone all the photos. However, observe that this has no effect on the state since *photosAlice* is only intensional. This uploading may therefore be considered a waste of resources. An optimizer may decide not to install the [*Upload*] rule at Picasa, i.e., not ask Picasa to upload anything. Now suppose that Bob asks his iPhone for the photos of Sue:

```
query@Bob-iPhone($X) :- photosAlice@Bob-iPhone($X, Sue)
```

where *query* is an extensional predicate. Now obtaining photos from Picasa changes the state. So the optimizer will install on Picasa the rule:

```
at Picasa: photosAlice@Bob-iPhone($X,Sue) :-
               photos@picasa(Alice,$X,Sue)
```

Observe that the optimizer performed some form of resolution in the spirit of query-subquery [Vie86] or rewriting in the Magic Set style [BMSU86] (see also [AHV95]). Indeed, the entire management of delegation can be optimized using these techniques. Note that strictly speaking this may change the semantics of applications: the derivation of some facts may take a little longer than if we had installed all the delegations in advance.

4.6 Conclusion

We have introduced a new Datalog-style language for distributed data management. The main novelty is the notion of delegation that allows a peer to install rules at other peers. We have studied the expressivity of the language and of restrictions. We have also studied convergence properties for fragments of the language.

One should observe that the power of delegation critically depends on the exact definition of the model. The situation would be different, for instance, if we were to consider an asynchronous version of the model in which messages between peers are not instantaneous. A natural direction for future work is the extension of our study of the power of delegation and related issues (e.g., possibility of electing a leader) to different variants of the model.

As another possible direction for future work, Active XML considers intensional data of a very different form, namely functions that may be included in documents and are defined intensionally. It would be interesting to investigate the relationships between these two kinds of intensional data.

Perhaps one of the most exciting direction of work open by *WebdamLog* is the possibility to describe Web data management tasks in a well-defined language. Indeed, we believe that one needs such a distributed rule-base language to focus on the most fundamental and common algorithms of distributed data management, by abstracting away most specific implementation details. This opens new well-founded avenues to tackle the apparent heterogeneity of the Web. It is of particular interest for access control and distribution management, as we illustrate in the next chapter while introducing *WebdamExchange*.

Chapter 5

A data model for Web data exchanges

Part of this work has been carried out in collaboration with Serge Abiteboul, Neoklis Polyzotis and Amélie Marian and presented in [AGP11].

With the Web, notably social networks and Web 2.0 applications, the sharing of information is generalizing. Users bring data to the network and are willing to share with others, but also wish to control what portions of the data can be viewed or updated by others. Users would also like to access and update information if desired and entitled to. This is the setting of the present chapter, namely *the specification and sharing of information with access control in a distributed environment*. We wish to do so with a similar level of security as in centralized systems, but we also want to leverage and accommodate the wide variety of systems already available on the Web.

Many studies have investigated the problem of distributed information management with access control. See, e.g., [MKKW99, REG+03, RD01, WABL94]. Similar to some of the previous studies, our approach, which we term *WebdamExchange*, uses a distributed knowledge base as its foundation. However, the originality of *WebdamExchange* is that the knowledge base unifies a wide range of information relevant to data management with access rights. Specifically, the knowledge base contains logical statements and rules to encode:

1. data (as in databases),
2. knowledge about other peers (e.g., replication of their data or trust in it),
3. access rights, and credentials (e.g., cryptographic keys pair or login/password),

4. localization information (where to find some particular data)
5. rules describing the policy of each peer,
6. the provenance of the information using time and trace of communication, and
7. possibly other kinds of knowledge that we will not discuss here such as ontologies, ontology mappings, beliefs, trust, etc.

The approach allows reasoning holistically about pieces of data, e.g.to determine from where they can be retrieved or who has some particular access right on them. Moreover, the "logic" of participants may be described declarativly using rules, which facilitates the development of distributed Web applications.

We illustrate these ideas with our motivating example. Recall that user Alice is organizing a rock-climbing outing in Fontainebleau, and wishes to put together an application for the event that she will share with the members of her rock-climbing group, Roc14. Part of the data she needs is the list of the group members, which is stored on Facebook. To promote the event, she also wants to use pictures from previous outings, which the group members store on public sites, such as Picasa or Flickr, private Websites and in an untrusted DHT that group member Bob set up. Finally, some information she will need can be obtained from public Web-services, e.g., she might use Google maps to obtain location information for bouldering areas in Fontainebleau. Using existing technology, Alice will have to use many different tools and APIs in order to check what data is available and from which Web-services, whether she has access to it and finally determine how to retrieve it. In contrast, the same task can be performed in *WebdamExchange* by issuing a declarative query that requests the needed data. *WebdamExchange* will process the query over the unified knowledge base, thus dealing with the thorny issues of distribution and access rights. For instance, the knowledge base provides all information about obtaining (from Facebook) the list of group members, finding where each member stores outing pictures and getting the data using proper credentials. Note that *WebdamExchange* has to perform this reasoning in an extremely heterogeneous setting, where systems, access controls and ontologies (data organization) may vary widely across members of the group.

From a formal viewpoint, the system consists of a set of *peers*, each with its own database and its own logic. The database contains logical facts capturing information such as documents, access control, credentials, localization, but also replicas of other peers' information. A peer logic is

expressed in datalog-style rules. We build on the *WebdamLog* language, presented in the previous chapter. We extend the language in a number of ways, notably by introducing the notion of *principal* (e.g., the group Roc14) which is common in security.

The holistic approach proposed by *WebdamExchange* brings several distinct advantages:

Large spectrum Because the model is general, it can capture very different scenarios ranging from centralized systems (such as central servers) to massively distributed systems, with peers ranging from fully trusted to totally untrusted, providing encrypted or clear information. Furthermore, it can capture scenarios combining the previous cases, which are the reality of today's Web, in arbitrarily rich ways.

Formal model Because the model is formally defined, we can prove (or disprove) desirable properties for a system described with our model, such as soundness (data is only acquired legitimately) and completeness (all legitimate data may be acquired). This is in the spirit of [Aba09] that uses logic to describe access control protocols. Also, peers may perform automatic reasoning using the knowledge base to obtain information on data, localization and access control.

Quality control Because our model addresses provenance and time, we can better control the quality of data. This is in-line with recent works on data provenance, e.g., [BT07]. We view time and provenance not as gadgets but as essential components of a solution for properly supporting access control in a distributed setting, in particular for detecting who is responsible for misuses of the system.

Overall, the thesis is that with *all* the information managed in a distributed knowledge base, and with reasoning, we can automate the management of the distributed information. In particular, we describe in details the management of access control and distribution, introducing special kind of knowledge and the corresponding rules.

Access control For access control, we consider three kinds of knowledge statements, namely *access right*, *secret*, and *hint*. Using these statements, we show how to describe an access control mechanism based on asymmetric cryptographic keys and one based on Web HTTP access, controlled by login/password. We show in particular how to exchange information between peers that are trusted or untrusted, in clear or encrypted.

Distribution For distribution, we use *localization* knowledge statements. Using these statements, we show how to manage data in different scenarios of data distribution, from centralized servers such as Facebook, to P2P communities organized around DHT or gossiping, possibly with replication.

Organization

The chapter is organized as follows. In Section 5.1, we present the general *WebdamExchange* model. Section 5.2 deals with access control, and Section 5.3 with distribution. In Section 5.4, we propose some general policies. We conclude in Section 5.5 with perspectives on future work.

5.1 The general model

5.1.1 Informal presentation

We consider autonomous peers exchanging data using messages. For this, we build on the distributed datalog language, *WebdamLog*, discussed in the previous chapter. We introduce two extensions of *WebdamLog* that are essential for *WebdamExchange*, namely virtual principals and semi-structured data.

Virtual principal

The *WebdamLog* language is tailored to physical peers such as Alice's iPhone or Bob's laptop, capturing data exchange between them. In *WebdamExchange*, we call *principal* an entity that owns data and has access right delegations on the data of other principals. A peer is such a *physical* principal. *WebdamExchange* also supports *virtual* principals, e.g., a user such as Alice, or a group such as *Roc14*. Contrary to a peer, a virtual principal has no storage or processing resources, relying on peers for that. The notion of virtual principal is primarily used to specify and manage access rights. Essential issues are who stores the data of a virtual principal, and who has read/write access to them. Typically, physical principals will store and process data for virtual principals. They may also temporarily create *avatars* of virtual principals. For instance, an avatar of Alice is created on her iPhone when she wants to access and update data from this device.

The idea of the extension is as follows. Besides facts of the form $r@p(u_1,...,u_n)$ where p is a peer, we have facts $r@q(u_1,...,u_n)$ where q is a

virtual principal. Such a fact is stored on a physical peer p as an external fact, i.e.,

says@p(r,q,u_1,...,u_n).

where *says* is a reserved relation name. When a relation about a virtual principal is used in a rule, the peer "resolves" it (using rules) to replace it by the external relations.

Note that the basic *WebdamLog* model is strongly typed. On the other hand, an external relation needs to store tuples of arbitrary arity. We next turn the model into a semi-structured data model, which fixes this typing issue.

Semi-structured data model

Another problem we have to face on the Web is that the data is naturally semi-structured. For instance, a climbing site may be recorded in *Roc14* as follows, using the standard syntax of JSON:

```
climbingSite@Roc14:{
   "id":"&cuvier",
   "Name":"Cuvier",
   "ClimbingSiteType":"Bouldering area",
   "Circuit": [{"idref":"circuit@Roc14&cuvier-orange"},
               {"idref":"circuit@Roc14&cuvier-blue"},
               {"idref":"circuit@Roc14&cuvier-red"}]
   "GoogleMapsURL":"..." }
```

This fact is representing the Cuvier bouldering area that includes 3 circuits, orange, blue and red. The number of circuits may depend on the size of the bouldering area. Brackets denote collections and ampersands denote references. Note that &*cuvier-orange* identifies a document within the domain of the *circuit* relation of *Roc14*.

To simplify data management, we extend *WebdamLog* with the notions of a *document* and a *collection*. A *document* is a coherent, self-contained piece of data, modeled by an XML-like tree. For example, we may have the following document `picture1@Bob` containing a jpeg picture, using the JSON syntax:

```
picture1@Bob = {"Name":"picture1", "Type":"jpeg", "byteStream":"..."}
```

A document corresponds to a relation that contains the different versions of the document at different points in time. A document update therefore corresponds to adding a new version to the relation.

A *collection* consists of a map of data values. Collections are also used for access control and localization (specifically, we employ collections of *access control* and *where statements* respectively, which we will define later). The most basic collection is a collection of document references, which is a list of references to documents. Collections are updated by adding or removing data values. For example,

pictures@Roc14 **+=** picture1@Bob

adds a reference to the previous document to the collection of pictures of Roc14.

The standard request to a document or a collection is to ask for its "current" version, i.e., the last version of the document or the consolidated list of references of the collection. These notions raise the issue of consistency that is particularly critical for collections of access rights. For instance, it is important to decide whether a peer requested an update to a document "before" or "after" it obtains the right to perform such an update. Distribution greatly complicates the issue. We will discuss more precisely this issue in Section 5.1.2.

Statements, instructions and external knowledge

In *WebdamLog*, there is no difference between a fact and a message. More precisely, if a peer p derives a fact $r@p(u_1,...,u_n)$, this is a fact to store; if p derives $r@q(u_1,...,u_n)$, for $q \neq p$, this is a message to send to q. The message is automatically accepted by q as a new fact. However, in our current setting, we want to allow the peer to process the fact based on its own logic. In this fashion, q sees the message as a request to insert the fact, but it may decide to not actually insert it in its local knowledge base.

We next detail this important distinction between an insertion instruction and the (actual logical) statement that may result from it. The following *statement* may for instance be installed by Alice-iPhone:

Alice-iPhone ***states*** climbingSite@Roc14={"id":"&cuvier",...}
requester **Alice**;

In other words, Alice-iPhone created a fact of the relation climbingSite@Roc14 (and typically stored it in its database). It is important to understand who the participants in such a fact are: Alice-iPhone *performed* the statement;

Roc14 *owns* this piece of data; Alice *requested* this update. This last information is used to trace the provenance of the fact. Such a complicated model is necessary because it is very typical on the Web to have a principal (Alice) who has the right to state a fact of another principal (Roc14) but has to rely on another principal (Alice's iPhone) to perform this task.

How did we get there? Typically, Alice made the following instruction:

Alice *requests* climbingSite@Roc14 = {"id":"&cuvier", ...}
to Alice-iPhone;

to her iPhone. Another example of an instruction is

Bob *requests* get climbingSite@Roc14&cuvier **to Alice-iPhone**;

which is an instruction of Bob to Alice's iPhone for some data. If Alice's iPhone can prove that Bob is entitled to have this data, it can send it to him. So, a peer may want to exchange statements and messages previously received from other peers. For this, we introduce another class of messages, external knowledge, that can contain statements, instructions or external knowledge. For example, Alice-iPhone may answer Bob's instruction using:

Alice-iPhone *says* Alice-iPhone *states*
climbingSite@Roc14= {"id":"&cuvier", ...}
requester Alice **to Bob**;

It means that Alice-iPhone sent this statement to Bob. We say that Alice-iPhone *performed* the communication and that Bob *received* it. One may also want to exchange rules. For example:

Alice-iPhone *says* Alice-iPhone *states*
climbingSite@Roc14:{...} :- climbingSite@Alice:{...}
requester Alice **to Bob-laptop**;

means that Alice-iPhone installed in the laptop of Bob a view machinery to copy Alice's data to Roc14.

The model allows capturing provenance information. Indeed, each message may contain information about the principals who sends it and who receives it, and the message is authenticated accordingly. When a message is transmitted from one peer to another, this information is pilled up, forming a chain of external knowledge of the following form:

P_n *says* ... (P_2 *says* (P_1 *says* (P_1 *states* ...) to P_2) to P_3) ... to P_{n+1}

If we want to trace the full communication of the system, the peers must only accept instructions and well-formed chains of external knowledge.

We should also mention that timestamps are also attached to messages. Indeed, we have shown previously that timestamps have an important impact on the expressive power of *WebdamLog*. We timestamp knowledge as follows:

Alice-iPhone *states* climbingSite@Roc14={"id":"&cuvier",...}
requester Alice ***at*** **06/12/2011 14:00:00**

In *WebdamExchange*, the timestamps are local to the performer, here Alice-iPhone. There is no global clock a priori.

5.1.2 Formal model

Alphabets and Schemas

We assume the existence of two infinite disjoint alphabets of sorted *constants*: *principal* and *relation*. We also consider the alphabet of *data* that includes in addition to *principal* and *relation* infinitely many other constants of different sorts (*integer*, *string*, *bitstream*,...). Similarly, we have the corresponding alphabets of sorted *variables*, implicitly denoted by their first symbol $.

A *schema* is an expression ($\mathcal{K}$, Π, Π' $\mathcal{E}$, $\mathcal{I}$, σ) where $\mathcal{K}$ is the set of *keywords*, that are special constants of the model; Π is a (possibly infinite) set of physical principal constants and Π' is a (possibly infinite) set of virtual principal such as $\Pi \cap \Pi' = \varnothing$; $\mathcal{E}$ and $\mathcal{I}$ are disjoint sets, respectively, of *extensional* and *intentional* names of the form *m@p* for some *relation* name *m* and some principal *p*; and the typing function σ defines for each *m@p* in $\mathcal{E} \cup \mathcal{I}$ the sort of its content. We do not describe in details the typing language here, but any kind of typing of semi-structured data, such as DTD or XML-Schema, could be used. We use a JSON syntax in the following. The system uses an equivalent XML syntax.

Statements

A (JSON) *term* is defined by the following regular expression:

term ::= label:value | {}
label ::= constant | $variable
value ::= constant | $variable | {term (,term)*} | label:[{term} (,{term})*]

Given a relation *rel@owner* and a physical principal *performer*, a ground relation fact, or *rel-statement* is an expression of the form:

performer *states* rel@owner:{term}

where *term* is a term without variables, of the proper type according to $\sigma(rel@owner)$. The rel-statements are not persistent by defaults and are consumed as soon as they are read. Acknowledgements are particular kinds of relation statements of the form:

performer *states* ack:{value:bool,type:ackType,content:{term}}

where *bool* is true or false, depending of the success of an operation, *ackType* is in the set of acknowledgement type (a subset of $\mathcal{K}$) (Communication, Security, Storage ...) and *term* is a term. Here, *owner* is implicitly equals to *performer*.

Given a document *doc@owner*, a physical principal *performer* and a principal *req*, a ground document fact, or *doc-statement* is an expression of the form:

performer *states* doc@owner={term} *requester* req *at* time

where *term* is a term without variables, of the proper type according to $\sigma(doc@owner)$ and *time* is the current time of the performer when it created the statement. A document is implicitly persistent and there is implicitly only one valid version of a document. This statement means that the valid version of the document *doc@owner* was set to *doc@owner:term* when it was performed.

Given a collection *coll@owner*, a physical principal *performer* and a principal *req*, a ground collection fact, or *coll-statement* is an expression of one of the form:

performer *states* coll@owner:{keyTerm}+={valueTerm} *requester* req *at* time
performer *states* coll@owner:{keyTerm}-={valueTerm} *requester* req *at* time

where *keyTerm* and *valueTerm* are terms without variables, of the proper type according to $\sigma(coll@owner)$ and *time* is the current time of the performer when it created the statement. A collection is implicitly persistent. The statements means that *valueTerm* has been added or removed as a value of the collection for the key *keyTerm*. For a given set of coll-statement $\mathcal{C}$ about the same collection *coll@owner*, one can compute the corresponding collection by

1. removing from $\mathcal{C}$ the coll-statement *performer states coll@owner:{keyTerm} += {valueTerm} requester req at t1* if the coll-statement *performer states coll@owner: {keyTerm}-= {valueTerm} requester req at t2* also appears in $\mathcal{C}$

2. for the remaining positive coll-statement, grouping them by their key terms *keyTerm1*, *keyTerm2* and build:

```
coll@owner:{
    keyTerm1:[valueTerm1-1,valueTerm1-2,...],
    keyTerm2:[valueTerm2-1,valueTerm2-2,...],
...}
```

In its general form, a collection can in particular capture an index, that maps a key (e.g., a keyword or a tag) to a set of references. We will also see that it can be used to model access control list and localization. One of the simplest way of using a collection is as a list of document references. The key is the empty term {} and the values are document references. It corresponds to the previous example of the pictures collection of Bob.

Remark 5.1. The documents and collections are presented here as a primitive concept of the model. In fact, they can be represented using rel-statement, and the specific way to manage them can be simulated using rules, as will be discussed later. They are so important that we use some specific notation to distinguish it.

Instructions and External Knowledge

Given a document *doc@owner* and two physical principals *performer* and *target*, a *doc-instruction* is an expression of the form:

performer *requests* doc@owner=term to target

Given a collection *coll@owner* and two principals *performer* and *target*, a *coll-instruction* is an expression of one of the form:

performer *requests* coll@owner:{keyTerm}+={valueTerm} to target
performer *requests* coll@owner:{keyTerm}-={valueTerm} to target

These instructions mean that *performer* asks *target* to create the corresponding statement on its behalf. If *target* chooses to execute the instruction and create the statement, we say it *factifies* the statement.

Finally, given a document *doc@owner* or a collection *coll@owner* and a key term *keyTerm* and two principals *performer* and *target*, a *get-instruction* is an expression of the form:

performer *requests* get doc@owner to target
performer *requests* get coll@owner to target
performer *requests* get coll@owner:{keyTerm} to target

Instructions are particular forms of knowledge communicated from a principal, the performer, to another principal, the target. It is usually added to the database of the target. Nevertheless, there is a main difference with *WebdamLog*: in *WebdamExchange*, one only exchanges special kind of facts: local data has to be nested in special facts to be sent. Given two physical principals *performer* and *target*, an *external knowledge* is an expression of the form:

performer *says* fact to target

where *fact* is a local statement, or an instruction or an external knowledge from another peer stored by the performer. An external knowledge of that form is *well-formed* if *performer* is the performer of *fact* for a nested statement or the target of *fact* for a nested external knowledge or instruction. We say that a chain of external knowledge is *well-formed* if all the external knowledge of the chain are well-formed.

Rules

A *fact pattern* is a statement or an external knowledge with terms that may contain variables or an expression *$f//pattern* where $\$f$ is a variable representing a statement or an external knowledge and *pattern* is an internal pattern of this fact. It corresponds to the descendant operator of classical tree languages. A (*WebdamExchange*) *relation rule* is an expression of the form

rel@p:{term} :- $(\neg)$ fact$_1$, ..., $(\neg)$ fact$_n$

where *term* is a JSON term and each $fact_i$ is a fact pattern. We also allow in the body of the rules atoms of the form $X = Y$ or $X \neq Y$ where X and Y are constants or variables. We require a rule to be *safe*, i.e.,

1. For each statement $fact_i$, if the performer is a variable, it has to be previously bound.
2. For each external knowledge $fact_i$, if the target is a variable, it has to be previously bound.
3. Each variable occurring in a literal $(\neg)fact_i$ must be previously bound in a positive literal.
4. Each variable in the head must be positively bound in the body

A *relation-rule statement* is an expression of the form:

performer *states* rule

where *performer* is a physical principal and *rule* is a relation rule.

The relation rules are extended to document and collection rules as follow:

(doc@p={term} *requester* req *at* t) :-
    ($\neg$) $\text{fact}_1, \ldots, (\neg)\ \text{fact}_n$
(coll@p:{keyTerm}+={valueTerm} *requester* req *at* t) :-
    ($\neg$) $\text{fact}_1, \ldots, (\neg)\ \text{fact}_n$
(coll@p:{keyTerm}-={valueTerm} *requester* req *at* t) :-
    ($\neg$) $\text{fact}_1, \ldots, (\neg)\ \text{fact}_n$

A (*WebdamExchange*) *general rule* is an expression of the form

fact_{n+1} :- ($\neg$) $\text{fact}_1, \ldots, (\neg)\ \text{fact}_n$

where each $fact_i$ is a fact pattern. We also allow in the body of the rules atoms of the form $X = Y$ or $X \neq Y$ where X and Y are constants or variables. We require a rule to be *safe*.

A (*WebdamExchange*) *external rule* is an expression of the form

performer *says* rule to req

where *performer* and *req* are two principals and *rule* is a general rule. The semantics of the facts and rules are explained in the next section, through a reduction to *WebdamLog*.

Interpretation of *WebdamExchange* in *WebdamLog*

From the notations, one may think that the link with *WebdamLog* is through relation names. In fact, the link is more subtle, due to the virtual principals. The extensional facts of the *WebdamLog* model are restricted to *states*, *requests* and *says*. A statement *performer states ...* is interpreted as *states@performer(...)*, and an external knowledge *performer says ... to target* or *performer requests ... to target* as *says@target(performer, ...)* and *requests@target(performer, ...)* respectively. Recall that we ask $perf ormer$ and *target* to be physical principal. It is now clear why it is important. In fact, the performer can be a peer, a wrapper or an avatar. A target is in general a peer. It can also be a wrapper or an avatar as long as the peer creating the external knowledge can directly contact it. If it is not the case, then we suppose that the external knowledge is lost.

The interpretation of document and collection statements depends of the consistency model. It is a subtle problem due to distribution. For

example, Bob-laptop may perform an update on the data of Alice without Alice-iPhone being aware of the change. Moreover, Alice-iPhone may create a different update on its own. Then, when the update of Bob-laptop and Alice-iPhone are received by a peer (e.g., Alice-iPhone), the peer needs to be able to resolve the conflict. This is a serious issue when access control are updated. In a distributed setting, it is important that the resolution of the conflict itself is consistent i.e., two different parts of the system resolve the same conflict in the same way. So, obtaining the "current" doc-statement or set of coll-statements is an issue. The distributed systems literature provides a host of techniques for different consistency models. Any well-founded consistency model can usually be expressed with our rules. Depending of the choice of the consistency model, one will choose the rules to use for the simulation.

Let us illustrate with an example. Suppose that we have a predicate *after@p(fact1,fact2)* that provides a total order in the revision history (*fact1* happens after *fact2*) and that does not depends on the peer p where it is called. For instance, supposing that the clocks of the peer are perfectly synchronized and infinitely precise, this predicate enforces that the timestamp of fact1 is older than the one of fact2. One can similarly use a global timestamping or revision identifier management, which can be centralized or distributed. The document are interpreted using rules of the form:

```
(at local)
  persistent db@local
  db@local(docId:$doc, owner:$p, content:$term, fact:$f) :-
       $f[@local]//_ states $doc@$p={$term} requester _ at _
  del.db@local(docId:$doc, owner:$p, content:$term, fact:$f) :-
       db@local(docId:$doc, owner:$p, content:$term, fact:$f),
       db@local(docId:$doc, owner:$p, content:_, fact:$f2),
       after@local($f, $f2)
```

where _ are unamed variables and *$f[@local]//_ states $doc@$p={$term} requester _ at _* means that there is a local fact $\$f$ (a statement performed by *local* or an external knowledge or instruction whose target is *local*) which contains this statement. For example, the fact $\$f$ could be

states@local(docId:doc, owner:p, content:term, requester:req, time:t)

or

says@local(perf:perf, fact:{perf *states* doc@p=term *requester* req *at* time})

The collection are similarly interpreted using the following rules:

(at local)
persistent db@local
db@local(collId:$coll, owner:$p, key:$keyTerm, value:$valueTerm, fact:$f) :-
$f[@local]//_ *states* coll@p{keyTerm}+={valueTerm} *requester* _ *at* _
del.db@local(collId:$coll, owner:$p, key:$keyTerm, value:$valueTerm, fact:$f) :-
db@local(collId:$coll, owner:$p, key:$keyTerm, value:$valueTerm, fact:$f)
$f2[@local]//_ *states* coll@p{keyTerm}-={valueTerm} *requester* _ *at* _
after@local($f2,$f)

Interpretation of rules

Then the interpretation of *WebdamExchange* rules in *WebdamLog* is relatively straightforward in absence of access control. The relation-rule statement

performer *states* rel@p:{term} :- ($\neg$) fact$_1$, ..., ($\neg$) fact$_n$

is interpreted as the *WebdamLog* rule:

states@performer(rel@p:{term}) :-
($\neg$) trans-fact$_1$, ..., ($\neg$) trans-fact$_n$

where each *trans-fact*$_i$ is the transformation of $fact_i$ in *WebdamLog*. A relation-rule statement inside an external knowledge is interpreted by extracting the relation-rule and replacing the performer by the current peer. This interpretation is naturally extended to document and collection rules. The general rules are directly interpreted as *WebdamLog* rules by converting the facts themselves and installation of an external rule is a direct application of that interpretation.

Programs and Policies

The behavior of a peer is governed by a *program*, i.e., a set of local or delegated rules. The program is in charge of managing the knowledge base. These rules deal with statements and high level tasks such as localize information, obtain it, verify it, etc, specified using instructions. Note that some rules are specific to the particular peer the program is running on. For instance, a powerful laptop and a smartphone will typically have different rules for specifying caching, because they do not have the same storage capacity or network bandwidth. The *state* of a peer consists of a set of consistent statements, its personal knowledge and the facts it received from other peers and did not process yet. Note that the personal knowledge of

a peer typically includes statements about other principals. A *move* of the peer's program consists in:

- inserting/deleting facts in its knowledge base;
- sending facts to other peers;
- receiving asynchronously facts that had been inferred by other peers.

Given a principal, the *policy* of the principal is the sub-part of the programs of all peers that has an effect in the management of the knowledge of this principal. For instance, recall the example in the introduction where Alice wants to search blogs on different machines. The task is specified as an instruction, that activates some high level rules. Based on the policies of the different friends, the execution of the high level rules leads to the activation of different policy rules for the different blogs. Note that the discovery of new principals may naturally lead to discovering new policies. The corresponding rules have to be loaded before the task is performed. For instance, if Lila uses some exotic system she developed with some experimental access control mechanism and Alice adds Lila to her list of friends in Alice-laptop, the query results in integrating the rules of Lila's access control policy to the program of Alice-laptop peer.

To support the claim that the *WebdamExchange* approach allows handling a wide variety of situations encountered on the Web, we consider in more detail access control and distribution. In both cases, we introduce generic statements and briefly discuss how they may be supported in different Web contexts. Moreover, installation of an external rule, or of a relation-rule statement inside an external knowledge, is not a simple task in the context of access control. Indeed, one wants to enforce access control policies in the rules. So we will discuss the interpretation of *WebdamExchange* with access control at the end of the next section.

5.2 Access control

5.2.1 Informal presentation

In *WebdamExchange*, the access control granularity is at the level of the principal. A principal gets some particular access right to all the data of another principal. For example, Alice may be in the *own* access control list of the Roc14 group. She may then delegate this right to Alice-iPhone and Alice-laptop. In general, the access control policies are not able to prevent

malicious principals to misbehave. But if enough information is recorded (notably provenance), the system is able to identify malicious peers.

Logical representation

In a standard way, access control in *WebdamExchange* is based on access control lists and credentials. For example, the following statement specifies that Alice-iPhone delegates the *write* right to Bob on Roc14:

Alice-iPhone *states* writer@Roc14+=Bob *requester* Alice *at* t

We also use the notion of *credentials*, which consist of two abstract dual notions, namely *hint* and *secret*. Concretely, hints and secrets depend on the particular means of enforcing access control. For example, in the case of RSA cryptography, the hint is an RSA public key and the secret the corresponding RSA private key. Based on these abstract notions, the model is general enough to match a wide range of scenarios found on the Web. Alice-iPhone may specify the creation of a pair of hint and secret for *write* right of Roc14 as follows:

Alice-iPhone *states* writeHint@Roc14{} *requester* Alice *at* t
Alice-iPhone *states* writeSecret@Roc14{} *requester* Alice *at* t

where *writeHint* and *writeSecret* are keywords of the
language. A principal who can create statements to enforce access policies (access control and credentials) essentially has delegation power. We choose to authorize this action with the strongest access right of our model, the *own* right.

The hint ans secret are mostly use to provide an *authentication* mechanism. For example, in the case of RSA cryptography, Alice-iPhone signs the previous statements with the owner secret of Roc14, to prove that she indeed has the right to delegate access on Roc14. Bob may check the signature with the corresponding hint. The hint and secret may also be used to provide a *protection* mechanism. For example, in the case of RSA cryptography, Alice-iPhone may encrypt the content of a document of Roc14 with the *read* hint of Roc14, if she does not trust intermediate peers. Then only principals with the *read* secret of Roc14 are able to read the document content. Since protection is not mandatory, we use a special logical form to make it explicit. For instance,

climbingSite@Roc14={...} ***encrypted*** **for reader@Roc14**

means that the content of the document *climbingSite* has been encrypted with the *read* hint of Roc14.

Physical interpretation

The logical form abstracts the physical interpretation of the credentials. For example, one may use RSA cryptography, or login and password. Given a set of algebraic properties on the physical interpretation (the most important one being symmetry/asymmetry), one may define policies without going into the details of the physical interpretation. It is important to keep in mind that the model is designed to support most of the security protocols. The model is particularly well suited for RSA cryptography, but we also consider symmetric cryptography and login/password authentication.

5.2.2 Formal model

Identity

The principals of the standard *WebdamExchange* model already have a globally unique identity. It is of most importance to enforce that this identity is unique and unforgeable on the context of access control. It means that the system always distinguishes between two different principals, and in particular that one is not able to create a principal which looks like an existing one. We will explain how to enforce this property when discussing physical interpretation.

Access rights

The extension of the standard *WebdamExchange* model is as follow. We introduce a special list of keywords for access rights: *own*, *write*, *append*, *remove*, *read*, *readAcc*, *readWhere*, *writeWhere*. They are organized following the hierarchy of Figure 5.1. The *own* right is used for authentication of performers and granting/revoking access rights. The *own* right also transmits the access rights of a principal: if p is owner of q and q has a right r on q', then p has also the right r on q'. The *write* right grants the right to edit basic documents and collections. The *append* and *remove* rights grant the right to append and remove elements of a basic collection respectively. The *read* right grants the right to read basic documents and collections. The *readAcc* right grants the right to read the list of access control. The *readWhere* and *writeWhere* rights will be introduced in the section on distribution.

We also add a special keyword for principals, *allPrincipal*, which can be used for granting a right to all principals.

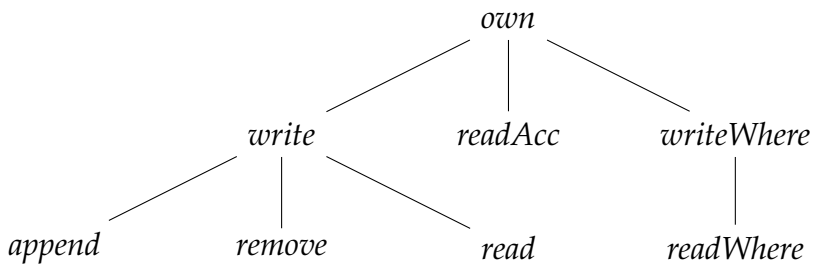

Figure 5.1: The hierarchy of rights in *WebdamExchange*

Statements for access control

Given a right keyword *right*, a physical principal *performer* and three principals *owner*, *req* and *principal*, an access control list ground fact, or *acc-statement* is an expression of one of the form:

performer *states right*er@owner+=principal *requester* req *at* time
performer *states right*er@owner-=principal *requester* req *at* time

The statements mean that the principal *principal* has been granted or revoked the *right* access right on the principal *owner*. The acc-statements are clearly special kinds of coll-statements. In particular, they are implicitly persistent and the corresponding instructions are straightforward. The main difference is that one need the *own* right to perform them and the *readAcc* right to read them.

Given a right keyword *right*, a physical principal *performer* and two principals *owner* and *req*, hint and secret ground facts, or *hint-statement* and *secret-statement* are expressions of the following forms respectively:

performer *states right*Hint@owner *requester* req *at* time
performer *states right*Secret@owner *requester* req *at* time

The statements mean that a new fresh value has been set for the hint or the secret of *owner*. The hint-statement and secret-statement are clearly special kinds of doc-statements. As for acc-statement, they are implicitly persistent, the corresponding instructions are straightforward and one needs the *own* right to perform them. The hint statements can be read by every principals. The secret statements can be read by the owners of principals who have the corresponding access right.

Protection

Finally, given a right keyword *right*, a principal *principal* and a data *rel@owner:{term}*, the *protection* (or encryption) of the data is an expression of the form:

rel@owner:({term} *encrypted* for *right*er@principal)

It means that the content of the relation (the term *term*) has been encrypted such that one needs the *right* secret of *principal* to decrypt it. If the credentials are asymmetric, one needs only the corresponding hint to perform the encryption. If the credentials are symmetric, then one needs the secret to perform the encryption. Usually, one uses the secret corresponding to the right needed to read the statement. For instance, it would be the *read* secret for a doc-statement. For a secret-statement, it would be the *own* secret of a principal who has the corresponding access right.

Interpretation of statements

The presence of access control clearly changes the basic interpretation of *WebdamExchange*. Indeed, one cannot let anyone create a fact or install a rule. We provide a basic interpretation model. Of course, more refined ones can be easily developed from that model.

Authentication is at the basis of credential-based access control. So the knowledge of *WebdamExchange* with access control is authenticated. In particular, checking a statement or an external knowledge is a local operation that is systematically executed before accepting a fact. A ground statement is *valid* if it is authenticated with a secret whose owner is the owner of the statement and whose right is higher or equals to the right needed to create this statement. The secret is checked with respect to the local database of the peer doing the verification. If the secret is asymmetric, one uses the hint to validate the secret. If the secret is symmetric, then the peer needs the secret for the validation. For instance, the statement

Alice-iPhone *states* climbingSite@Roc14={“id”=“&cuvier”,...}
requester Alice *at* t;

is authenticated with the *write* secret of Roc14. It can be checked using the *write* hint (or secret if there is no hint) of Roc14. Note that secrets and hints may change (concurrently). The process fails if one uses some secret with a different version of the hint, which highlights consistency requirements.

Authentication is also used to verify provenance for instruction and external knowledge. These messages are authenticated with the *own* secret of their performer. The receiver can check the identity of the performer

and later prove that it indeed received the message. An instruction is *valid* if it is authenticated with the *own* secret of its performer. An external knowledge is *valid* if it is well-formed, if its internal fact is valid and if it is authenticated with the *own* secret of its performer. A physical principal p only accepts valid external knowledge or valid instructions from which it is the target. If it receives a fact f that is unvalid, it sends the following fact to the performer *perf* of the external knowledge:

p *says* p *states* ack:{value:false,type:Security,content:f} to perf

Of course, it signs it with its *own* secret.

If the instruction is valid, the peer decides if it executes it or not using rules which checks that the performer of the instruction has indeed the right to execute it. For example, with a get instruction, we may use a rule of the form:

(at local)
local *says* $f to $client :-
_[@local]//$client *requests* get $docId@$owner to _
[@local]// *states* reader@$owner += $client *requester* _ *at* _
$f[@local]//_ *states* $docId@$owner=_ *requester* _ *at* _

where *local* is the name of the peer executing the rule and [@local] means that the data is searched in the local database of this peer. The meaning of the rule is as follows. If the (local) peer receives an instruction for a particular document $docId@$owner (that may be nested in a fact), the peer checks that the client has the *read* right (looking for the corresponding statement, that may be nested in a fact). Then, if this is the case, it finds the statement (that may be nested in fact $f) and sends back $f (using the *says* literal). One may also use intentional rules to check the access right and obtain the data, depending on the policy of the principal *$owner*. The peer also stores the instruction, so that it can later prove that the requester indeed requested the data.

Remark 5.2 (Role of the different principals)**.** The notions of *owner*, *performer*, *target* and *requester* should now be clear in our model. The notions of owner is used to check that the update was valid. The notions of performer and target are used to check the validity of a provenance chain. The notion of performer of an instruction is also used to decide whether an update should be performed or not. Finally, the requester is used to link two chains of provenance without having to carry the whole history. A performer

must be able to provide the valid chains of provenance of data used to perform a statement, in particular the corresponding instruction.

Interpretation of rules

The treatment of rules is somewhat more complex. In fact, deciding wether a rule has to be accepted by a peer or not depends at least of the fact that is produced and of the facts that are read. Globally, one expects the principal that created the rule, to have the right to perform the produced fact and to read the fact in the body. Indeed, if a principal can read the body and perform the fact, it may do the operation locally, even if it may be less efficient. So it is not gaining any access right by installing the rule. How these properties are checked may depend on the policy. Not checking them may lead to security leaks. The problem is that in general, one may ignore which kinds of facts are in the rule and who are their owner. The basic interpretation is the following. We suppose that we only allow document and collection rules. So any rule will be of the kind:

perf *states* rel@p:{term} :- ($\neg$) fact_1, ..., ($\neg$) fact_n

Then it is easy to check if the performer of the rule statement has the right to perform the relation, using the same authentication scheme than for ground statements. Next, the peer can easily check that the performer of the statement rule has read access to the content of fact_i when this one contains a nested statement that has its owner instancied. If a fact contains a statement whose relation owner *$owner* is not instancied, then the peer modifies the rule by adding at the end a pattern of the kind:

_//_ *states* writer@$owner += perf *requester* _ *at* _

where _ denotes an unamed variable , *$owner* is the variable corresponding to the variable in the owner variable of the fact and *perf* is the performer of the rule statement.

The previous interpretation guarantees that the rules that are accepted respect access control specification. But one may want to restrict further the delegated rules that are accepted by a peer, for example to avoid being overloaded by undesired tasks. One may of course ask an administrator to authorize each rule installation, but it is far from the dynamicity we have in mind. So we may also consider pattern-based rules acceptance: the administrator of the peer defines positive or negative rules patterns, and the rules are filtered accordingly. It is very similar to the usage of view to enforce access control in database. We believe this is a most interesting direction of research, that is left for future work.

5.2.3 Properties

A wide variety of scenarios may be considered to share information in a distributed setting. Given a scenario, a global policy specifies how real participants (users or peers) behave and how the information corresponding to each participant (real or virtual) is stored and found. In this section, we are interested in controlling that the data is correctly used in any given system. In particular, we introduce desirable properties systems should guarantee, namely *completeness*, *consistency*, and *soundness*. We also discuss how control can be performed to detect illegal behaviors.

Well-formed

We assume that the systems we study manage only *well-formed* knowledge. This means that only facts legally authenticated (with time and provenance) are considered, which can be verified by checking signatures. Strictly speaking, a server may record some non well-formed fact. This may be the case because the fact is encrypted and the server does not have the key to decrypt and verify. However, such a pattern is eventually detected before the fact is used when someone with access rights decrypts it. Since non well-formed knowledge is never used and is eventually discarded, we may as well consider without loss of generality that it is not present.

Soundness

Since we are primarily interested in systems where principals may store information for others and serve it, we focus here on *data-privacy* that states that a principal can read (in clear) only the contents of documents it has access to according to access rights. Note that one could also consider stronger forms of privacy that would require that one has limited access to access rights knowledge (*right-privacy*) or that one does not even know of the existence of a document that one is not entitled to read (*docId-privacy*). We define a system as (data-privacy) *sound* if, when used legally, it guarantees data-privacy. Note that this implies in particular that a principal can not acquire a key it should not hold.

To guarantee soundness, we can use the following general rule:

Definition 5.3 (*sound-rule*)**.** A peer is allowed to perform a statement only if it has a proof that the requester has the right to request that statement. When sending knowledge to another principal, the content of the statements is encrypted with the read access-right secret of its owner, unless the sender has a proof that the recipient has the right to access the information.

A system is *monotone* if it only allows adding knowledge, i.e., it disallows statements that would revise previous knowledge. In particular, documents cannot be updated, access rights cannot be revoked and items can be appended to collections but not removed. We next state that monotone systems respecting *sound-rule* are sound.

Theorem 5.4. *When all principals in a monotone well-formed system respect* sound-rule, *the system is guaranteed to be sound.*

The proof of the result is by induction and uses the fact that when all principals in a monotone well-formed system respect the *sound-rule*, no illegal update may occur. One can in fact show a stronger result, namely, that if some principals do not obey the rule, (i) their coalition will not gain more information than the union of the information they legally have access to and (ii) as soon as they distribute illegal information outside their coalition, it is possible to detect their misbehavior. Of course, one may want to consider non-monotone systems. The issue is then to be sure that one gets all relevant information including information that may invalidate previous access rights. This leads to considering other desirable properties of such systems, namely completeness and consistency.

Completeness and consistency

The guarantee to have access to *all* the relevant knowledge is essential. For instance, a peer serving the documents for a principal p needs to know the list of all principals that can read p and for that be aware of all read access revocations.

We say that a system is *complete* if each principal can realize an instruction (read or update) it is entitled to and in case of a read instruction, get the complete answer. This is clearly a nontrivial property to guarantee in a distributed setting. It includes a number of facets.

awareness First, a principal p should be able to find which principal q owns information, p has some right on.

reachability Then p has to be able to find where and how to send a particular query/update instruction concerning q. Note that reachability takes a particular flavor when granting some access to all principals. The system should provide some form of global query facility to actually find information, in the style of querying public data in Facebook.

denial-free Also, a peer receiving a legal instruction within its competency should be willing to serve. More precisely, a peer refusal to serve (because of failure or out of malevolence) should not prevent from actually realizing a read instruction (*read-denial-free*) or an update instruction (*update-denial-free*). To guarantee the denial-free properties, we can rely on trusted servers. Otherwise, if peers may potentially be unreachable or possibly malevolent, we have to rely on replication and on guarantees that is probabilistic in nature (based on the probability of error and illegal behavior).

consistency Finally, if the system uses replication, different principals serving the same information should be consistent in the sense of distributed databases. For a given query, at a given time, one should get the same answer everywhere. There may be several writers and replicas for the same information. To perform an update with a guarantee of consistency between the replicas, one has to update them all in an atomic operation, a very hard requirement in a distributed setting. In particular, one has to guarantee that the corresponding access right will not be updated during the operation. To support consistency control, one could rely on concurrency control mechanisms provided by participating systems. One could also develop light (but provably correct) concurrency control solutions adapted to particular applications. One could also not care (often the solution adopted on the Web) and rely on ad hoc conflict resolutions. Observe that this would possibly generate delays in the propagation of updates (a rather imprecise notion since there is no universal clock in the system). The management of concurrency control in our setting is a challenging and interesting research directions.

Verifying legality

We discussed properties of systems when all participants behave legally, e.g., obey the *sound-rule*. But in most applications, in particular social networks, it is very likely that some principals will not behave legally, out of malevolence or errors. For instance, Betty may transmit some document d of some principal Alice to someone who does not have read access to it, a violation of the *sound-rule*. If this remains within a a group of misbehaving users who are collaborating, nothing can be done about it. Now Betty may send d to George, a honest user, who does not have read access to Alice. She may also send him the *write* key of Alice, allowing George to update Alice data illegally, but in good faith. Indeed, even if Betty is the owner

of Alice, she has to add George as a reader/writer before sending data or keys to him. We would like to detect this kind of illegal behaviors as soon as some illegal knowledge reaches an honest peer.

We briefly discuss two issues related to (i) who verifies and (ii) how. First consider who performs the verification. One may require that anyone can verify whether some data is transmitted to someone legally or whether some credential was used legally. This clearly imposes that access rights are fully disclosed. One can alternatively rely on some authority to perform the verification. This authority should know all the access rights and all peers should report all their activity to it. This notion of authority (even if supported in a distributed manner) is very far from a social network spirit. One may impose that the exchange of any information about principal p be reported to $p_{trusted}$ (an "avatar" of p or a system that p trusts for verification). Lastly, one can rely on each principal to verify how information in this principal is handled.

We now consider how verification may be performed. One can request systematic checking or reporting. One can then detect immediately any illegal fact or instruction. One can alternatively check or report either randomly or when suspecting an illegal operation. Because we keep provenance information, a principal who wants to check a particular statement can trace "backward" its origin and find who behaved illegally.

Access rights delegation

It is critical for verification to be able to compute who has a given access right on a particular principal. Consider for instance the question *who has write access to Alice's principal*. Three kinds of principals have to be considered: (i) whoever is given explicitly *write* access to her principal; (ii) whoever has higher access right (here *own*) on Alice; and (iii) whoever owns a principal who has write access to Alice, and this recursively. Note that answering such questions is also typically important in practice. For instance, Alice may want to ask the query "who can see my profile?" Note that the transfer of rights via ownerships of (iii) leads to some recursive query processing. The fact that access rights are somewhat not clearly "visible" may be seen as a weakness in terms of data control. One may want to impose the *primitive-only rule* that states that access rights are granted only to *primitive* principals, i.e., principals who are forbidden to be owned by someone else. Any attempt to claim that one owns a *primitive* principal is then viewed as an illegal statement. A natural case of *primitive* principals are *users*. One can consider relaxation of the primitive-only rule to allow *primitive* principals to have *avatars*, e.g., Alice-iPhone. The primitive-only

rules clearly simplify verification.

5.2.4 Physical implementation

In *WebdamExchange*, a specific access-control physical implementation is fully defined by specifying (i) how to use a secret to authenticate a statement and access to protected data (ii) how to use a hint to check the authentication and protect a statement. To illustrate, we next show how two very different and standard such implementations, one using cryptography and the second using login/password, can be used in our model based on hints and secrets.

RSA cryptography

For this implementation, the credentials are a pair of asymmetric RSA cryptographic keys. The identity of a principal is defined by an immutable *own* hint (an RSA public key). We also use in the system a login for readability. The corresponding secret (an RSA private key) is used for authentication. A user typically finds the identity that actually corresponds to the real principal, he wants to communicate with, by external means (e.g., by exchanging it by mail) or via other principals, he already trusts. This problem of linking identity to real entities is not specific to our model. Having the identity of a principal is equivalent to having its *own* hint. Since the key is immutable, this hint never changes.

The creation of a hint/secret pair results in generating an RSA public key to be used as the hint and an RSA private key as the secret. From an implementation point of view, the RSA keys are part of the statement serialization. The system uses such keys for authentication and protection. For authentication, the data is signed using the private key. Everyone can verify the signature with the public key. For example, suppose given the following facts:

Bob *requests* get climbingSite@Roc14&cuvier to Alice-iPhone;
Alice-iPhone *says* Alice-iPhone *states*
climbingSite@Roc14={“id”=“&cuvier”, ...}
requester Alice *at* t to Bob;

The instruction is signed with Bob’s *own* private RSA key while the external knowledge is signed with the *own* private RSA key of Alice-iPhone. The internal statement is signed with the *write* public RSA key of Roc14. Symmetrically, one can use the credentials to protect the data and enforce read access control. For example,

climbingSite@Roc14={...} *encrypted* for reader@Roc14

means that the content of the fact has been encrypted using the *read* RSA key of Roc14. More precisely, the content is encrypted with a fresh DES symmetric key, that is in turn encrypted with the *read* RSA public key of Roc14. So only someone with the private key will be able to decrypt the data.

When access rights are revoked, the signed or encrypted data become obsolete. New keys (i.e., new hint and secret) have to be generated and the new data constructed, distributed and substituted to previous data.

On the Web

We now discuss how to enforce access control using totally different kinds of credentials. More precisely, we consider the access control enforced using URL and login/password credentials, that are typical on the Web.

For login and password credentials, one needs the notion of *authority*. It is a peer that is able to check the password for a given login. For Websites, we define the identity of these peers by their URL. We consider that any data received from a connection to that URL is authenticated by the peer. It means that we trust the DNS system to route properly our connections. Some wrapping software is used to have the Website behave as a *WebdamExchange* peer with typically degraded functionalities. For instance, the provenance information may not be recorded on the Website. Recent hacks of DNS servers have shown limitations, so one may prefer to only trust Websites using Web certificates. Since Web certificates are usually based on RSA cryptography, one may use the RSA implementation previously presented to handle them.

Many Websites have no interesting access control, i.e., anyone can read the data and no one can update it. On Web 1.0 Website, the edition (and some time read access) of data may nevertheless be controlled using a login/password authentication, for example base on a .htaccess file. This is generalized with Web2.0. With social network Websites, there is now an important trend towards even more complicated access control, but still based on login/password authentication. We model this authentication by creating a virtual principal login@server, that the user and the server owns. The (*own*) secret of this principal is its password. Since a password is symmetric, there is no corresponding hint. So this secret is directly used by the user to authenticate its instructions on this virtual principal, by adding the secret to the serialization of the fact. Of course, this fact has to be kept by trusted peers only (in fact, it should be kept only by owner of the virtual

principal). The data of the virtual principal is authenticated by the server, or more precisely by the fact that they are found on the Website.

The most recent systems, such as OpenId or Facebook, also use token-based three-party authentication. The user is redirected by the application to the service, and requests a first token for this application using its credentials (login/password). Then, the users is redirected by the service to the application site. The application can then use the first token to request the authentication token to the service using its credentials (an id and a long chain of bits, which is nothing else than a password). A complete description of this scheme in *WebdamExchange* would be an interesting topic of research. For now, we only consider that the token is a kind of password for the user.

Remark 5.5 (More on URL-based authentication)**.** We quickly introduced URL-based authentication. It has other interesting properties. When using URL-based authentication, the identity of a principal is defined using a (folder) URL. Each fact in this folder (or more precisely each page or file) is by definition authenticated by this particular principal. With that method, a natural way of delegating ownership is inclusion of URLs. For example, if the URL of Alice (as an user of Facebook) is http://facebook.com/Alice/ (she is hosted by http://facebook.com/), then we will consider that http://facebook.com/ owns Alice, since it usually has full access to Alice URL content. This kind of delegation is completed by redirection. If Alice uses an HTTP redirect in her folder to another URL, say http://Alice.com/, then we will consider that http://Alice.com/ has a *write* delegation for Alice. Both types of delegation cannot express more than one delegations. So one may use an hybrid model of *WebdamExchange*, where the implicit data (such as URL or redirection) is completed by regular statements found in the folder, for example access control statements.

5.3 Distribution

5.3.1 Informal presentation

In this section, we consider distribution. As we did for access control, we illustrate that a wide range of standard situations found on the Web, from very simple to very sophisticated, can be handled with the model. Distribution is captured by a particular collection statement, namely *where*. These statements specify on which peers some particular data may be

found. For example, to specify that the list of members of the group Roc14 may be found in the Facebook group of Roc14, one can use the fact:

Alice-iPhone *states*
where@Roc14:{member} **+=**http://facebook.com/Roc14
requester Alice *at* t

The URL *http://facebook.com/Roc14* is understood by the "wrapper" of Facebook as a denotation for that Facebook group. Localization collections are updated as other collections by appending or removing hosts. The language introduces new rights for controlling who can decide where such data may be kept and found, namely, *readWhere* and *writeWhere* rights.

The primary use of localization statements is to enable the localization of a principal's data. For instance, Bob typically has some basic information about Alice that states where to look for information about her, perhaps a Web page where she puts some basic information. Starting from that, Bob is then able to learn that she keeps her list of friends on Facebook, her pictures on Picasa with a backup on her laptop, her music on her TV box at home with a copy on Bob's laptop, etc. So, just with an entry point (i.e., a kind of extended vCard, or electronic business card) Bob can find all the information he needs from her (and he has access to).

Again, the main advantage of our approach is that the system can handle the heterogeneity of the Web. In the same manner that hint/secret enabled using data protected by different access control protocols, the logical form of localization information abstracts away various communication protocols.

5.3.2 Formal model

Given two physical principals *performer* and *host*, a principal *req* and a relation *rel@owner*, a ground localization fact, or *where-statement* is an expression of one of the form:

performer *states* where@owner:{rel}+=host *requester* req *at* time
performer *states* where@owner:{rel}-=host *requester* req *at* time

The statements mean that the relation *rel* (usually a document or a collection id) of the principal *owner* is stored (or not stored anymore) on the principal *host*. The where-statement are clearly special kinds of coll-statements. In particular, they are implicitly persistent. The main difference is that one needs the *writeWhere* right to perform them and the *readWhere* to read them. The instructions corresponding to the statements are straightforward.

We also introduce a special list of keywords, *allStatements*, *allDocuments*, *allCollections* that can be used as relation names to localize a bunch of statements.

To localize data, the policies usually use the following kind of rule:

(at local)
  local *says* $f to $server :-
    $f[@local]//_ *requests* get $docId@$docOwner to _
    [@local]// *states* where@$docOwner{$docId} += $server *requester* _ *at* _

Intuitively, the rule specifies that if an instruction is received locally (the first fact of the rule), one should find where it is located (using the second fact of the rule) and contact the corresponding server (using the *says* constructor). Of course, the rules for distribution interact with the rules for access control. When an instruction is received at the local peer, the above rule is used to obtain the data from a server that has them. Then the access control rule is used to determine whether the data can be sent to this particular client. Observe that the localization information may be already available at the peer or that the local peer may have to use other rules to obtain it, depending on the policies of the principal. Observe also that the rules may typically be more complex, e.g., one might want to check that the client actually has access to the data *before* trying to obtain it or choose between different replicas of the same data.

The where-statements do not really change the interpretation of *WebdamExchange*. It is another meta-data used for data management, that can be used in rules, but which does not change the semantics of the model or the other rules.

5.3.3 Physical implementation

The physical implementation of the direct communication protocols are abstracted away by the localization statements, similarly than for access control. Most of the communication we model is between *WebdamExchange* peers, using a special Web-service. But *WebdamExchange* peers can also communicate with others principals using wrappers (e.g., for Web peers) or avatar interfaces. When a message is sent to a principal, the peer chooses the appropriate means of sending the data.

The wrapper for standard Websites is based on HTTP protocol. The pages of the Website are considered as documents. The localization statements specify the root URL as host for all the data of a given Website. The instructions to the Website principal are translated by the wrapper as GET

(for query instruction), POST (for write instructions), PUT (for append instructions) and REMOVE (for remove instructions) HTTP requests. Returning to our original example, i.e., Alice's iPhone searching for pictures of rock-climbing outings belonging to members of Roc14, the first phase is to find where they store these pictures and to correctly interact with the Websites that stores this pictures, such as Picasa or Flickr. Of course, one may build specific wrappers for special Websites. For example, one may build special wrappers for Flickr, Picasa and Facebook. For blogs, one can also consider the RSS feed of a blog as a collection of documents.

There is a large number of others communication protocols (SSH, FTP, POP, or specific protocols such as the one of PastryDHT) that can be handled similarly. In particular, we discuss in the next section how to make localization more dynamic, for example in the case of gossiping or structured P2P.

5.4 Four policies of interests

We further illustrate the *WebdamExchange* model by discussing how it can support four very different scenarios of data management. These four scenarios are graphically represented in Figure 5.2. Information on trusted peers is represented in white and encrypted data on untrusted ones in grey. The data of principal P1 is marked in bold. With @home and @host policies, all its data is in a single place. With @friend and @host-DHT, it is possibly replicated on several peers.

5.4.1 @home

In the @home scenario, each principal, say Alice, is owned by a single trusted peer, say Alice-host, that hosts the information about her data and access rights. Only the hosting peer can provide data and perform updates for the principal. At the limit of @home, one finds systems where a single peer is "trusted" by everyone and manages the information of everyone (such as Facebook). Encryption is not needed because it is assumed that the host recognizes the principal, which is typically realized using some login procedure then secured communications, e.g., via *HTTPS*.

For each principal for which it stores information, the knowledge base of the host includes particular statements. For example, Alice may choose to use the host Alice-host, for example a social networking site. This system has the secrets of Alice and all her information. In particular, Alice-host has the statements:

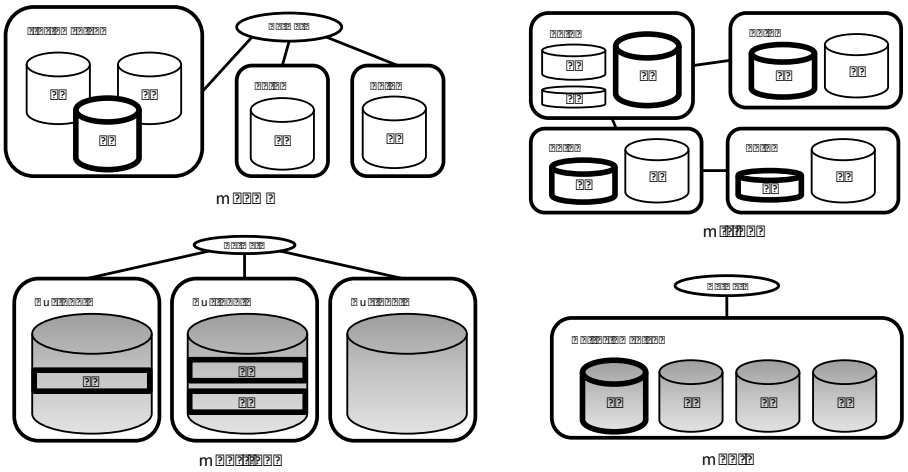

Figure 5.2: Four distribution scenarios

(0) Alice-host *states* ownSecret@Alice *requester* Alice *at* t
(0′) Alice-host *states* ownHint@Alice *requester* Alice *at* t
(0″) Alice-host *states* where@Alice:{allStatements} += Alice-host *requester* Alice *at* t
(0‴) Alice-host *states* owner@Alice +=Alice-host *requester* Alice *at* t

The first two statements record the identity of Alice, the third declares that Alice stores her data on Alice-host and the fourth that Alice fully trusts Alice-host with her data. (The *real* Alice may distinguish between this principal Alice and others for information she holds elsewhere, so Alice here should be interpreted as *Alice in Alice-host*.) Alice-host also stores statements of the form:

(1) Alice-host *states* writer@Alice += Bob *requester* Alice *at* t

that grant access rights to friends of Alice (so that, for example, Bob can updates data of Alice or George can read her data).

Now, Alice-host stores the data statement of Alice, for example of the form

(2) Alice-host *states* profile@Alice := [...] *requester* Bob *at* t

for some document *profile* of Alice that has been updated by Bob. To update the data, principals such as Alice or Bob send update instructions to the

principal, that check they are in the access control list before performing the update, and similarly for get instructions.

The host also stores external localization information for its hosted principals. For instance, if Alice "knows" Bob, then Alice-host stores the "*business card*" of Bob, with the principal identity and hosting peer. This business card is in fact the hint and localization statements:

(3) BobHost *says* Bob-host *states* ownHint@Bob *requester* Bob *at* t to Alice-host
(3') Bob-host *says* Bob-host *states* where@Bob:{allStatements}+= Bob-host
requester Bob *at* t to Alice-host

For example, Alice-host stores that all statements of Bob can be found on Bob-host, e.g., the personal computer of Bob. From that, if a user of Alice-host asks any instruction on data of Bob, Alice-host forwards the instruction to Bob's computer.

5.4.2 @friend

The @friend scenario is an extension of the previous one, based on the fact that people prefer to interact with friends only and are more willing to store data they care about on their machine. Like in @home, a user of @friend has a primary host that handles her information. But she also replicates her data on the hosts of trusted friends. For example, Alice will both store data on his personal computer and on the computer of her friend Bob. Each host now stores both knowledge for the principal it hosts and for the friends of this principal. The external knowledge may be organized as a cache, with a caching strategy such as Least-Recently-Used. Updates to the information of a principal have to be propagated to the different hosts that maintain this information. The host of a principal is the primary copy for its statements. It appears in its localization statements and is in charge of propagating corresponding updates. For instance, the other peers only remember that the computer of Alice is a primary copy of any of her document and they send any corresponding update to it. To retrieve a document $d@q$, a peer p first looks locally, then asks some friends in a gossip manner. The specification of such a basic behavior is easy using our language. To answer a particular query using gossiping, it is important to choose properly who to ask: who is more likely to have the information, to be the specialist, to be trustworthy, etc. By making localization a component of reasoning for answering queries, we simplify

the task. Some kind of routing data structure may also be used to increase efficiency, by introducing more precise localization statements if necessary.

The main improvement of that method is that the availability of the data is better, even if some hosts temporarily leave the network, e.g., because the host is a smartphone and lost data connection. Clearly, to serve an instruction, a host always has to verify that the requester has the corresponding right (following the *sound-rule* spirit). Observe that @friend uses no encryption since everything is founded on trust. We now turn to scenarios where host are not trusted.

5.4.3 @host

In @host, one particular untrusted peer, called *host*, serves all the information. So all information is fully available but encryption is used to protect against misuses of the system. Observe that this policy is very well adapted to the RSA access control implementation and does not really fit with symmetric implementation.

Each principal, e.g., Alice, publishes in *host*:

(0) Alice *says* (Alice *states* ownHint@Alice *requester* Alice *at* t) to *host*
(0′) Alice *says* (Alice *states* writeHint@Alice *requester* Alice *at* t) to *host*
(0″) Alice *says* (Alice *states* readHint@Alice *requester* Alice *at* t) to *host*
(0‴) Alice *says* (Alice *states* Alice where@Alice:{allDocuments}+= *host* *requester* Alice *at*) to *host*

so that Alice is now publicly known. (Observe that none of this data is encrypted). To give *read* right to Bob, Alice publishes in *host* the following statement:

(1) Alice *says* (Alice *states* reader@Alice += Bob *requester*Alice *at* t) to *host*
(1′) Alice *says* (Alice *states* readSecret@Alice *encrypted* for owners of Bob *requester*Alice *at* t) to *host*

The first statement is used to help the host filter instructions. It is not really mandatory, but it makes the global system more efficient. The second statement provides to Bob, the *read* key of Alice, that he needs to read documents published by Alice in *host*. Note that the secret is encrypted so that only Bob has access to it. Now, some peer Sue (assuming proper credentials) publishes data about Alice in *host* with statements of the form:

(2) Sue *says* (Sue *states* profile@Alice = ([data] *encrypted* for readers of Alice)) to *host*

Before accepting such a statement, the host checks that Sue is indeed a writer of Alice, and check that the signature of the statement is done using the *write* hint of Alice. The first step is not mandatory, but increases the efficiency of the system since checking a signature is relatively time consuming. The fact is encrypted such as only readers of Alice are able to read the actual data. Now, Bob can obtain the value of profile@Alice from *host*, by sending the corresponding instruction. It will get the fact

(3) *host says* Sue *says* (Sue *states*
profile@Alice = ([data] *encrypted* for readers of Alice)) to *host* to Bob

It can decrypt it using the *read* secret of Alice and verify its validity by checking the *own* signature of *host* and Sue and the *write* signature of the statement. Similar techniques are used for delegating other rights. Observe that this scheme only protects the secrecy of data. One could consider more complicated schemes for protecting the secrecy of meta-data as well.

Main issues with @host are availability and performance. In particular, if *host* is down, all the information becomes unavailable. Note also that a malevolent *host* cannot forge data but may perform denials of services by returning *fail*() for some data it has, and denials of updates as well by sending old version of the data instead of the most recent one. Indeed, Alice has to suppose that the peer is willing to serve her data, even if she does not trust it to read and write it. We explore next some solutions based on replication to avoid these limitations.

5.4.4 @host-DHT

To "fix" issues raised by the @host system, we use replication. More precisely, we propose a variant of @host, called @host-DHT, that is based on the DHT technology (See, e.g., [TS04]). An interesting work based on declarative specification of a DHT has been proposed in [LCH+05]. One may want to similarly specify it using *WebdamLog*. We are more interested in using standard DHT such as Pastry. Then, one can naturally define the where-statements using standard DHT functions as in:

(at DHT)
DHT *states* where@$owner:{$rel} += $host *requester* DHT *at* t :-
[@DHT]// *states* relation@$owner+={"relId":$rel},
hash@DHT($rel@$owner, $host)

The rule specifies that the peer DHT adds $host to the list of hosts of the relation $rel of $owner if hash@DHT($rel@$owner, $host) holds. The

evaluation of the hash@DHT predicate is supported using a standard Web-service of the DHT, i.e., the classical localization service of the DHT that takes a key as input and returns the address of a peer that has the desired information.

@host-DHT is exactly used as @host except that the information is now distributed among the DHT peers and replicated α times. Each statement is published in the DHT. Observe that we are using the DHT peers as stores and not only as indexes. Like for @host, the DHT peers serve instructions that are properly authenticated. For instance, the document profile@Alice is published by sending the following statements to the corresponding peers, for i in [1..α]:

(0_{DHT}) Alice *says* (Alice *states* ownHint@Alice *requester*Alice *at* t) to peer$_i$,

where *peer*$_i$ is determined using the localization information provided by the DHT discussed earlier. The parameter α is used to manage replication.

We have an extra cost due to replication. In particular, when one revokes some *read* right, one has to select a new *read* secret and republish each replica of each document of the principal. We will see next in more details how replication allows avoiding denial of services, relying on the fact that a majority of the DHT peers are honest. This leads to substantially more complicated schemes that those discussed so far. However, it is interesting to note that we can still support these more complicated schemes with our model, at the price of slightly more complicated rules (not shown here).

The first idea is to send each instruction (query or update) to all the hosting peers in charge of the corresponding statement. Note that it is easy to obtain the set of hosting peers, using the localization statements. Consider queries, the same authentication mechanism as for @DHT allows verifying all the answers we get. Note that there may be conflicts in these answers. With respect to update, note that the peers of the DHT are usually not able to factify statements. So, if a principal can not factify the fact itself, it publishes the corresponding instruction on the DHT (with the data properly encrypted) and waits for some principal to perform it. The requirement to send all statements or instructions to all peers in charge may be computationally too demanding for many applications. It multiplies by α (the replication factor) the number of these facts. Typically, one can use less, say $\alpha' << \alpha$, if the first peers who are asked return the same answer.

Another equivalent counter measure that can easily be modeled using rules and provenance is to accept only messages that have gone through a certain number of peers in charge of storing this relation. An honest peer compares the content of the incoming message (which contains the

instruction fact and the corresponding fact after the first peer of the chain) to the data it stores and transmit valid messages to another peer which is not in the chain of message yet, until α' peers has been encountered. If the peers in charge of a relation are randomly chosen, for example using a hash function as in a DHT, one can give a precise bound on the probability for the adversary to succeed in a denial of update. It has also the interesting property of helping peers of the DHT to synchronize. But the overhead may still be considered too high.

We consider how this can be avoided using two different mechanisms, *time-to-live*, very adapted to document updates, and *consensus*, better tailored to collection updates. Time-to-live relies on the temporal information that is managed in the model and on some basic comparison functions. Consensus requires aggregate functions to count the number of peers who are in agreement and check whether the quorum is reached.

Time-to-live

This method uses an additional kind of meta-data that is provided by the performer of some fact, named *time-to-live* (TTL). Intuitively, the values in this field indicates how long the statement is considered valid starting from the time it was performed. As soon as the TTL has expired, this piece of information is considered stale. So, the data has to be republished regularly, either in *push* mode or in *pull* mode when someone detects that the TTL of some requested information expired.

TTL can be understood as a signed right to distribute some information during a certain period of time but not as a proof of validity of this information during all that time. For instance, Alice can give George her email and claim it is valid for a month. Nothing prevents her from changing it in the meantime. Still George is able to exhibit the old email and prove that it has been legitimately acquired. George can choose a trade-off between freshness and query load, since he can always ask Alice for the freshest information.

The notion of TTL suffers from a common issue in distributed processing: the absence of a global clock. The TTL is a reference to a starting point of time on some machine and to some time interval on that specific machine. It is not a guarantee of global truth. Although TTL may be used also for collections by considering snapshots of the collection, this is not well adapted to rapidly changing large collections because for each new item in the collection, we have to republish the entire collection. We next describe a complementary techniques that is better adapted.

Consensus

Assuming that the DHT peers are in majority honest, one can have a certification of validity for a snapshot of a collection, by consensus of peers. More precisely, we use *group authentication*. Intuitively, the peers in charge of the collection engage in a consensus phase as follows. One of them, randomly chosen each time (the *leader*) contacts the other peers in charge of a particular information (the *partners*) and initiates a consensus phase. The leader sends to the partners the list of updates to the collection since the last consensus, say γ. Each replies with a list of updates to the collection that complements γ. The leader builds a consolidated list of updates. Note that, since the data is encrypted, we have to assume that one still sees in each collection update, the Id of the collection, the nature of the operation and the time of the update. This consolidated list of updates is sent to each partner. If the partner agrees with it (it includes the updates that this particular partner submitted), the partner signs this list and sends the signature to the leader with a TTL annotation. If the leader has enough signatures (say more than a fixed threshold agreed upon in advance), the leader sends to each partner the list of signatures with a TTL and each partner commits the new consensus snapshot of the collection. Observe that a leader who correctly implements the protocol gets signatures from all the honest peers. Otherwise the consensus phase fails, the honest peers detect the failure, and one initiates a new round of consensus.

Now, each time a principal asks for the content of a collection, it is given the consolidated list of updates and the corresponding signatures. The peer also send to it the update statements which are not in the consolidated list. This method guarantees that after enough time, the updates are committed, meaning that the update are in all the consolidated list that the principals would accept. Note that this even works if update instructions are also sent to only some peers (and not all) in charge, at the cost of some delay in the propagation of updates. The update is only seen globally after the next successful round of consensus.

Remark 5.6. A peer in charge of a collection already has the collection updates (if update instructions are sent to all). So instead of exchanging information they already have, one can use Bloom filters to compare sets of update signatures. It then suffices to exchange deltas. It is interesting to revisit the issue of signing the consolidated list of updates to reach the same snapshot of the collection. To do that, one can build the list of updates ordered by append time and sign this list. Alternatively, one can think of the collection as a list of separate append/remove operations ordered by the time of the operation. Signing each one separately would again possibly

lead to update denial. On the other hand, if one signs each together with its index in the collection and the size of the collection, this prevents update denial. The advantage is that one provides to users the *delta* with what they already know and not resend the entire collection.

To conclude this section, let us recall that our main goal here was not to propose new schemes for managing information in a distributed environment but to illustrate how to use our model to support a wide variety of such schemes. In this section, we delved into rather complex techniques such as TTL and consensus. Both the information these techniques need to record (TTL or required size of consensus) as well as the information the peers have to exchange can be seen as *WebdamExchange* knowledge.

5.5 Conclusion

We presented a model for Web data management with access control and distribution that captures a variety of protocols found on the Web. This illustrates the versatility of *WebdamExchange* to describe very different kinds of systems. This unique setting allows expressing very different security policies, very different distribution policies, as well as other essential aspects for distributed data management ignored in this paper, such as translating from the ontology of one peer to that of another.

The *WebdamExchange* language is based on an extension of *WebdamLog*. The goal is to be able to reason about such programs or at least with portions of them. Indeed, proving general properties of datalog programs is in general undecidable but there are cases such as monadic or non-recursive programs where some reasoning is feasible. Reasoning is very useful for a number of reasons, and open interesting directions for future work:

- One can optimize tasks in particular by avoiding useless communications. An open issue is whether standard datalog optimization techniques are adapted to such a setting or whether there is a need for new ones.
- One may be able to prove formal properties of policies. We discussed example of interesting properties of policies, such as *soundness*, *completeness* and *consistency*. We think that *WebdamExchange* provides a sound foundation for analyzing these properties
- Finally, one can reason on the programs of physical principals and some traces of a run of the system, to understand what happened,

and possibly diagnose why some instructions failed or who violated some access control law.

We also need to understand how the rules can be specified for a particular participant. A human participant is very likely to be reluctant to writing rules. So, we have to define ways of selecting some from existing libraries and perhaps customizing them.

Chapter 6

The *WebdamExchange* System

Part of this work has been carried out in collaboration with Serge Abiteboul, Neoklis Polyzotis, Amélie Marian, Émilien Antoine and Kristian Lyngbaek and presented in [AGP11] and [AGL+11].

We implemented the *WebdamExchange* system as a proof of concept of the model. The system fully supports the *WebdamExchange* data model, in an XML format. The system also uses three different kinds of credentials (RSA, URL-based, login/password). The *WebdamExchange* system is based on a core of *WebdamExchange* peers, that can interact with Web peers and iOS peers. Special kinds of Web peers such as blogs, Facebook and PastryDHT are supported. Finally, we demonstrated a social network application on top of the *WebdamExchange* system [AGL+11], based on the motivating example.

6.1 Architecture

6.1.1 System Architecture

The system is currently composed of three kinds of peers:

- the *rich* peers are the main peers of the system, and have most of the advanced functionnalities of the model: they manage all different kinds of credentials and communication protocols with other peers.
- the *client* peers are virtual peers of the system, that communicate with *rich* peers using their avatar interface. The relation is assymetric, since clients can reach peers but peers cannot reach clients. A client is supposed to have few computation capabilities or storage capacities. It can be for example a Web interface or an iPhone.

- the *poor* peers are some servers of the Internet with degraded *WebdamExchange* functionnalities, such as standard Websites or Pastry DHT peers. The rich peers communicates with them via *wrappers*, that translate the various protocols and data of poor peers in the *WebdamExchange* model.

The data of the system follow the *WebdamExchange* model. We use an XML serialization of this model with extensions concerning authentication and encryption, discussed next. At the time this thesis is written, the system is not based on a *WebdamLog* system. The *WebdamLog* development is still in an early stage. So the system does not support exchange of rules and most of the behavior is hard-coded. Nevertheless, the system has been designed to inter-operate nicely with the *WebdamLog* system, as soon as this one will reach a mature level of development.

6.1.2 Data model

The data model of the *WebdamExchange* system is a direct translation in XML of the *WebdamExchange* model. We use Java XML Binding (JAXB) technology to construct a direct equivalence between Java classes, used fo computation, and the XML representation, used for communication and storage. It would be tedious to discuss the precise implementation of the data model. Indeed, the pdf documentation of the data model generated automatically by `doxygen` from our Java code is more than 300 pages long! One can note the use of:

facts These capture acknowledgements, various kinds of statements and instructions, and external knowledge. Basic data is represented as String or as XML. The unstructured documents such as pictures are represented as byte streams. We also have different kinds of contents for secrets and hints, depending of the type of credentials.

principals We support different kinds of principals (for different kinds of peers), with a complex interface inheritance to capture the different facets of principals: computation power, kind of credentials, kind of communication protocol.

annotation The annotations are data that can be added to any kind of facts to capture authentication. There are different kinds of annotation contents depending of the kind of credentials. More generally than in the standard data model, we authorize an unbounded number of annotations for the same fact (using for example different principals

or credentials). This is useful in complex scenarios. For example, suppose that a data, authenticated with a password, has to be send to an untrusted peer. We have to remove the authentication since it contains the password, that should not be disclosed to the untrusted peer. But a trusted intermediary principal may guarantee the validity of internal facts via an RSA signature. If the final target of the fact trusts the intermediary peer, then it can trust the fact without getting the initial password-based authentication.

protection The protections are encrypted data that are used at the place of regular data. There are different kinds of protection contents depending of the kind of credentials. We authorize an unbounded number of protection in the same fact, for example for different principals.

6.1.3 Rich Peer Architecture

The rich peer is implemented in Java and includes:

- a Java representation of the data model, annotated using XML Binding technology to facilitate serialization and deserialization in XML.
- a set of tool classes, for RSA and DES cryptography, for special kinds of data representations such as XML or JSON, for accessing Websites such as Blogs and Facebook and for interacting with Pastry API.
- four interacting modules, namely, Communication, Security, Manager and Storage that are described in Section 6.2.

The peers have a communication module, with a Web-service interface, to communicate with other rich peers. They use special communication module called *avatars* to communicate with clients and *wrappers* to communicate with poor peers. The messages coming from the communication module go through a security module. It is the responsibility of that module to verify the authentication of incoming data (e.g., by checking the RSA signatures) and unprotects incoming data (e.g., by decrypting their content). It is also its responsibility to authenticate outgoing data (e.g., by signing them) and protect them if requested (by encrypting their content). The peers use RSA cryptography for authentication and protection of their data and use special security modules to manage other kinds of credentials.

All the reasoning is performed by the manager module that is responsible for applying access control and distribution policies. In general, the policy of the manager determines (among other responsabilities) whether

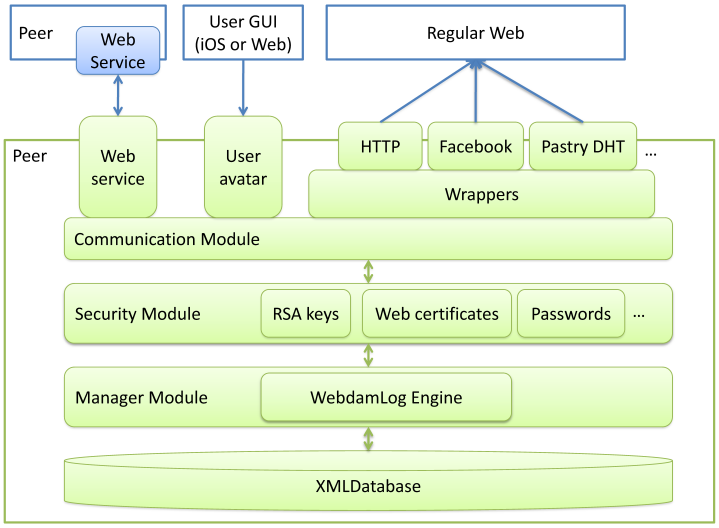

Figure 6.1: *WebdamExchange* Peer Architecture

an instruction is accepted or not and whether a fact is encrypted or not. The facts are stored in an XML database.

The modules of the rich peer are described in more details in Section 6.2.

6.1.4 iOS Architecture

We implemented a *client* peer running on the Apple iPhone (simarly iPad or other iOS devices), in Xcode. The policy of this peer is extremly restricted, since it relies on an avatar interface provided by a rich peer, that is executing most of the complex tasks. This development was meant to illustrate two aspects. First, it shows how to integrate data from a personal device into the global (distributed) information system, when the device is often but not always available on the network. In particular, we consider that the device provides a data store, both from the data of the user accessible on the iPhone and of the data stored by the application in the SQL-like memory of the iPhone. Second, it illustrates how a peer may need to delegate some of its work to other peers.

We also wanted to show how an iPhone user can manage his own secret data or several iPhone users interact privately, without having to trust any other machine or system than their personal devices. However, we encoutered numerous problems with cryptography and key management on the iPhone, restricting largely the autonomy of the peer. These problems

seem to be particular to iOS. A smartphone with Android or Windows operating systems may not raise such issues.

6.2 Peer Modules

In this section, we describe the main modules of a rich peer.

6.2.1 Communication

The *Communication* module handles communications with other peers through (SSL-secured) Web-services. This module also manages communication with poor peers such as standard HTTP server. In particular, it includes wrappers to legacy data, e.g., a wrapper to the Facebook graph API. The communication module is composed of plug-ins for different kinds of communications, and a routage layer, that chooses the correct module to activate for outgoing communications and transmit outcoming communications to the security module.

Web-service

The first plug-in of the communication module is a standard Web-service to communicate with other rich peers. Communication is direct and symmetrical (i.e., any peer can contact any other peer it knows about). It provides one operation :

```
ExtKnow receiveFromPeer(ExtKnow ext)
```

This operation allows another peer to send an external knowledge to the peer, which then replies with another external knowledge. Since the whole process is done using the Web-service and the standard XML serialization, one can easily interoperate with our system using another programming language or operating system, thereby equiping it with the flexibility that is essential for the kinds of applications we have in mind. We also restrict the Web-service to this simple form to avoid painfull development of client interface, which may not be automatized in every programming languages. The Web-service is build in Java using Java Web-service annotations. It is currently deployed on Apache Tomcat 6.0.29, using the Java Machine 1.6.0 (18-b18) of Sun Microsystems, on a Linux machine of INRIA (http://cendrillon.saclay.inria.fr:8090/WebdamExchange/).

Avatar

A second plug-in of the communication module is a more complex Web-service to communicate with an avatar hosted on the peer. It gives to a virtual principal the means to create a transient simple peer with nonpersistent storage, communicating directly with the peer the principal is logged on. From the *WebdamExchange* system point of view, the avatar behaves as if the user was actually physically participating using the user's *own* private key. The interface of the Web-service provides operations to:

- create an avatar given some *own* RSA credentials. The avatar returns a token that is used for further authentication. In our implementation the credentials of the principal are not transmitted to the hosting peer. Note that it would nonetheless be unsafe to use an untrusted peer to host an avatar, since one can not guarantee that the peer implementation is the one we developed.

- send messages to the peer. The main functionnality of the avatar is to provide the RSA cryptography support for the virtual principal (signature and encryption). In particular, the avatar takes care of signatures of the principal, so the instructions and statements can be sent unsigned. After this local step, the incoming facts are transmitted to the hosting peer, which can process the data as if the data came from an external rich peer.

- to get the messages that have been stored by the peer on the avatar. Indeed, the communication is not bidirectionnal, since the peer can not reach the client. So the avatar stores the data sent by the peer to the client and the client has to look-up for new messages regularly.

We also built a simple client, that provides locally the interface of the avatar's Web-service, and a more sophisticated one, that abstracts away the Web-service interface by providing one method for each kind of statements and instructions, that takes care of building the data, sending it and checking the result. In particular, it hides the authentication part and the verification of signatures. This client is used for the communications of our Web GUI with the rest of the system. We implemented a similar version of the client for the iPhone.

Wrappers

The other plug-ins of the communication are called *wrappers*. They are used to initiate communications and fetch data from servers which are not

real *WebdamExchange* peers, such as standard Websites. We have developped a wrapper for Websites, with or without login/password htaccess authentication. This wrapper considers a site as a principal authenticated by its URL and a page of the site both as a document, identified by the local part of its URL, and as a collection of hypertex references (the references found on anchor tags). We also have a special wrapper for blogs (that can interpret an RSS feed as a natural collection).

We also constructed a wrapper for Facebook. This wrapper manages the three-partites authentication scheme to get a token, that is considered as the authentication credential of the user. It then interprets the Facebook graph as documents (for nodes) and collections (for set of edges).

Finally, we constructed a wrapper for the Pastry DHT, that can store the data on the DHT (using the reference of the statements as a key) and lookup statements on the ring.

6.2.2 Security

The *Security* module isolates all the security operations from the rest of the code, making it easier to control this most sensitive part of the system e.g., by supporting it in a separate smartcard, following ideas of [AAB+10]. The role of this module is to authenticate and decrypt facts going in and out of the peer. The basic security module has different plug-ins for the different kinds of credentials. For each operation on the fact (signature, verification of signature, encryption, decryption), the fact is routed to the proper plug-in, that executes the corresponding operation.

For an incoming fact, the module tries to decrypt all the protection and tries to check all the signatures (adding a boolean tag to the annotation depending of the result). The manager module then decides from the boolean what to do with the fact. For an outcoming fact, the module tries to perform the authentication of any partial annotation. The manager module adds some annotations with only the references to the credentials to use for the authentication, and the security module gets these credentials and decides how to enforce the authentication from it. Following the same idea, it performs all the partial protections.

The main plug-in is managing the RSA encryption and signature operations. They are enforced using standard protocols and libraries. In particular, we use SHA1withRSA for signature, RSA/ECB/PKCS1Padding and DESede for encryption, and PKCS8, x509 and DESede for key serialization. We use the XML standard for encryption (XML encryption [W3C02b]) and signature (XML signature [W3C08b]) to serialize the corresponding

data.

The security module similarly includes plug-ins for access by login and password and for authentication based on Web URL and certificates.

6.2.3 Manager

The *Manager* module is in charge of driving the system policies, i.e., running the program of the peer. For now, the program is in Java; so it handles only a finite set of policies. The current version uses class inheritance to manage properly the different policies and provides already an abstract level of reasoning. Typically, the manager first checks the provenance of the data and signatures, and decides to accept the data or not. The manager may accept extended signatures, from principals which are not the expected ones with respect to the *WebdamExchange* model but that have the corresponding right by delegation. Then the manager stores the data using the storage module for tracing provenance. If the data is an instruction, the manager then decides if it has to execute the instruction or not. If the manager decides to execute the instruction, it checks if it can perform the fact locally. If it cannot, it tries to find another peer able to manage the instruction, using localization statements and the manager transmits the fact to that peer. If a peer sent a message to another peer that is not accepted (i.e., if the peer receives a negative acknowledgement), it tries to understand from the acknowledgement the cause of the problem, and if possible to solve the problem (for example by sending a missing hint if a signature verification failed).

6.2.4 Storage

The *Store* module is in charge of local data management, i.e., storing logical statements and the history of messages. Within this module, storage and query processing are supported by an embedded native XML database. We use *eXist*, that is easy to deploy on a Tomcat server. But this particular store can easily be substituted with another XML database. The storage is organized by data references, with a structure giving access to the last valid consolidated version of the data, a trace of provenance of that data for further exchanges, and a complete trace of the history of the data and of the instructions concerning this data.

6.3 Demonstration

A demonstration of the system has been presented at ICDE [AGL+11]. The goal was to illustrate the *WebdamExchange* system by deploying a Social Network on top of the system. The application was based on the example presented in Chapter 2. We developed a special GUI to let the user manages rocks and friends. This GUI communicates with the system via the extended version of the *WebdamExchange* client. It is mostly designed for displaying and manipulating the profile information of the user (basic information about the user and a list of rocks in Fontainebleau), but does not reason on the data and meta-data. A user can create new rocks, bookmark rocks of other users and share this data with other users. To simplify the interface, we provided three levels of access control for each of the data (public (everyone can read the data), friend (the user was able to choose friends among the list of user he knew) and private (no one except the user can see the data)). Except for interface issues, letting the user create more groups would not be a difficulty. The application also fetchs events from Facebook and displays it on the profile. This specific development was mind to illustrate how the *WebdamExchange* system can nicely interoperate with other systems. We also developed the corresponding version of the application for the iPhone, also supported by our extended version of the client. The system was deployed on different peers (server of INRIA, laptop, iPhone), and the user was able to choose which one he trusted to store its data, and to find data of friends stored on other peers of the system.

Chapter 7

Other works

In this chapter, we present some works we participated in, in the context of distributed or social data management, that do not correspond to the precise focus of the thesis. We only briefly sum up them and refer the reader to the corresponding articles for more details.

7.1 Corroboration

The Web provides an interface to access a wide variety of information and viewpoints from individual Web sources that have different degree of trustworthiness based on their origin or bias. A most daunting problem when trying to answer a question is which answer to trust among the ones reported by different Web sources. This happens not only when no true answer exists, because of some opinion or context differences, but also when one or more conflicting answers are reported. Such conflicting answers can arise from disagreement, outdated information, or simple errors.

In [GAMS10], we consider each Web source as a separate view over the data. To accurately answer a question in the presence of conflicting information, a natural approach is to simply count the number of occurrences of each answer, i.e., the number of views reporting each answer. This simple voting strategy performs well in many scenarios but is easily misguided in a Web environment where many sources can either malignantly collude to propagate false information, or naively replicate outdated or wrong data. The quality of the views should then be taken into account when corroborating answers to identify the best answer to a query. Without a priori knowledge on the quality, or trustworthiness, of views, or on the correctness of answers, we are left with a recursive definition: a correct

answer is returned by many trusted views and a trustworthy view returns many correct answers.

We propose fixpoint computation techniques that derive estimates of the truth value of facts reported by a set of views, as well as estimates of the quality of the views. We first introduce a probabilistic data model for corroboration that takes into account the uncertainty associated to facts reported by the views, as well as the limited coverage of the views. Our main contribution consists in three algorithms, namely COSINE, 2-ESTIMATES and 3-ESTIMATES, that estimate the truth values of facts and the trust in sources. They all refine these estimates iteratively until a fixpoint is reached. Their particularities are as follows: COSINE is based on the cosine similarity measure that is popular in Information Retrieval; 2-ESTIMATES uses two estimators for the truth of facts and the error of views that are proved to be perfect in some statistical sense; 3-ESTIMATES refines 2-ESTIMATES by also estimating how hard each fact is, i.e., the propensity of sources to be wrong on this fact.

We presente an experimental evaluation of the algorithms with respect to two baseline algorithms, VOTING and COUNTING, as well as a method from the literature, TRUTHFINDER [YHY07], over both synthetic and real-world data. Our results show that our three algorithms are able to predict correct truth values better than the baseline algorithms in cases where views have various degrees of trustworthiness. Furthermore, we show that in general, 3-ESTIMATES provides better estimates than the other two, which demonstrates the interest of taking into account the hardness of facts.

7.2 Recommendation

In [AYGSY08], we develop a recommendation system called x.qui.site that is based on the *social tag graph*. This graph consists of user and item nodes connected by tagged edges. We pursue this line of work in [AAYG$^+$10]. Our goal is twofold: extract knowledge from this rich social information by clustering social data based on affinity (i.e., proximity in the social network), and provide better query support and navigation on the semantically enriched data. In [AAYG$^+$10], we introduce a general data model where data of interest is captured by a social tag graph. An essential aspect of extracting knowledge from a social tag graph is the identification of groups of users, items and tags. The clustering of these objects is based on a distance called affinity that defines object proximity. It is used for building expressive queries in a QBE (query-by-example) style. Of course, there are important differences between various applications, and due to that,

we rely heavily on different clustering distance measures. However, the thesis of this work is that knowledge extraction, in the applications we are focusing on, shares such a large number of features that it is worth developing a general model capturing them and generic knowledge extraction tools to support it. We also introduce a dynamic tag-based navigation that relies on the incremental graphical construction of queries, facilitating the common tasks one would like to perform in such an environment: consider only portions of the tags (e.g., those by your friends, or those from last week), filter or cluster objects according to certain criteria, get descriptions of objects or groups of objects, zoom on some aspects of interest.

Independently of the previous works, but in the same topic, we wrote a chapter on recommendation systems [Gal11] in the *Webdam* book [AMR$^+$ar].

7.3 Active XML Artifacts

Active-XML artifacts (Axart in short) provide a data-centric approach of workflows. We introduce this artifact model to capture data and workflow management activities in distributed settings in [ABGM09]. The model is built on Active XML, i.e., XML trees including Web-service calls. The model captures the essential features of business artifacts as described informally in [NC03] or discussed in [HNN09]. The main idea of business artifacts is to represent the workflow as data rules involving queries on the documents. In particular, it eases the verification of temporal properties for the systems.

Active-XML artifacts present the following facets that, in our opinion, have to be captured by an artifact model:

State An artifact is an object with a universal identity (e.g., URI). Its state is self-describing (e.g., XML data) so that it may be easily transmitted or archived. It has a host that is a peer or another artifact.

Evolution An artifact is created, evolves in time (possibly space), hibernates, is reactivated or dies according to a logic that is specified declaratively. Its evolution may be constrained to obey some laws, e.g., some workflow.

Interactions An artifact interacts with the rest of the world via function calls (e.g., Web-services) both as a server and a client. An artifact provides for communications, storage and processing for the artifacts it hosts.

History As in scientific workflows, an artifact has a history with time and provenance information that may be recorded and queried.

We also propose a demonstration of the AXART system in [ABMG10]. The system is a distributed platform for collaborative work that harnesses the power of the model. It is illustrated with an example taken from the movie industry. The demonstration scenario considers both standard workflow process and dynamic workflow modifications, based on two extension mechanisms: workflow specialization and workflow exception. The workflows, modeled using artifacts, are supported by the AXART system by combining techniques specific to active documents, like view maintenance, with security techniques to manage access rights.

The AXART system has several interesting characteristics: the simplicity of the interaction with the user, the dynamic modification of the workflow, and the use of artifacts in a distributed environment. The dynamic modification of the workflow and the management of the access rights are new with respect to previous works.

Chapter 8

Conclusion

Let us revisit my journey towards this thesis. My work at Yahoo! Research and on Corroboration was a natural gateway in the realm of Social Networks. The needs of such applications guided us towards sketching out the main features of *WebdamExchange*. To support the reasonning on this model, we were lead designing a language for distributed datalog with exchange of rules. This brought to studying theoritical issues about expressivity and semantics *WebdamLog*. Finally, we returned to Social Network applications by enriching *WebdamExchange* with *WebdamLog* and developing the corresponding system.

From the social network stage, one may try to find new features that will correspond to new properties of our supporting models. One may for example consider the following natural extensions. From a technical point of view, our holistic knowledge-base model would also need to integrate these extensions, that have so far been studied apart.

Non-monotonicity One could consider negation in heads of rules. For instance, someone may state that Bob is not an expert in rock climbing. This may contradict the statement of someone else who states that he is. Clearly, such inconsistencies are frequent on the Web and a comprehensive model for Web data management should take this into account.

Data integration A general issue is how a principal integrates knowledge from other principals. In particular, it may adapt them to its context and own way of representing knowledge, which leads to the fields of data exchange [Kol05] and data integration [HZ96]. Ontologies can be used to structure a participant's vocabulary and to translate knowledge between the vocabularies of different participants in a

distributed environment, cf. e.g., [ACG+06]. Some simple ontology statements, like predicate inclusions (e.g., $Photo \sqsubseteq Document$), can be straightforwardly handled by our proposed framework. However, other important ontological constructs, like existential restrictions ($Parent \sqsubseteq \exists hasChild$) that may introduce incomplete information, are not supported. Extensions of datalog in this direction have been considered, see [CGL09].

Beliefs Peers may also decide that they do not fully believe what they perform or the information that they received. This leads to the field of (perhaps contradicting) beliefs, e.g., [GBKS09], possibly with probabilities [AKSS09]. The statements that are handled in *WebdamExchange* could be interpreted as beliefs by a participant or a group of participants. The degree of confidence may be captured, for instance, by extending the statement with a special value. The fact

Alice *states* where@Bob(album)+=Bob-iPhone
with probability 72% ...

would mean that the Alice believes that the album of Bob can be found on Bob's iPhone with a reasonable probability (72%).

Machine learning This work leads naturally to reasoning about knowledge [FHMV03]. Sue may for instance know that Bob is a good friend of Alice and that he probably replicates the pictures of Alice. Then to get these pictures, she may decide to ask him instead of going to the server. Reasoning about who knows what is an essential component of answering queries in this kind of systems. Note that such reasoning may also be used to acquire knowledge violating secrecy; see, e.g., [FJ02].

Intensional data Finally, Active XML considers intensional data of a very different form, namely functions that may be included in documents and are defined intensionally. It would be interesting to investigate the relationships between the intensional data supported by our model and this other kind of intensional data.

The system stage also questions our ability to optimize the execution of our model's semantics, in particular to reach scalability. There are numerous technical challenges to make the approach feasable. It is specially true for evaluation of *WebdamLog*. But one may also consider problems at the level of *WebdamExchange*. For example, query evaluation in the

context of encrypted data may be extremely inefficient, the basic idea being to get all the possible data, decrypt it, and look at this data locally. One may consider querying encrypted indexes. This is in particular interesting for keyword-based queries; see, e.g., [CM05]. We have extended the model with collections, the essential ingredients for supporting indexing. More work is clearly needed in that direction.

Finally, we may also retain the central point of interest of this thesis: *distribution of heterogenous data on heterogenous systems*. Each stage considers a different facet of the problem, providing new original problems and solutions. Heterogenous data means that identifier and schema may change from one system to another (think of *WebdamLog*), or that the data may have various semantics and usage of the data (think of the different kinds of data in *WebdamExchange*). This last point is an important direction for future research, that is already a topic of investigation by other members of the *Webdam* project, using ontologies or probabilities. Indeed, we provided a global semantics to our system for some of the data, but interpreting data of other systems using our model or providing further semantics to content of document would benefit from more automatic reasoning. Heterogenous system means that they may be completely different and never be intended to communicate (think of poor peers in *WebdamExchange*) or that their behavior is different (think of peers executing their own set of rules in *WebdamLog*). In a world, such as the Web, where systems are specialized (mailer, search engines, social networks…), it is imperative to provides means of supporting heterogenous data management that are both efficient and flexible to ease interoperability, letting the programmer concentrate on creating new applications and functionnalities from building blocks.

Self references

Conferences

[ABGA11] Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Emilien Antoine. A rule-based language for web data management. In *Principles of Database Systems*, 2011.

[ABGM09] Serge Abiteboul, Pierre Bourhis, Alban Galland, and Bogdan Marinoiu. The AXML Artifact Model. In *2009 16th International Symposium on Temporal Representation and Reasoning*, pages 11–17, Washington, DC, USA, July 2009. IEEE.

[GAMS10] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating information from disagreeing views. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 131–140, New York, NY, USA, 2010. ACM.

Workshops

[AAYG+10] Serge Abiteboul, Sihem Amer-Yahia, Alban Galland, Amelie Marian, and Pierre Senellart. Birds of a tag flock together. In *Third Annual Workshop on Search in Social Media*, 2010.

[ABGR10] Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Marie-Christine Rousset. Distributed datalog revisited. In *Datalog 2.0 Workshop*, 2010.

[AGP11] Serge Abiteboul, Alban Galland, and Neoklis Polyzotis. A model for web information management with access control. 14th International Workshop on the Web and Databases, 2011.

Demonstrations

[ABMG10] Serge Abiteboul, Pierre Bourhis, Bogdan Marinoiu, and Alban Galland. Axart: enabling collaborative work with axml artifacts. *Proc. VLDB Endow.*, 3:1553–1556, September 2010.

[AGL+11] Emilien Antoine, Alban Galland, Kristian Lyngbaek, Amélie Marian, and Neoklis Polyzotis. Social networking on top of the webdamexchange system. In *International Conference on Data Engineering*, pages 1300–1303, 2011.

[AYGSY08] Sihem Amer Yahia, Alban Galland, Julia Stoyanovich, and Cong Yu. From del.icio.us to x.qui.site: recommendations in social tagging sites. In Jason Tsong Li Wang and Jason Tsong Li Wang, editors, *SIGMOD Conference*, pages 1323–1326, New York, NY, USA, 2008. ACM.

Book Chapter

[Gal11] Alban Galland. Putting into practice: recommendation methodologies. In *Web Data Management and Distribution [AMR+11]*, pages 365–374. Cambridge University Press, 2011.

External references

[AAB+04] Serge Abiteboul, Bogdan Alexe, Omar Benjelloun, Bogdan Cautis, Irini Fundulaki, Tova Milo, and Arnaud Sahuguet. An electronic patient record on steroids: distributed, peer-to-peer, secure and privacy-conscious. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 1273–1276. VLDB Endowment, 2004.

[AAB+10] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Yanli Guo, Lionel Le Folgoc, Benjamin Nguyen, Philippe Pucheral, Indrajit Ray, Indrakshi Ray, and Shaoyi Yin. Secure Personal Data Servers: a Vision Paper. *Proceedings of the VLDB Endowment*, 3(1), 2010.

[AAHM05] Serge Abiteboul, Zoë Abrams, Stefan Haar, and Tova Milo. Diagnosis of asynchronous discrete event systems: datalog to the rescue! In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 358–367, New York, NY, USA, 2005. ACM.

[Aba09] Martín Abadi. Logic in Access Control (Tutorial Notes). In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V*, volume 5705, chapter 5, pages 145–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[ABCM04] Serge Abiteboul, Omar Benjelloun, Bogdan Cautis, and Tova Milo. Active XML, Security and Access Control. In *SBBD*, volume 4, pages 13–22, 2004.

[ABM04] Serge Abiteboul, Omar Benjelloun, and Tova Milo. Positive active XML. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '04, pages 35–45, New York, NY, USA, 2004. ACM.

[ABM08] Serge Abiteboul, Omar Benjelloun, and Tova Milo. The Active XML project: an overview. *The VLDB Journal*, 17(5):1019–1040, August 2008.

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann Pub, 2000.

[ACC+10] Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell Sears. Boom analytics: exploring data-centric, declarative programming for the cloud. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 223–236, New York, NY, USA, 2010. ACM.

[ACG+06] Philippe Adjiman, Philippe Chatalic, Francois Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *J. Artif. Intell. Res. (JAIR)*, 25:269–314, 2006.

[AH08] Dean Allemang and James A. Hendler. *Semantic web for the working ontologist: modeling in RDF, RDFS and OWL*. Morgan Kaufmann, 2008.

[AHV95] Serge Abiteboul, Rick Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AKSS09] Serge Abiteboul, Benny Kimelfeld, Yehoshua Sagiv, and Pierre Senellart. On the expressiveness of probabilistic XML models. *The VLDB Journal*, 18:1041–1064, October 2009.

[AKSX02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 143–154. VLDB Endowment, 2002.

[AMP05] Serge Abiteboul, Ioana Manolescu, and Nicoleta Preda. Constructing and Querying Peer-to-Peer Warehouses of XML Resources. In *Semantic Web and Databases*, pages 219–225. 2005.

[AMP+08] Serge Abiteboul, Ioana Manolescu, Neoklis Polyzotis, Nicoleta Preda, and Chong Sun. XML processing in DHT networks. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 606–615, Washington, DC, USA, 2008. IEEE Computer Society.

[AMR+ar] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. *Web Data Management*. Cambridge University Press, 2011 (to appear). `http://webdam.inria.fr/textbook`.

[ASV09] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Static analysis of active XML systems. *ACM Trans. Database Syst.*, 34, December 2009.

[AV91] Serge Abiteboul and Victor Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43(1):62–124, August 1991.

[AvH08] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer, 2nd Edition (Cooperative Information Systems)*. The MIT Press, 2 edition, 2008.

[AVM07] Bader Ali, Wilfred Villegas, and Muthucumaru Maheswaran. A trust based approach for protecting user data in social networks. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 288–293, New York, NY, USA, 2007. ACM.

[AW08] Martín Abadi and Bogdan Warinschi. Security analysis of cryptographically controlled access to XML documents. *J. ACM*, 55(2):1–29, 2008.

[Bec80] Leland L. Beck. A security machanism for statistical database. *ACM Trans. Database Syst.*, 5:316–3338, September 1980.

[BFG07] Moritz Becker, Cedric Fournet, and Andrew Gordon. Design and Semantics of a Decentralized Authorization Language. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 3–15, Washington, DC, USA, 2007. IEEE Computer Society.

[Bir05] Kenneth P. Birman. *Reliable Distributed Systems: Technologies, Web Services, and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[BLC90] Tim Berners-Lee and Robert Cailliau. WorldWideWeb: Proposal for a hypertexts project. http://www.w3.org/Proposal.html, November 1990.

[BM10] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.98. http://xmlns.com/foaf/spec/, August 2010.

[BMSU86] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In *Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems*, PODS '86, pages 1–15, New York, NY, USA, 1986. ACM.

[Bry05] Jery Bryans. Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35, New York, NY, USA, 2005. ACM.

[BSVD09] Sonja Buchegger, Doris Schiöberg, Le H. Vu, and Anwitaman Datta. PeerSoN: P2P social networking: early experiences and insights. In *SNS '09: Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*, pages 46–52, New York, NY, USA, 2009. ACM.

[BT07] Peter Buneman and Wang C. Tan. Provenance in databases. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1171–1173, New York, NY, USA, 2007. ACM.

[CCHM08] Tyson Condie, David Chu, Joseph M. Hellerstein, and Petros Maniatis. Evita raced: metacompilation for declarative networks. *Proc. VLDB Endow.*, 1(1):1153–1165, 2008.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.*, 26(2):1–26, June 2008.

[CGI+99] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast security: a taxonomy and some efficient constructions. In *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 708–716 vol.2, 1999.

[CGL09] Andrea Cal'ı, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 14–30, New York, NY, USA, 2009. ACM.

[CH85] Ashok K. Chandra and David Harel. Horn clause queries and generalizations. *The Journal of Logic Programming*, 2(1):1–15, April 1985.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CL82] Shi-Kuo Chang and An-Chi Liu. File allocation in a distributed database. *International Journal of Parallel Programming*, 11(5):325–340, October 1982.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy Preserving Keyword Searches on Remote Encrypted Data. In *Applied Cryptography and Network Security*, pages 442–455. 2005.

[CNP82] Stefano A. Ceri, Mauro Negri, and Giuseppe Pelagatti. Horizontal data partitioning in database design. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, SIGMOD '82, pages 128–136, New York, NY, USA, 1982. ACM.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *Lecture Notes in Computer Science*, chapter 4, pages 46–66. Springer Berlin / Heidelberg, Berlin, Heidelberg, March 2001.

[Dat10] Datalog 2.0. http://www.datalog20.org/, 2010. Oxford Univ.

[DdVPS02] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for XML documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, 2002.

[DF82] Lawrence W. Dowdy and Derrell V. Foster. Comparative Models of the File Assignment Problem. *ACM Comput. Surv.*, 14(2):287–313, June 1982.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, volume 41 of *SOSP '07*, pages 205–220, New York, NY, USA, 2007. ACM.

[FAZ09] Philip Fong, Mohd Anwar, and Zhen Zhao. A Privacy Preservation Model for Facebook-Style Social Network Systems. In Michael Backes and Peng Ning, editors, *Computer Security – ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, chapter 19, pages 303–320. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.

[FHMV03] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about knowledge*. The MIT Press, 2003.

[FJ02] Csilla Farkas and Sushil Jajodia. The inference problem: a survey. *SIGKDD Explor. Newsl.*, 4:6–11, December 2002.

[FM07] Irini Fundulaki and Sebastian Maneth. Formalizing XML access control for update operations. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 169–174, New York, NY, USA, 2007. ACM.

[FMS09] John Field, Maria C. Marinescu, and Christian Stefansen. Reactors: A data-oriented synchronous/asynchronous programming model for distributed applications. *Theor. Comput. Sci.*, 410:168–201, February 2009.

[FSW81] Eduardo B. Fernandez, Rita C. Summers, and Christopher Wood. *Database Security and Integrity*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1981.

[GBKS09] Wolfgang Gatterbauer, Magdalena Balazinska, Nodira Khoussainova, and Dan Suciu. Believe it or not: adding belief annotations to databases. *Proc. VLDB Endow.*, 2(1):1–12, 2009.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[GN08] Yuri Gurevich and Itay Neeman. DKAL: Distributed-Knowledge Authorization Language. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, volume 0, pages 149–162, Washington, DC, USA, 2008. IEEE Computer Society.

[GW10] Stéphane Grumbach and Fang Wang. Netlog, a rule-based language for distributed programming. In *PADL*, pages 88–103, 2010.

[Hel10] Joseph M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Rec.*, 39(1):5–19, 2010.

[HKM78] David K. Hsiao, Douglas S. Kerr, and Stuart E. Madnick. Privacy and security of data communications and data bases. In *Proceedings of the fourth international conference on Very Large Data Bases - Volume 4*, VLDB'1978, pages 55–67. VLDB Endowment, 1978.

[HNN09] Richard Hull, Nanjangud Narendra, and Anil Nigam. Facilitating Workflow Interoperation Using Artifact-Centric Hubs. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *Service-Oriented Computing*, volume 5900, chapter 1, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[Hul89] G. Hulin. Parallel processing of recursive queries in distributed architectures. In *Proceedings of the 15th international conference on Very large data bases*, VLDB '89, pages 87–96, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[HZ96] Richard Hull and Gang Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 481–492, New York, NY, USA, 1996. ACM.

[JOV05] H. V. Jagadish, Beng C. Ooi, and Quang H. Vu. BATON: a balanced tree structure for peer-to-peer networks. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 661–672. VLDB Endowment, 2005.

[JSAV09] Mohamed Jawad, Patricia Serrano-Alvarado, and Patrick Valduriez. Protecting Data Privacy in Structured P2P Networks. In Abdelkader Hameurlain and A. Tjoa, editors, *Data Management in Grid and Peer-to-Peer Systems*, volume 5697 of *Lecture Notes in Computer Science*, chapter 8, pages 85–98–98. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.

[JSS97] Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31+, Washington, DC, USA, 1997. IEEE Computer Society.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells, and Ben Zhao. OceanStore: an architecture for global-scale persistent storage. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, volume 28, pages 190–201, New York, NY, USA, December 2000. ACM.

[KFJ03] Lalana Kagal, Tim Finin, and Anupam Joshi. A Policy Language for a Pervasive Computing Environment. In *In IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 63–74, 2003.

[KGG+06] Sebastian Kruk, Sławomir Grzonkowski, Adam Gzella, Tomasz Woroniecki, and Hee-Chul Choi. D-FOAF: Distributed Identity Management with Access Rights Delegation. In Riichiro Mizoguchi, Zhongzhi Shi, and Fausto Giunchiglia, editors, *The Semantic Web – ASWC 2006*, volume 4185 of *Lecture*

Notes in Computer Science, chapter 15, pages 140–154. Springer Berlin Heidelberg, 2006.

[Kim10] Il-Gon Kim. Static Verification of Access Control Model for AXML Documents. In Guozhu Dong, Xuemin Lin, Wei Wang, Yun Yang, and Jeffrey X. Yu, editors, *Advances in Data and Web Management*, volume 4505 of *Lecture Notes in Computer Science*, chapter 71, pages 687–696. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[Kol05] Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 61–75, New York, NY, USA, 2005. ACM.

[KW94] Brigitte Kröll and Peter Widmayer. Distributing a search tree among a growing number of processors. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, SIGMOD '94, pages 265–276, New York, NY, USA, 1994. ACM.

[LAE+04] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 108–119. VLDB Endowment, 2004.

[LCG+06] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: language, execution and optimization. In *SIGMOD*, pages 97–108, 2006.

[LCG+09] Boon T. Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan,

Timothy Roscoe, and Ion Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, November 2009.

[LCH+05] Boon T. Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, 39(5):75–90, October 2005.

[LHSR05] Boon T. Loo, Joseph M. Hellerstein, Ion Stoica, and Raghu Ramakrishnan. Declarative routing: extensible routing with declarative queries. *SIGCOMM Comput. Commun. Rev.*, 35:289–300, August 2005.

[Lit80] Witold Litwin. Linear hashing: a new tool for file and table addressing. In *Proceedings of the sixth international conference on Very Large Data Bases - Volume 6*, pages 212–223. VLDB Endowment, 1980.

[LLM98] Georg Lausen, Bertram Ludäscher, and Wolfgang May. On Active Deductive Databases: The Statelog Approach. In Burkhard Freitag, Hendrik Decker, Michael Kifer, and Andrei Voronkov, editors, *Transactions and Change in Logic Databases*, volume 1472 of *Lecture Notes in Computer Science*, pages 69–+. Birkhäuser Basel, 1998.

[LM75] K. Dan Levin and Howard L. Morgan. Optimizing distributed data bases: a framework for research. In *Proceedings of the May 19-22, 1975, national computer conference and exposition*, AFIPS '75, pages 473–478, New York, NY, USA, 1975. ACM.

[LMO+08] Changbin Liu, Yun Mao, Mihai Oprea, Prithwish Basu, and Boon T. Loo. A declarative perspective on adaptive manet routing. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, PRESTO '08, pages 63–68, New York, NY, USA, 2008. ACM.

[LNS94] Witold Litwin, Marie A. Neimat, and Donovan A. Schneider. RP*: A Family of Order Preserving Scalable Distributed Data Structures. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 342–353, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[LNS96] Witold Litwin, Marie A. Neimat, and Donovan A. Schneider. LH* a scalable, distributed data structure. *ACM Trans. Database Syst.*, 21(4):480–525, December 1996.

[MKKW99] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. *SIGOPS Oper. Syst. Rev.*, 33(5):124–139, December 1999.

[MMSW07] Maged Michael, Jose E. Moreira, Doron Shiloach, and Robert W. Wisniewski. Scale-up x Scale-out: A Case Study using Nutch/Lucene. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, 2007.

[MS02] Gerome Miklau and Dan Suciu. Cryptographically Enforced Conditional Access for XML. In *Fifth International Workshop on the Web and Databases (WebDB*, 2002.

[MS03] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 898–909. VLDB Endowment, 2003.

[MTKH06] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. XML access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006.

[MZZ+08] William R. Marczak, David Zook, Wenchao Zhou, Molham Aref, and Boon T. Loo. Declarative Reconfigurable Trust Management. In *Conference on Innovative Data Systems Research (CIDR)*, 2008.

[NC03] Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specfication. In *IBM Systems Journal, vol. 42, no. 3*, pages 428–445, 2003.

[NCW93] Wolfgang Nejdl, Stefano Ceri, and Gio Wiederhold. Evaluating recursive queries in distributed databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):104–121, February 1993.

[NCWD84] Shamkant Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. Vertical partitioning algorithms for database design. *ACM Trans. Database Syst.*, 9(4):680–710, December 1984.

[NR09] Juan Navarro and Andrey Rybalchenko. Operational Semantics for Declarative Networking. In Andy Gill and Terrance Swift, editors, *Practical Aspects of Declarative Languages*, volume 5418 of *Lecture Notes in Computer Science*, chapter 6, pages 76–90–90. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009.

[OAS04] OASIS. Uddi version 3.0.2. http://uddi.org/pubs/uddi_v3.htm, October 2004.

[OAS07] OASIS. Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, April 2007.

[ÖV99] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, 1999.

[PRS09] Juan A. Pérez, Andrey Rybalchenko, and Atul Singh. Cardinality Abstraction for Declarative Networking Applications. In *CAV '09: Proceedings of the 21st International Conference on Computer Aided Verification*, pages 584–598, Berlin, Heidelberg, 2009. Springer-Verlag.

[Prz90] Teodor C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundam. Inform.*, 13(4):445–463, 1990.

[RD01] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *SIGOPS Oper. Syst. Rev.*, 35(5):188–201, 2001.

[REG+03] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: The OceanStore Prototype. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2003. USENIX Association.

[RFC74] RFC675. Specification of internet transmission control program. http://tools.ietf.org/html/rfc675, December 1974.

[RR02] Indrakshi Ray and Indrajit Ray. Using Compatible Keys for Secure Multicasting in E-Commerce. In *IPDPS '02: Proceedings*

of the 16th International Parallel and Distributed Processing Symposium, pages 327+, Washington, DC, USA, 2002. IEEE Computer Society.

[SHLX03] Arnaud Sahuguet, Rick Hull, Daniel Lieuwen, and Ming Xiong. Enter Once, Share Everywhere: User Profile Management in Converged Networks. In *Conference on Innovative Data Systems Research (CIDR)*, 2003.

[SW85] Domenico Sacca and Gio Wiederhold. Database partitioning in a cluster of processors. *ACM Trans. Database Syst.*, 10(1):29–56, March 1985.

[TS04] Stephanos A. Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004.

[Vie86] Laurent Vieille. Recursive axioms in deductive databases: The query-subquery approach. In *Proc. 1st Int. Conf. on Expert Database Systems*, pages 179–193, 1986.

[W3C99a] W3C. Html 4.01 specification. http://www.w3.org/TR/html401/, December 1999.

[W3C99b] W3C. Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt, November 1999.

[W3C02a] W3C. Web services conversation language (wscl) 1.0. http://www.w3.org/TR/wscl10/, March 2002.

[W3C02b] W3C. Xml encryption syntax and processing. http://www.w3.org/TR/xmlenc-core/, December 2002.

[W3C04] W3C. Xml schema part 0: Primer. http://www.w3.org/TR/xmlschema-0/, October 2004.

[W3C07a] W3C. Soap version 1.2 part 1: Messaging framework (second edition). http://www.w3.org/TR/soap12-part1/, April 2007.

[W3C07b] W3C. Web services description language (wsdl) version 2.0 part 1: Core language. http://www.w3.org/TR/wsdl20/, June 2007.

[W3C08a] W3C. Extensible markup language (xml) 1.0. http://www.w3.org/TR/REC-xml/, November 2008.

[W3C08b] W3C. Xml signature syntax and processing (second edition). http://www.w3.org/TR/xmldsig-core/, June 2008.

[W3C09] W3C. Owl 2 web ontology language document overview. http://www.w3.org/TR/owl2-overview/, October 2009.

[W3C10] W3C. Xquery 1.0: An xml query language (second edition). http://www.w3.org/TR/xquery/, December 2010.

[WABL94] Edward Wobber, Martín Abadi, Michael Burrows, and Butler Lampson. Authentication in the Taos operating system. *ACM Trans. Comput. Syst.*, 12(1):3–32, 1994.

[Wal03] Dan Wallach. A Survey of Peer-to-Peer Security Issues. In Mitsuhiro Okada, Benjamin Pierce, Andre Scedrov, Hideyuki Tokuda, and Akinori Yonezawa, editors, *Software Security — Theories and Systems*, volume 2609 of *Lecture Notes in Computer Science*, chapter 4, pages 253–258. Springer Berlin / Heidelberg, Berlin, Heidelberg, June 2003.

[Web] ERC grant Webdam. http://webdam.inria.fr/.

[WL82] Paul F. Wilms and Bruce G. Lindsay. A database authorization mechanism supporting individual and group authorization. In *Distributed data sharing systems: proceedings of the Second International Seminar on Distributed Data Sharing Systems*, page 273, June 1982.

[YC77] C. T. Yu and F. Y. Chin. A study on the protection of statistical data bases. In *Proceedings of the 1977 ACM SIGMOD international conference on Management of data*, SIGMOD '77, pages 169–181, New York, NY, USA, 1977. ACM.

[YHY07] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. Truth discovery with multiple conflicting information providers on the web. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1048–1052, New York, NY, USA, 2007. ACM.