Policy Representation and Reasoning with Preferences and Reactivity

Von der
Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation von

Dipl.-Inf. Philipp Kärger

geboren am 6. September 1981 in Leipzig

L3S Research Center, 2010

1. Reviewer: Prof. Wolfgang Nejdl

2. Reviewer: Prof. Matthew Smith

Day of the defense: 30.11.2010

Keywords: policies, preferences, reactivity

Schlagworte: Policys, Präferenzmodellierung, Reaktive Sprachen

Abstract

Policy-based privacy protection in open decentralized information systems such as the Web attracted a lot of research effort in the last years. Policies provide a flexible means to define access control conditions, to realize advanced trust-establishment techniques, and to describe and guide the behaviour of systems.

The change in the use of the Web and its movement from a static information system to a user generated, highly dynamic environment requires policy-based techniques to keep step. New requirements to policy specification and reasoning are set up because, on the one hand, users expose more and more personal data and, on the other hand, the role of the Web as a communication platform gained momentum yielding an increasing dynamicy.

In this thesis, the limitations of nowadays' policy frameworks are analyzed towards two fundamental principles, the specification of preferences and the representation of reactive behaviour. Based on these observations two new approaches for policy representation and reasoning are introduced, namely preference-enabled policies and reactive policies. These approaches allow to incorporate arbitrary preference statements in policy specifications expressed by rule-based languages. They further extend policies to the dimension of events and reactions to go beyond the rather strict grant-or-deny-access principle of current policy-based systems. It is further described how both, preference-enabled and reactive policies can be exploited in policy-based Trust Negotiation, and policy languages as well as protocols for advanced trust establishment based on preference-enabled and reactive policies are developed.

Finally, this thesis describes how policies in general and reactive policies in particular can be exploited for privacy control and trust establishment in dynamic Social Web applications and showcases these findings by describing implementations of the new principles in the context of online Social Networks.

Zusammenfassung

Die Forschung im Bereich Policy-Sprachen und Policy-basierte Zugangskontrolle für offene verteilte Systeme hat in den letzten Jahren stark an Dynamik und Bedeutung gewonnen. Neueste Erkenntnisse konnten vor allem die Zugangskontrolle und die Sicherstellung von Trust sowie die Beschreibung und Steuerung von Systemprozessen mit Hilfe von Policys stark verbessern.

Die Änderung des Nutzerverhaltens im größten offenen verteilten System—dem World Wide Web—stellt aber neue Anforderungen sowohl an Policy-basierte Systeme als auch an die formale Definition von Policys und das Policy-Reasoning. Denn heute ist das Web nicht mehr nur ein statisches Informationssystem, in dem hauptsächlich Informationen konsumiert werden, sondern es ist zu einer dynamischen Kommunikationsplattform geworden, in der immer mehr persönliche Daten veröffentlicht werden.

Unter diesen Vorzeichen werden in der vorliegenden Dissertation zwei neue Konzepte für die Policy-Definition und das Reasoning mit Policys vorgestellt. Zunächst wird untersucht, in welcher Form Präferenzen in der Definition von Policys verwendet werden können. Auch werden Ansätze für das Einbinden von Präferenzen in allgemeine regelbasierte Policy-Sprachen als auch in Policy-basierte Verhandlungen vorgestellt. Des Weiteren führt diese Arbeit eine reaktive Erweiterung der klassischen Policys ein, in der das Ergebnis einer Policy-Auswertung nicht mehr nur "ja" oder "nein" ist, sondern wo Policy-basierte Systeme auf allgemeine Ereignisse reagieren und beliebige Reaktionen initiieren können. Wie auch bei den Präferenzen wird das Prinzip der reaktiven Policys sowohl auf regelbasierte Policy-Spachen als auch auf policy-basierte Verhandlungen angewendet.

Im letzten Teil dieser Arbeit wird analysiert, wie Policys im Allgemeinen und reaktive Policys im Speziellen die Kontrolle von Anwendungen im sogenannten Social Web vereinfachen und so eine stark personalisierte und höhere Sicherheit gewährleisten können. Weiterhin werden konkrete Implementierungen beschrieben, die die Prinzipien der reaktiven Policys im Social Web realisieren.

Danksagung

Die Arbeit an dieser Promotion wäre kaum möglich gewesen ohne die Unterstützung von Kollegen, Freunden und meiner Familie. Zunächst bedanke ich mich bei Professor Wolfgang Nejdl für die Möglichkeit einer Promotion und seine Förderung und Betreuung während meiner Zeit am L3S. Bei Professor Matthew Smith bedanke ich mich für die wertvollen inhaltlichen Hinweise zu dieser Arbeit und seinen Einsatz als Korreferent. Einen großen Anteil am Abschluss dieser Arbeit hat Daniel Olmedilla, der mich von Anfang an zielorientiert betreut hat und mich sowohl methodisch als auch inhaltlich mit Wissen und Fertigkeiten versorgte. Auch Wolf Siberskis Betreuung am Ende meiner Promotion habe ich sehr geschätzt. Daniel und Wolf gilt mein herzlicher Dank für die Unterstützung und Förderung, ohne die ich nicht soweit gekommen wäre.

Die Zusammenarbeit mit Axel Polleres habe ich sehr genossen, und durch ihn habe ich viel gelernt. Ich möchte mich ganz besonders für seine geduldigen Erklärungen und seine Gastfreundschaft bei meinen Forschungsaufenthalten in Irland bedanken. Ein weiterer herzlicher Dank geht an Professor Piero Bonatti. Mit Axel und Piero vor dem Whiteboard erschlossen sich mir viele interessante, wertvolle und für diese Arbeit notwendige Erkenntnisse.

Meinen netten Kollegen am Forschungszentrum L3S habe ich das Arbeitsklima zu verdanken, das durch eine perfekte Mischung aus neuem, vielseitigen Input und fokussierter Forschungsarbeit an meiner Promotion erheblich mitgewirkt hat. Ein besonderer Dank gilt unseren Administratoren Marko Brosowski, Olaf Jansen-Olliges und Dimitar Mitev: ihre Hilfsbereitschaft, Fachkompetenz und Freundlichkeit bilden eine bemerkenswerte und seltene Kombination, die mir oft eine große Hilfe war. Auch bei meinen Studenten Miao Li, Tri-Thong Truong, VenkatRam Yadav, Emily Kigel und Philipp Bähre bedanke ich mich; sie haben mich oft durch Fragen und Interesse motiviert und mich im Implementierungsteil dieser Arbeit unterstützt.

Der Abschluß meiner Promotion wäre ohne meine Familie nicht möglich gewesen. Ein Dank geht daher an meine Eltern und Geschwister, die mich von Beginn an unterstützt haben. Ganz besonders danke ich Fine: für ihre Zeit mit mir und all die Kraft, die sie mir gegeben hat.

Contents

Chapter 1

Introduction

Since the advent of the Internet, the amount of data published online grew tremendously thus posing new challenges to the automated handling of such data and to the control of who has access to which portion of data. Together with this tendency, applications got more and more interleaved with the Web leading to a growing complexity and distributivity. On top of that, the users of online systems changed from specialized IT experts to average people who—more often than not—are not sufficiently aware of privacy risks. At the same time, the ubiquitous access to online resources amplifies the continuous growth of the number of users and the amount of data. All these factors form a big challenge to secure data handling and privacy control in modern information systems.

In the last years, policy-based security control emerged as one of the most promising and flexible solution for privacy protection in open systems. Policies are declarative statements defining the behaviour of a system thus offering the flexibility to separate the actual implementation of a system from its behaviour description which is represented by means of policies. Among the various advantages of policy-based systems one can find that

- for changing the behaviour of a system, only the policies need to be modified without the need to re-compile and re-deploy the whole application;
- policies act as a formal representation of the system's behaviour and can thus be exploited to formally verify software systems or automatically detect potential errors.

Policies are pervasive in security related applications, however, they are able to serve for a variety of general purposes wherever systems feature a high level of autonomy to act on a user's behalf. Such applications are ranging from the rather simple e-mail filters in Outlook to highly complex and dynamic control of multi-agent systems. Although policies are all-purpose means to define a system's behaviour, nowadays, they are mostly applied in scenarios settled around security and access control. Here, by means of rule-based policy languages with a declarative, well-defined semantics, policies are exploited to exactly define under which conditions a potential requester is allowed to perform a certain action, for example to access data. As a matter of fact, policy-based systems have to keep step with the rapid and constant changes in the online world. Thus the expressiveness of existing policy languages has to meet all requirements set up by the ever growing dynamicy of the Web and by the large amount of potentially sensitive data published nowadays.

Due to this fact, the principles behind policy-based systems attracted a lot of research efforts. Formal languages have been analyzed to investigate their capabilities to serve as policy languages. Knowledge representation techniques have been extended to serve as means for policy specification. Collaborative solutions originally developed in the realm of multi-agent systems have been studied and exploited for policy evaluation, policy enforcement, and the automated establishment of trust among users and systems in distributed information systems.

The work described in the present thesis is carried out in context of this movement. It comprises investigations of existing policy languages and identifies two important principles which are pervasive for complex information systems but have not yet been sufficiently addressed and supported in policy-based solutions. Related areas which potentially provide solutions are considered and, where applicable, adopted and integrated to cater for advanced policy representation and reasoning. On top of that, new techniques are developed for policy systems thus extending the state-of-the-art in policy languages and providing new insights into application areas where the new advanced policy principles improve behaviour control and privacy protection compared to traditional approaches. In this thesis, the two principles which are investigated to this respect are *preferences* and *reactivity*. First, it is investigated why both principles are important aspects in modern policy-based information systems and, second, solutions for both directions are developed and applied to various scenarios.

For a policy-based system it is crucial to offer a policy language that is a well-balanced compromise between two contrary dimensions: the expressiveness of the language and its complexity. Concerning the expressiveness of a language, it is required that it offers all the features needed in order to capture the particular domain of interest. There are various examples for such features, starting with more formal properties like the support of negation or approaches for conflict resolution. More practical features include the coverage of conditions which may vary, e.g., conditions may only be allowed to refer to a requester or they may be more general and include external conditions like the temporal context or legacy data sources. Some policy languages support delegation of decisions or the automated information exchange with other systems in order to reach a decision. However, as it will be pointed out in the course of this thesis, features supporting preference expressions and reactive behaviour definition are not sufficiently provided in existing policy frameworks and are thus subject to research. Both issues are briefly sketched in the following, starting with preferences.

Preferences. Requests that are handled by automated policy evaluation sometimes lead to several alternative options to be taken as reaction. In such cases, it is cumbersome to interfere with the automated evaluation process and request the user to decide which of the possible actions to take. Contrarily, a priori statements shall be supported that allow users to arbitrarily rank alternative reactions. Such statements are called preferences: they allow an intuitive representation of a user's intention which data to disclose or which reaction to take for a given request in case several options are valid.

Preference expressions are a well-studied formalism to represent a user's likes and dislikes in a well-defined manner. They are means to express soft constraints and typically rank options a priori before a solution is provided by a system. A typical example for a preference statement is

> "*I prefer green cars to yellow cars, and I prefer yellow cars to any other color.*"

Given a user searching for a car, a system—following the standard preference semantics—will try to satisfy the preference statement at best. That is, roughly speaking, it will return green cars. Only in case no green cars are available at all, yellow cars will be considered and returned. Further, only in case no car in either of both colors is available, the result set will contain cars in other colors. Such preference expressions have been exploited for database search because they provide a very flexible way to express queries whose result set is unlikely to be empty. Other applications of preferences are multi agent systems and planning problems. Here, the behaviour of an agent can be nicely described by a preference order like

> "*Prefer to perform step A, if A is not possible, try B, and so on.*".

In general, preference statements have been proven to be intuitive models of user intentions. This thesis shows that for the definition of behaviour in the context of policy-based systems, the use of preferences is promising as a fundamental extension to existing policy principles. Consequently, such extensions are given for two important aspects:

1. preferences for the credential handling in policy-based Trust Negotiations and

2. the incorporation of expressive preferences in general Logic Programs which serve as a basis for many advanced policy languages.

1. INTRODUCTION

Trust Negotiation is a process that helps systems to automatically establish trust without knowing each other in advance. This is important for sensitive transactions like buying goods or booking a flight online. Here, both parties need to trust each other because sensitive information like payment details (e.g., credit card number), personal data (e.g., home address for shipping), or other sensitive information (e.g., travel date and time, personal interests exposed with the desired book) are disclosed. Trust Negotiation is typically carried out by the exchange of digital credentials which are set as preconditions for one of the partners to trust the other. In policy-based Trust Negotiations these conditions are expressed by means of policies guiding the behaviour of the negotiating agent and the exchange of messages on behalf of real users. However, more often than not, situations arise where more than one credential potentially makes the negotiation succeed. In such a situation, a policy-driven agent cannot easily decide which option to chose: the disclosure conditions of several credentials are fulfilled but only one of those is required to be disclosed for a successful negotiation. A user, however, may not be indifferent among those options, for example, one may in general prefer to provide a less sensitive credential if it is among the alternatives. It turned out that such a scenario is not sufficiently supported in existing policy-based systems. To this end, in this thesis, a methodology will be introduced allowing to incorporate preference statements about credential disclosure into a policy-based Trust Negotiation process.

Preferences are not only important for policy-based negotiations, they should, in general, be first-class citizens of policy languages in order to support any kind of preference-enhanced behaviour description. Although a few policy languages support preference expressions already, they are not sufficiently expressive to cater for advanced behaviour definition. To this end, the present thesis introduces a new Logic Programming language allowing to define preference extensions inside a policy definition. Particular emphasis will be placed on *partial order* preferences, which include the expression of indifferences. An example partial order preference is

> "*I am indifferent between green and red cars, but I prefer both to yellow cars.*"

where, in fact, two options are incomparable (i.e., the user is indifferent between green and red). This is in contrast to total order preferences where any two possible options are comparable. It turned out that so far no Logic Programming language is able to easily capture partial order preferences which are required for indifferences. Moreover, they are also not well supported in nowadays' policy frameworks; in particular, in combination with policy-based Trust Negotiation. Hence, partial order preferences for Logic Programming as they are introduced in this thesis broaden the expressiveness of policy languages based on Logic Programming.

Reactivity. The traditional way of using policies for privacy control is to define conditions that a requester has to fulfill in order to be allowed to perform a certain action. Consequently, a policy-based system's output is binary only, namely, given a policy and a certain request, access is either granted or denied. This principle works well for classical access control scenarios where privacy control is concerned only with granting or denying access to certain resources. But such a request may, on the one hand, not intentionally be a request to access some private resource rather than be part of a general event observed by the system. And, on the other hand, the reaction to such a general event may not only include the acceptance or denial of the request: general reactions like granting access only to a certain extent or further complementary method calls may form a result of a policy evaluation. As an example, a phone call which is received by a system cannot only be allowed or not: reactions may look different dependent on the user's policies and the context of the call; for instance, the call may be forwarded, put on the answering machine, or simply be held. As another example, the request to view a user's profile in an online Social Network platform may lead to several reactions beyond grant and deny. It may, for example, lead to showing only parts of the profile, notifying the profile's owner, or charging the profile visitor.

One of the most prominent examples for policies and, at the same time, the most common application of policies, is the use of filter rules on e-mail clients like Outlook or Thunderbird. Looking at these policies, they also offer more reactions than just allow and deny: e-mails can be moved to folders and can be automatically tagged or deleted. On top of that, the way communication happens on the Internet is not anymore restricted to e-mail: with the advent of social applications such as Twitter, Skype, and Facebook, the possibilities to communicate via Internet increased dramatically, and so did the number of kinds of requests a user may face. These so-called Social Network applications typically offer plenty of ways to communicate among their members, examples are chat messages, wall posts, and micro-blogging posts. Moreover, in addition to such messages, a lot of communication happens via notifications. For example, in Skype the change of a contact's online status is shown in a small notification pop-up. Other kinds of notifications include friendship requests, notifications about being tagged in a picture, etc. In any case, independently of whether a message comes from a single user or if it is a notification generated by the Social Network application, users typically want to canalize the way such messages approach them, just like they are used to with e-mail filters.

Policy languages that are designed for the classical binary access control decisions do not serve for general scenarios like these. For such behaviour descriptions, the definition of reactive behaviour is required which takes into account general events and supports reactions, i.e., rules defining upon which event a certain action is to be executed. Such behaviour is typically expressed by means of so-called

Event-Condition-Action rules. As an example, an Event-Condition-Action policy stating that calls from people who are not students are put on the answering machine looks as follows

ON *call comes in* **IF** *not student* **DO** *put on answering machine.*

To support general policy scenarios beyond the traditional binary evaluations, in this thesis, the classical principle of policy evaluation is amalgamated with the theory of reactive behaviour control: a policy framework is defined supporting Trust Negotiations and complex policy evaluation in combination with Event-Condition-Action rules. This framework has been successfully exploited to enforce reactive policies on the Social Network and communication tool Skype. With this framework, a user is enabled to first provide fine-grained policy specifications stating who is allowed to send calls or chat messages dependent on the particular situation and, second, to specify reactions to events happening in the Social Network.

This thesis is structured as follows. In the next chapter, background information will be given on theories important for the research described in this thesis including the areas policy-based systems, Trust Negotiation, preference modelling, and the Social and Semantic Web. For all these fields, important terms and notations will be introduced and set in context. Subsequently, in Chapter 3, preference-enhanced policies will be introduced and discussed, in particular the use of preferences for policy-based Trust Negotiation and for general, Logic Programming-based policy languages. Chapter 4 deals with reactive policies by first introducing a framework for specifying and evaluating policies in the form of Event-Condition-Action rules and, second, by discussing this framework's applicability to Social Web scenarios and its implementation for policy-based behaviour control in Skype. An extensive review of related work will be given in Chapter 5 where the two areas *preference handling* and *reactive behaviour control* are both considered and respective intersections with policy handling are highlighted and compared to the solutions given in this thesis.

The approaches presented in this thesis have been published in various scientific publications at conferences and in journals. In what follows, the list of publications which prominently contributed to this thesis' content is given. A thorough list of scientific papers which have been published during the course of the presented thesis is provided at the end of this thesis on page 125.

- Section 3.1:
  Philipp Kärger, Daniel Olmedilla, and Wolf-Tilo Balke. *Exploiting Preferences for Minimal Credential Disclosure in Policy-driven Trust Negotiations.* In VLDB Workshop on Secure Data Management (SDM), Lecture Notes in Computer Science, Springer, Auckland, New Zealand, August 2008.

- Section 3.2:
  Philipp Kärger, Nuno Lopez, Axel Polleres, and Daniel Olmedilla. *Towards Logic Programs with Ordered and Unordered Disjunction.* In ICLP Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), Udine, Italy, December 2008.

- Section 4.1.2:
  Piero Bonatti, Philipp Kärger, and Daniel Olmedilla. *Reactive Policies for the Semantic Web.* In 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece, June 2010.

- Section 4.2:
  Philipp Kärger and Wolf Siberski. *Guarding a Walled Garden - Semantic Privacy Preferences for the Social Web.* In 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece, June 2010.

- Chapter 4:
  Jose Julio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. *Towards Reactive Semantic Web Policies: Advanced Agent Control for the Semantic Web.* In Poster and Demo Session of the 7th International Semantic Web conference (ISWC 2008), Karlsruhe, Germany, October 2008.

- Section 4.2:
  Philipp Kärger, Emily Kigel, and Daniel Olmedilla. *Reactivity and Social Data: Keys to Drive Decisions in Social Network Applications.* In 2nd International ISWC Workshop on Social Data on the Web (SDoW2009), Washington, DC, USA, October 2009.

- Section 4.2:
  Philipp Kärger, Emily Kigel, and VenkatRam Yadav Jaltar. *SPoX: Combining Reactive Semantic Web Policies and Social Semantic Data to Control the Behaviour of Skype.* In International Semantic Web Conference (ISWC 2009), Poster and Demo Session, Washington, DC, USA, October 2009.

Chapter 2

Background

This chapter provides background information and introduces notations which will serve as context and basis for the theory described in the subsequent chapters. To this end, first, the term *policy* and its context is made clear, subsequently, focus will be given to the theory of preference modelling. Finally, the terms *Social Web* and *Semantic Web* will be explained since they will serve as application environment of the policy approaches developed.

2.1 Policy Specification and Reasoning

Generally speaking, a policy is the description of a system's behaviour [36]. In computer science, policies are used to describe the intended behaviour of a system: for policy-based systems, a formal policy specification is exploited to govern a system's choices in behaviour [55]. The key-feature of policies in an automated system is the separation of the behaviour description (the policy) and the system interpreting this description and thus realizing the actual behaviour (policy enforcement). This separation facilitates the dynamic change of behaviour in automated systems and permits it to adapt to evolutionary changes in the system and to new application requirements [122]. A very practical implication of this fact is that changes in the behaviour of the system do not require a stop or even a re-implementation of the system but solely a change in the policy specification. Moreover, if the specification of a system's behaviour is represented in a policy, formal processes such as validation or conflict detection can be executed fully automated.

According to Damianou et al. [55], the difference between the policies ruling a system and the actual implementation of the system lies in the abstraction level of the instructions that are given to the system. Policies on the one hand define the choices in the behaviour at a rather abstract level, typically in terms of the conditions under which predefined operations or actions are invoked. Whereas, on the other hand, the actual implementation of a system describes the functionality and the actual operations themselves which are be performed while running the system.

Policies are often used as a means to implement flexible and adaptive systems for management of Internet services, distributed systems, and security systems [55]. Although computer security is the most popular application area of policies (termed "security policies"), the use of policies is established in many aspect of information systems. This is due to the fact that by means of policies any kind of behaviour can be encoded, ranging from business rules and quality of service specifications to access control rules and trust management policies [36].

2.1.1 Types of Policies

There are a lot of different types of policies and for each application area different policy languages with different properties have been developed. It is worth noting that there is no common agreement on the borders between the categories of policies and policy languages. Following the main scenarios addressed by the different types, the terms *network* or *routing policies* [127], *management policies* [123], and *security policies* [126] have been coined in different contexts and eras of policy research. The principles behind those policy types are always the same: a declarative way of describing a system's behaviour which can easily be modified to change the management approach without recoding the management system [55].

Semantic policies

An obvious and common differentiation among policy languages in research is the one between policy languages featuring a well-defined semantics (*semantic policy languages*) and those that do not. Following the distinction presented in [33] and [117], a policy language's semantics is considered well-defined if the meaning of a policy written in that language is independent of the particular implementation of the language [117]. Providing a formal semantics is very important when using a policy language in automated systems due to the following reasons.

- It is important to ensure that the same policy ruling two different systems have the same effects on both systems, that is, the same semantics.
- A well-defined semantics is a requirement for provability: given a policy, it has to be well-defined what consequences will apply given a certain situation; thus, proofs can be provided stating that a certain behaviour will be a consequence or not. Such proofs are typically the basis for explanation generation in policies (cf. [37]), where natural language text is generated justifying the result of a policy evaluation.

- Policy-based automated agent interaction (e.g., Trust Negotiation), is impossible without a common understanding of policies among the interacting peers. Thus, for instance, if Bob has confidence, that he satisfies Alice's policies (e.g., since he has the required credentials), Alice should also conclude that this is the case—otherwise, Bob's actions to fulfill Alice's policy (e.g., by sending his credentials) may not necessarily give access as he believed [117].

Typically, semantic policy languages are built on knowledge representation frameworks that per se feature a well-defined semantics, such as Logic Programs, Description Logics, and Relational Algebra. Throughout this thesis, only *semantic* policies will be considered, hence the terms "policy" and "semantic policy" respectively "policy language" and "semantic policy language" will be used interchangeably.

Policy enforcement

A second important distinction among the existing policy languages and frameworks is dealing with a policy's influence in a system. Policies can act as an upfront enforcement mechanism or as a right expression mechanism [119]. Policy frameworks of the second type are not concerned with the enforcement of the policy itself (cf. [119, 135]); they rather comprise a formal language to express policies and a mechanism to check if certain situations or conditions satisfy the policy. Such frameworks are used where enforcement is impossible or very difficult. For example, a policy stating that a certain resource exposed on a Web site is not allowed to be copied cannot be easily enforced. Typically, if the resource is copied, solely a notification is shown to the user informing about the violation of the policy.

A widely-used framework of this type is the Platform for Privacy Preferences (P3P) [54]. This framework enables users to specify which personal data may be used by service providers while they are browsing the Web. In a privacy preference used in context of the P3P system, a user defines which personal data—e.g., address or age—is allowed to be accessed by a service provider. On top of that, the purpose why the provider is requesting the data and what is going to happen with it, can be part of the P3P preference as well. Consequently, for every Web site visited, the policy of the Web site provider stating which data is required and the policy of the visitor stating the conditions for providing the data are matched and in case of conflicts, a notification is shown to the user. However, P3P does not deal with the fact that the Web site provider can violate its own policy and still use the data for other purposes than specified in the policy. (Further details on P3P, in particular its use of preferences, will be also given in the related work chapter, see Section 5.1.)

An endeavour similar to P3P is been carried out by the Internet Engineering Task Force (IETF), in particular its GeoPriv working group [110]. Here, privacy policies are defined in PIDF documents (presence information data format) in the form of usage rules which describe acceptable usage of location information, such as whether retransmission of the data is allowed or at what date the information expires and must be discarded [61]. Just like P3P, this approach does not provide any means to force a receiver of data to comply with the privacy policy that is attached to it.

In contrast, policy approaches that actually enforce policies do not rely on regulatory pressure to ensure privacy protection [61]: they act as guards in the system and decide to either allow some action or not. Upon a certain request or action, which is protected by a policy, the policy's evaluation is triggered and only if the evaluation gives a positive result, the request is allowed, otherwise, it is not. In the following, unless stated otherwise, this thesis restricts its elaborations on policy frameworks dealing with policy enforcement.

2.1.2 Policies for Security and Trust Management on the Web

Security policies play a role in all facets of today's Web. Access control policies are needed to protect any system open to the Internet. The early policy-based systems like KeyNote [26] and PolicyMaker [28] were already introduced in the nineties of the last century and provided a separation between the access decision to be made and the actual enforcement of the policy. However, the early policy frameworks were rather decision support systems than systems for declarative behaviour control [34]. Nowadays, there is a wide offer of policy languages that have been developed addressing the general requirements for a semantic policy language acting on the Web, namely high expressiveness, simplicity, enforceability, scalability, and analyzability [132]. For the most prominent frameworks handling semantic policy languages, the reader is referred to KAoS [134], Rei [79], PeerTrust [67], and Protune [35, 32]. These policy languages allow for automated reasoning and, with most of them, policies can also be exchanged between entities which is essential for self-initiated agent interaction and negotiations on the Web. Therefore, they are typically based on languages with well-defined semantics; that is in most of the cases Description Logics (e.g., KAoS and Rei) or Logic Programming semantics (e.g., Cassandra [16], PeerTrust, and Protune), but also Relational Algebra (e.g., P3P [140]).[1]

In this thesis, elaborations focus on policies expressed by means of rules. A rule-based policy representation is commonly regarded as the best approach to formalizing policies due to its flexibility, formal semantics and closeness to

[1]For a more comprehensive overview and comparison of policy languages see [52].

the way people think [36]. Another reason is that with Logic Programming, a well-understood declarative semantics can be given to rule-based policies. And, moreover, there are efficient tools for computing solutions of Logic Programs thus easing the implementation of policy evaluation system [124]. On top of that, basic requirements to distributed trust management such as automated exchange of policies and credentials (as it is required for Trust Negotiation, see Section 2.1.3) are more easily expressible in a rule-based fashion that are evaluated according to a Logic Programming-inspired semantics [117].

Throughout this thesis, a standard policy specification will be used (similar to the ones used in [35] and adopted from [141, 31]); policies are written like Logic Programming rules of the following form.

$$\mathit{condition} \leftarrow \mathit{condition}_1, \ldots, \mathit{condition}_n.$$

(with $n \geq 0$), meaning that the condition *condition* holds if all conditions $\mathit{condition}_i$ are satisfied.[2] As an example, the policy

$$\mathit{allow}(\mathit{access}(\mathit{Requester}, \mathit{Resource})) \leftarrow friend(\mathit{Requester}), family\_picture(\mathit{Resource}).$$

states that a requester can access a resource if the requester is a friend and the resource is a family picture.[3] Adding the following facts $family\_picture('holiday.jpg')$. and $friend('Bob')$. will allow Bob to access the picture `holiday.jpg`.

As stated above, policies are well suited to protect digital credentials which may be required to be disclosed in order to establish trust in an online transaction. In the following section, first the term trust is set in context and second, the process of policy-based Trust Negotiation will be introduced which allows a mutual disclosure of credential leading to an automated establishment of trust among strangers.

2.1.3 Policy-based Trust Negotiation

Classical approaches to authorization, like role-based access control do only work well in a closed environments with rather static settings. Access control lists for example do only work if all potential requesters are known in advance and if it has

[2]As syntactic sugar and for the sake of readability, in some places in this thesis a semicolon ; is used to express disjunction of conditions in a rule's body. Following the standard Logic Programming semantics [97], a rule $H \leftarrow B_1; \ldots; B_n$. is a shortcut for the list of n rules of the form $H \leftarrow B_i$. $(1 \leq i \leq n)$.

[3]According to the standard Logic Programming conventions, terms starting with a capital letter refer to variables; that is, *Resource* indicates that the policy applies at least to every resource.

been decided for each requester whether she is on the list or not. For role-based access control a similar limitation applies: before such a system can be used, for each user who shall be able to access to system, a role has to be defined and assigned.

In open environments such as the Web, it is not always the case that two parties know each other before an interaction is started. This makes it difficult to set up a trust relationship needed for a transaction. Consider for example an online shop and a customer who first visits this shop: both have yet no information about the other party. There is no registration telling the shop who the customer is and what rights (e.g., access rights or discounts) to grant. And similarly, the customer has no information about the shop which is needed to decide if, for example, she is willing to give access to personal data such as address, age, or her credit card number.

Trust

The term trust is used to generally describe what is missing in such a situation. Before Trust Negotiation is explained in detail, this paragraph will first introduce the understanding of trust which is used in this thesis in the context of Trust Negotiation. As a matter of fact, there is no common understanding of the term trust. However, in trust management and policy-based systems, the following formulation has been established in [72]: trust is the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context. In the context of this thesis, decisions made based on policy evaluation may involve the consideration of trust. That is, in order to make a policy true or applicable, trust may be required and may potentially be set up first. Such policy decision may for example be concerned with *authorization* [72]: if I develop a trust relationship with a particular student, I may authorize her to perform a certain action. *Authentication* is another example for a policy-based decision which may be built on trust: here a trusted entity needs to issue some identifying certificate. Again, a trust decision is part of a complete policy evaluation. In the following, Trust Negotiation will be introduced as a policy-based means to check and to establish trust by exchanging information.

Exchanging information

To automatically estimate a potential trust relationship between two parties, typically the exchange of information is required. Returning to the scenario above, on the one hand, the buyer may for example check if the shop can provide a specific certificate proving that it belongs to some trusted trading organization like the Better Business Bureau. On the other hand, the shop may require the customer to prove that she is older than 18, registered at a university, or member of a specific customer program like a frequent-flyer program. In general, the service provider as well as the service requester may both require the other peer to prove specific attributes. These proofs can be realized by providing digitally signed credentials [137, 27]. Such credentials are issued by authorities and form verifiable, unforgeable statements about the owner's properties. In a Trust Negotiation scenario, the involved parties are assumed to have available a digital wallet containing different kinds of credentials. Since a credential can contain sensitive information, its disclosure is governed by an access control policy that specifies the credential combination that the other party must disclose before the credential is provided [117]. This is an iterative procedure and Trust Negotiation is the process of credential exchange being the result of a request for a resource. The goal of the credential exchange is to make the request succeed, that is, to make the resource owner confident that the requester is trustworthy for getting access to the resource. In a policy-based Trust Negotiation, each party specifies policies defining conditions for disclosing a specific credential or resource. For example, the condition for disclosing a specific book at an online book store may be that the purchaser is over 18 and provides a valid credit card. On the other hand, a customer's policy protecting her credit card may perfectly say that her credit card is only to be disclosed to parties providing a valid VeriSign certificate. The two policies must be matched in order to find out whether a negotiation exists that satisfies both parties' policies and leads to the provision of the resource initially requested.

A formal model of this process has been introduced in several research papers, for example [138, 141, 35, 51]. All these models share the basic principles of rule-based policies which are protecting credentials and a protocol for exchanging information about policies and the credentials themselves. The formal considerations in this thesis will be based on the Trust Negotiation model first presented in [35] and further extended in [51]. In these publications, Trust Negotiation is presented as a distributed prove based on the exchange of portions of Logic Programs. These programs represent the particular parts of the policy rules that are needed to be communicated among the partners for a specific request. This process—called policy filtering—will be explained in detail in Section 4.1.2. To further illustrate the negotiation process, an example scenario is detailed in the following; it will serve as running example throughout this thesis.

A negotiation scenario

This section presents a detailed example scenario illustrating the process and the message exchanges and highlighting some of the problems to be addressed during a negotiation.

$$
\begin{aligned}
purchase &\leftarrow p_{register}, p_{payment}.\\
p_{register} &\leftarrow (c_{name}, c_{bdate}, (c_{email}; c_{pcode}));\\
& c_{id};\\
& c_{passport};\\
& ((c_{name}; c_{email}), c_{id}).\\
p_{payment} &\leftarrow (c_{bank\_name}, c_{bank\_account});\\
& (c_{credit\_card}, c_{pin}).\\
c_{bbb} &\leftarrow true.\\
c_{osc} &\leftarrow true.
\end{aligned}
$$

Figure 2.1: The book store's policies.

$$
\begin{aligned}
c_{name} &\leftarrow true.\\
c_{birthdate} &\leftarrow c_{bbb}.\\
c_{telephone} &\leftarrow c_{bbb}.\\
c_{email} &\leftarrow c_{bbb}.\\
c_{post\_code} &\leftarrow c_{bbb}.\\
c_{id} &\leftarrow c_{bbb}.\\
c_{passport} &\leftarrow c_{bbb}.\\
c_{bank\_name} &\leftarrow c_{bbb}, c_{osc}.\\
c_{bank\_account} &\leftarrow c_{bbb}, c_{osc}.\\
c_{credit\_card} &\leftarrow c_{bbb}, c_{osc}.\\
c_{pin} &\leftarrow c_{bbb}, c_{osc}.
\end{aligned}
$$

Figure 2.2: Alice's policies.

A user named Alice wants to buy a book in an online book store. For committing the transaction the store requires users to register and also to specify the payment information in order to be charged with the price of the book. The store accepts several registration possibilities involving the disclosure of several different credentials: registration can be achieved by some sort of identification of the user; this is possible by specifying the name, the date of birth[4] and the user's country code or her e-mail address (from which the country of registration can be extracted). Identification can also be achieved via the personal ID card number or the passport number. In addition, registration is automatically performed, if either name or e-mail address, and an ID card number are provided. (Although in this simple example it may be evident that this last possibility of registration overlaps with other policies, it may not be so evident for more complex policies or policy languages, or when the whole policy is specified by more than one single administrator.) Regarding payment, the book store offers two options: either bank account information together with the bank's name or a credit card number together with a PIN must be provided. These combinations of credentials which are required for a transaction are represented as the book store's policy in Figure 2.1.

As stated above, not only a service provider may protect resources with policies but also requesters like Alice may protect their credentials with policies in a similar way. In the present scenario, Alice defined policies protecting her credentials, just like the book store. First of all, Alice does not mind to share her name with anyone, but she specifies some conditions before other information about

[4]Note that in some cases it is possible to apply zero-knowledge proofs (e.g., [95]) in order to avoid disclosure of information (e.g., whether a user is older than 18). This is orthogonal to the present approach since here it is dealt with situations where the actual value *must* be retrieved (e.g., name).

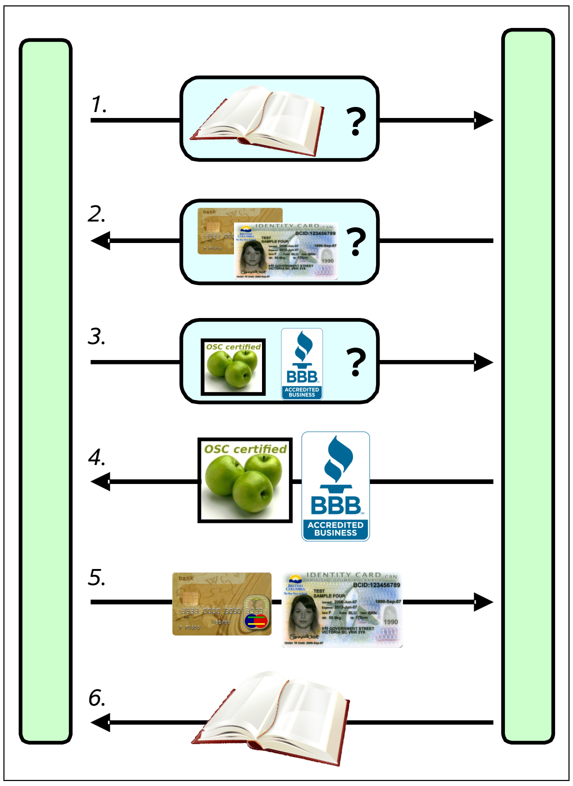


Figure 2.3: One possible successful negotiation between Alice (left) and the book store (right).

her is disclosed. She is willing to provide her general personal information to any entity that is certified by the Better Business Bureau (BBB) seal program that guarantees that its members will not misuse or redistribute disclosed information. However, in order to disclose her bank account or credit card, she defines that an additional Online Security Certificate (OSC) must be disclosed, ensuring that the store Web site is secured and nobody else will be able to retrieve her information during the transaction. Alice's policies are shown in Figure 2.2.

The goal of a Trust Negotiation is, given these policies and Alice's request for a book, to find a way such that Alice finally gets the book. Consequently, a set of credentials need to be exchanged among the two parties in order to satisfy all policies involved in the negotiation. An example exchange for this scenario is shown in Figure 2.3 where Alice asks for the book in Step 1, the book store's negotiation agent replies with a request for an ID card and credit card information in Step 2, and Alice in turn asks for the BBB and the OSC credential in the following Step 3. In Step 4 the book store finally discloses both certificates since no conditions are provided in the store's policies for c_{bbb} and c_{osc}. Since now, after Step 4, Alice's conditions for disclosing her credit card information and her ID are satisfied, her agent discloses both credentials in Step 5 and finally, in Step 6, the book store discloses the book. It is obvious that this is only one potential successful negotiation between the two parties. This scenario will be used later in this thesis to motivate the use of preferences in policy-based negotiations (see Section 3.1). In the next section, a detailed introduction into the area of preference modelling will be provided.

2.2 Preference Modelling

Preferences are a way to model a user's needs, wishes, and expectations. Beyond the specification of a preferred, single desired value or behaviour (such as "I like green.") the notion of preference that is exploited in the present thesis supports alternatives (such as "If possible, I prefer green. Otherwise, yellow is fine as well."). According to Chomicki in [49], the handling of user preferences is becoming an increasingly important issue in present-day information systems. Among others, preferences are used for information filtering and extraction to reduce the volume of data presented to the user. They are also used to keep track of user profiles and formulate policies to improve and automate decision making. Preferences provide means for expressing soft constraints (as opposed to hard constraints). Preference orders, which are built up by assembling several preferred alternatives, represent precise knowledge about what a user would like to get. However, the term "if possible, ..." already suggests that precise and meaningful semantics of these preference orders has to be provided: the soft constraints have to be relaxed step by step until the most preferred option, or rather the most preferred combination of attributes, is found.

As basis for the work presented in this thesis, two different application areas of preference modelling are exploited for rule-based policy representation and evaluation:

1. **Preferences in database retrieval**, typically exploited for selecting the set of optimal objects from a database given a so-called preference query; example applications are e-learning [85, 1] and e-commerce [60].

2. **Preferences in Logic Programming**, in particular for non-monotonic reasoning paradigms. Preferences are pervasive in commonsense reasoning and have a decidedly non-monotonic flavor [58]. Typically, several conclusions drawn from a set of facts are listed in a preference order where the most preferred conclusion is only neglected, if a logical contradiction makes the conclusion impossible. Only in this case, the second preferred option is considered, and so on. Example applications of those principles can be found in algorithmic planning [125] or network policies [25].

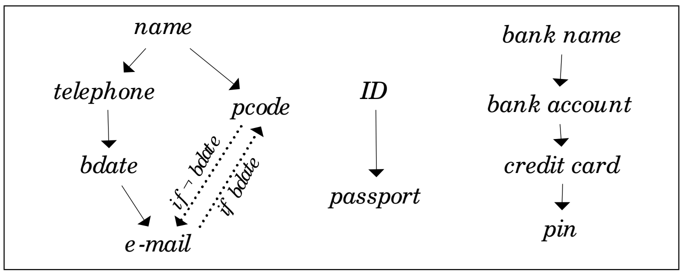


Figure 2.4: An example for qualitative, conditional, partial order preferences stating which credentials are preferably disclosed in a negotiation: the higher a credential in the order, the more preferred.

In the following, different properties of preferences will be explained which hold for preferences in both areas. Subsequently, the two application domains of preferences are introduced in detail and definitions reused in later chapters are provided.

2.2.1 Properties of Preferences

In the following, some important properties of preferences are detailed which will be relevant in later sections of this work. Figure 2.4 serves as an example preference: it shows a preference relation among credentials involved in the Trust Negotiation example from Section 2.1.3.

Quantitative vs. qualitative preferences. The way alternatives in a preference are sorted can either be quantitative or qualitative. In a quantitative (also called weighted or numerical) model of a user's preferences, the value to what extent a user prefers one alternative over another is explicitly given. In a quantitative model, typically numeric scores are associated to the alternatives available. In contrast, qualitative preferences between options are specified directly using preference relations (see Section 2.2.2) [49]. The qualitative approach is strictly more general than the quantitative one since one can define qualitative preference relations in terms of scoring functions but not every intuitively plausible preference relation can be captured by scoring functions (e.g., partial order preferences have no such representation). This lack of expressiveness is well known from utility theory [66, 49]. Moreover, from a usability point of view, it is not desirable that a user has to specify a numerical representation for each attribute and each attribute's value to indicate how much a certain alternative is preferred. This thesis contributes to the qualitative approach to preferences.

Total vs. partial orders. A preference relation defined among a set of alternatives can be a total order or a partial order relation. In a total order, any two alternatives are comparable whereas in a partial order there may be pairs of alternatives where no preference is given. As an example, Figure 2.4 shows a partial order preference on the left hand side (no preference between postal code and date of birth) and two total order preferences. From a user perspective, it may be difficult to define a total order preference for all the attributes; this is a time consuming process and it may well be the case that people are indifferent for some attributes—total orderings cannot capture such preferences. Requiring total order preferences is a restriction that does not fit the world of subjective expressions. Total order preferences do not allow for cases where for several options it is not known which one is preferred, be it due to incomplete information about the world or due to the lack of decision of a user between options. Some parts in this thesis focus on exactly this point: which extensions are needed in a preference theory to allow for indifferences.

Conditional preferences. A preference among two options is not always true per se; it may vary depending on some circumstances. In Figure 2.4, the preference whether the e-mail address or the postal code is preferred to be disclosed, depends on the fact if the date of birth is additionally disclosed or not. Such preferences are called conditional since they depend on some context.

2.2.2 Preferences in Database Retrieval

The notion of retrieval preferences in the context of databases and information systems has been formalized, e.g., by Kießling [89] and Chomicki [49], to describe users' preferences in a way exploitable for selecting optimal objects from a data set. Selecting the right action in a Trust Negotiation or during policy evaluation is basically similar to those personalization techniques, like for instance retrieving the best object out of a database or a digital product catalog. In this thesis, Chomicki's approach presented in [49] will serve as a basis for object comparison in the context of automated Trust Negotiation. In Chomicki's extension to Relational Algebra, preferences are expressed on object level as binary relations over a set of objects O showing certain attributes with particular values.

Definition 2.1 (Object-Level Preference). *Let $A = \{A_1, \ldots, A_n\}$ be the set of available attributes of the elements in O, and U_i, $1 \leq i \leq n$ the respective domain of possible values of A_i. Then any binary relation $\succ$ which is a subset of $(U_1 \times \ldots \times U_n) \times (U_1 \times \ldots \times U_n)$ is a qualitative preference relation over the object set O.*

Typically, preference relations are not directly defined on object level. In the area of database retrieval, users usually need to explicitly provide their preferences on the attribute values of each object attribute. For example, concerning the attribute “model of the car” a user needs to explicitly state that she prefers a Volkswagen to a Nissan. (The attribute preference concerning the model of the car is opposed to the object-level preference that is—in this example—comparing whole cars.) Therefore, preferences are rather stated with respect to the attribute values of each single attribute (in this case the model of the car). Certain values are preferred over others, thus forming a partial order of attribute values:

Definition 2.2 (Attribute Level Preference). *Let A be the set of available attributes of the elements in O and $A_i \in A$ an attribute in such set with U_i its respective domain of possible values. The attribute level relation $\succ_i$, which is a subset of $U_i \times U_i$, is a qualitative preference relation over the value set of A_i.*

The extension of an attribute level preference to respective object level preferences generally follows the well-known *ceteris paribus* semantics [102] ("all other things being equal"). The ceteris paribus condition states that a domination relationship between two objects can only be applied, if one object dominates the other in one attribute and the remaining attributes of both objects show exactly the same values. That is, if $x_i \succ_i y_i$ and $x_j = y_j (\forall j \neq i)$ then x is preferred to y.

Preference composition

After defining preferences with respect to each attribute, objects have to be compared considering all attribute preferences. Therefore it is needed to combine the attribute preferences and build up an object level preference. For such a composed preference, the combined attribute level preference relations are called *dimensions* of the resulting preference relation. Two multidimensional compositions are common [49]: *lexicographic composition* combines any two dimensions by strictly considering one as more important than the other. *Pareto composition* allows to combine two preference relations without imposing a hierarchy on the dimensions: all dimensions are considered as being of equal importance.

A lexicographic composition $\succ_L$ is based on the assumption that one relation can be considered more important than the other, i.e., there is a total ordering between all attributes. Thus, objects are generally ranked according to the more important attribute and only in case of ties the less important attribute is used for ranking. This ranking follows the general idea of the alphabetic order: if the first letter in word A has a higher position in the alphabet than the first letter in word B then A is strictly better than B independent of the subsequent letters.

Definition 2.3 (Lexicographic Composition). *Given the preference relations $\succ_1,\ldots,\succ_n$ over the attributes $A_1, \ldots, A_n$ and assuming a total order among*

$A_1, \ldots, A_n$, the lexicographic composition $\succ_L$ is defined as $x \succ_L y$ iff

$$x \succ_1 y \vee (x_1 = y_1 \wedge x \succ_2 y) \vee (x_1 = y_1 \wedge x_2 = y_2 \wedge x \succ_3 y) \vee$$
$$\ldots \vee (x_1 = y_1 \wedge \ldots \wedge x_{n-1} = y_{n-1} \wedge x \succ_n y).$$

In contrast, Pareto composition yields a new preference relation following the fair principle of Pareto domination: an object X is said to *Pareto-dominate* another object Y iff X shows better or equal values than Y with respect to all attribute preferences and is strictly better with respect to at least one attribute preference.

Definition 2.4 (Pareto Composition). *Given $\succ_1, \ldots, \succ_n$ as preference relations over the attributes $A_1, \ldots, A_n$, the Pareto composition $\succ_P$ is defined as:*

$$x \succ_P y \Leftrightarrow (\forall i : x \succ_i y \vee x =_i y) \wedge \exists j : x \succ_j y.$$

The evaluation of Pareto composition for database retrieval is often referred to as "skyline queries" and is a direct application of the maximum vector problem [91]. Like for the preference modeling recently a lot of research has been invested to find efficient skylining algorithms, as e.g., [39, 106, 14].

Amalgamating preference relations

Specifying preferences for each single attribute leads to the challenge of combining them on the object level. But if an object is better than another in terms of one attribute and worse in terms of another, there is no way to compare both although the first attribute may be considered more important than the second. As an example, a user may want to state that although she prefers a Volkswagen to a Nissan, this preference may be disregarded in favor of the preference for a cabriolet. For instance, given the preference that a cabriolet is preferred to a coupe, a Nissan cabriolet may still be better than a Volkswagen coupe—although, from a Pareto point of view, both are incomparable.

To overcome this problem, the concept of preference amalgamations (or trade-offs) has been proposed in [12] forming a useful extension of Pareto composition. It does not only consider all attributes equally desirable, it additionally allows the user to specify a connection between two or more attributes. This is especially helpful, because in many practical applications, users are willing to perform trade-offs, i.e., relax their preferences in one or a set of attributes in favor of an object's better performance in another set of attributes.

Definition 2.5 (Amalgamated Preference). *Given the preference relations $\succ_1, \ldots, \succ_n$ over the set of attributes $A_1, \ldots, A_n$, a set of amalgamated dimensions $\mu \subseteq \{1, \ldots, n\}$ with cardinality k, as well as two $k-$dimensional tuples X_μ, Y_μ restricted to attributes with indices in μ, then (with π as the projection in the sense of Relational Algebra) the function $AmalPref(X_\mu, Y_\mu)$ is defined as:*

$$\begin{aligned}AmalPref(X_\mu, Y_\mu) := \{(x, y) \mid\ & \forall i \in \mu : (\pi_{A_i}(x) = \pi_{A_i}(X_\mu) \wedge \pi_{A_i}(y) = \pi_{A_i}(Y_\mu)) \wedge \\ & \forall j \in \{1, \ldots, n\} \backslash \mu : (\pi_{A_j}(x) = \pi_{A_j}(y) \vee \pi_{A_j}(x) \succ_j \pi_{A_j}(y))\}.\end{aligned}$$

An amalgamated preference is a relation denoted by $\succ_\mu^{X_\mu, Y_\mu}$ *such that*

$$x \succ_\mu^{X_\mu, Y_\mu} y \Leftrightarrow (x, y) \in AmalPref(X_\mu, Y_\mu).$$

In general, the intuition of this definition is that given two tuples X_μ , Y_μ from the same amalgamated dimensions given in μ, the relation $\succ_\mu^{X,Y}$ is a set of pairs of the form (o_1, o_2) where the attributes of o_1 projected on the amalgamated attributes equal those of X_μ, the attributes of o_2 projected on the amalgamated dimensions equal those of Y_μ, and furthermore all other attributes (which are not within the amalgamated dimensions defined in μ) are identical in o_1 and o_2. The last requirement again denotes the ceteris paribus condition (cf. page 29), i.e., the dominated object has to show equal values with respect to all non-amalgamated attributes.

2.2.3 Preferences in Logic Programming

In the area of Logic Programs, as one of the major ways to represent policies, preferences have been introduced as well to express orders between possible outcomes or consequences of a reasoning process. This is in particular interesting for non-monotonic reasoning techniques where *default* assumptions are allowed, that is, facts that are true unless their contrary is explicitly expressed. In such a setting, preferences are applied between defeasible consequents of rules such that the most preferred non-defeasible consequents survive. Thus, rules or facts that may be contradictory without preferences, lead, empowered with preferences, to an optimal solution. For a general overview of preferences used in non-monotonic reasoning approaches the reader is referred to [58]. In the present thesis, preferences for non-monotonic Logic Programming with so-called stable model semantics (known as Answer Set Programming) has been applied. This section introduces the terminology and background information on Answer Set Programming needed for later elaborations in this thesis and introduces LPODs, a way to formulate preferences in Answer Set Programming.

Stable model semantics

The stable model semantics extends the typical least model semantics for Logic Programs (where all rules are definite, i.e., negation-free Horn clauses) such that so-called Normal Logic Programs, i.e. programs allowing negation as failure in rule bodies, get an intuitive semantics. Logic programming under the answer set semantics, often referred to as “Answer Set Programming”, further extends the stable model semantics by features such as various forms of disjunction. In the following, first, Disjunctive Logic Programs (DLPs) are shortly introduced together with some extensions used later in this thesis. Subsequently, an approach for preferences under the stable model semantics will be introduced, namely Logic Programs with Ordered Disjunction (LPOD).

Disjunctive Logic Programming

Disjunctive Logic Programs (DLP) extend standard Normal (disjunction-free) Logic Programs such that disjunction in a rule's head is supported. DLPs allow for representing problems of lower complexity in a simple and more natural fashion [59]. In this thesis, the standard notation for Disjunctive Logic Programs is used (cf. [15, 113]). Roughly speaking, a DLP is a set of rules of the form

$$h_1 \vee \ldots \vee h_l \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n.$$

Its syntax is formally defined as follows. Given disjoint sets of predicate and constant symbols, σ^{pred}, σ^{con} respectively, atoms can be defined as $p(t_1, \ldots, t_k)$ where $p \in \sigma^{pred}$, $t_1, \ldots, t_k \in \sigma^{con}$ and k is called the arity of p. Atoms such that $k = 0$ are called *propositional.* A literal is either an atom a or its negation $\neg a$ where '$\neg$' represents classical negation.

Definition 2.6 (DLP). *A Disjunctive Logic Program (DLP)*[5] *P is defined as a set of rules r of the form*

$$h_1 \vee \ldots \vee h_l \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n$$

where each h_i (b_j) is a literal and not *represents negation as failure. It is further defined:*

- *$Head(r)$ to be $\{h_1, \ldots, h_l\}$,*
- *$Body^+(r)$ to be $\{b_1, \ldots, b_m\}$,*
- *$Body^-(r)$ to be $\{b_{m+1}, \ldots, b_n\}$, and*
- *$Lit(P)$ to be the set of all literals occurring in P.*

The semantics of DLPs is defined by its disjunctive stable models, also called answer sets. A set of literals S is an answer set of P if and only if it is a minimal model of the Gelfond-Lifschitz reduct P^S. In this thesis, a formal definition of stable model semantics is not given, the reader is referred to [69, 113] for further details.

[5]In this thesis, elaborations and examples on answer set programs are restricted to propositional programs. As usual in answer set programming, rules containing variables—also called rule schemata—may be considered as representations of their instantiations where variables are replaced by the constants occurring in the program and the semantics is defined in terms of the set of all possible rule instantiations (also called the program's grounding).

Head-cycle freeness. Head-Cycle Free Logic Programs [19] are a special kind of Disjunctive Logic Programs which will be of interest later in this thesis. They are defined based on the notion of a program's dependency graph:

Definition 2.7 (Dependency graph). *The dependency graph of a Logic Program P is defined as a directed graph where every literal that occurs in P is represented as node l and there is an edge from l' to l if there is a rule in P such that $l \in Head(r)$ and $l' \in Body^+(r)$.*

Definition 2.8 (Head-Cycle Free Programs). *P is head-cycle free if its dependency graph does not contain directed cycles that go through two literals occurring in the same rule head.*

Cardinality constraints. Later in this thesis, a specific syntax extension for DLPs will be exploited, called *cardinality constraints* [120]. This extension allows heads of rules to be a cardinality constraint of the form $L \ \{l_1, \ldots l_n\} \ U$. Here, $l_1, \ldots, l_n$ are literals and L (lower bound) and U (upper bound) are natural numbers. The intuition is that this statement holds if at least L and at most U of the literals $l_1, \ldots, l_n$ are satisfied.

Logic Programs with Ordered Disjunction (LPOD)

There is a lot of work about modeling and exploiting preferences in Logic Programs with stable model semantics, the reader is referred to [58, 105] for a complete overview; more details are also given in Section 5.1.2. The most prominent work which also serves as a basis for the present thesis was presented by Brewka et al. in [44]. Brewka describes so-called Logic Programs with Ordered Disjunction (LPOD) for expressing preferences in Logic Programming based on a special kind of disjunction called *ordered disjunction* (denoted by $\times$). It expresses a disjunction while at the same time building up a preference order between the single disjuncts. This ordered disjunction is—similar to DLP—only allowed to appear in a rule's head. A typical example rule is

$$c_{name} \times c_{telephone} \times c_{bdate}.$$

This rule states that c_{name} is preferred to be true, i.e., is preferred to be contained in the answer set. If for some reason c_{name} can not hold, for example the policy for disclosing it is not satisfied or it is not required to make the negotiation succeed, $c_{telephone}$ would be the second option, and so on.

Given several such rules in a program, the preference which is built up by $\times$ acts like an attribute level preference (cf. Definition 2.2) and each rule having the $\times$-sign sets up a new attribute. The object level preference (cf. Definition 2.1) acts on the actual answer sets which are models for the program: given two answer sets which both are models of the program, one answer set is better than the other if for each rule a "better" literal is satisfied—that is, if the leftmost literal satisfied in the answer set appears before the leftmost literal satisfied in the other answer set. A formal presentation of LPODs and their semantics can be found in [44].

2.3 The Social and the Semantic Web

Parts of the scenarios developed in the present thesis play an important role in a specific aspect of the World Wide Web. In particular, the contribution of reactive policies is shown in an application acting on the Social and Semantic Web. This section introduces both terms and explains them in detail.

The Web, as the most prominent open information system, is increasingly becoming social: there has been a shift from just *existing* on the Web to *participating* on the Web [29]. Indicator of this movement is the popularity of online tools like wikis, blogs, sharing platforms, or Social Networking sites [90]. These applications typically offer the same basic functionality: network of friends listings, person surfing, private messaging, discussions in forums or communities, event management, blogging, commenting and resource sharing [40]. The term *Social Web* has been introduced to refer to this particular part of the Web that is highly social, conversational and participatory [41]. Beyond the social aspects of the Web's use, in this thesis also the social aspects of the information accessible on the Web will be of particular interest. Thereby, it is referred to social information for any personal data that is describing persons or the relationships among persons. Thus, the Social Web can be considered a repository of social data available to the public.

An extensive exploiting of this information is hindered by one of the major limitations of nowaday's Social Web [40]: Web sites storing social data of various kinds do not make use of common data formats and knowledge representation standards. Thus, although social information is visible to a Web site's visitor, it is not at all machine-understandable, and automated systems are not enabled to grasp and reuse this information. This is an instantiation of a general problem of the current Web: it lacks a well-defined representation of its content. To address this problem, an extension to the traditional Web has been proposed by Tim Berners Lee in 2001 [21]: the so-called Semantic Web, "an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." The last couple of years have seen large efforts going into the definition of foundational standards aiming at a universal data exchange among heterogeneous systems. Currently, a well-defined Semantic Web technology stack exists [41] (see Figure 2.5), ranging from metadata standards and associated vocabularies to reasoning mechanisms on Web data and semantic techniques for privacy, provenance, and trust establishment (e.g., semantic policies in Section 2.1.1). Among the findings and developments of the Semantic Web research community there are vocabulary standards which achieved wide deployment. The following three standards are widespread for social information:

FOAF (Friend-of-a-Friend)[6] is an ontology for describing people and the relationships among them.

SIOC (Semantically-Interlinked Online Communities)[7] is another ontology defining concepts to uniformly represent information about blogs, forums, and mailing lists [29].

DOAP (Description-Of-A-Project)[8] is a standard to describe software projects and people working in them.

Below all those vocabularies, as the standard way to represent semantic information on the Web, RDF (Resource Description Framework) [100] is used (cf. the tower in Figure 2.5). In RDF, information is stored by means of triples statements $(subject, predicate, object)$. The *subject* is an identifier for a resource, *predicate* is an identifier referring to a relationship, and *object* is the resource that relates to the *subject*. A set of such triples builds up a so-called RDF graph consisting of nodes which are URIs, blank nodes or literals, and edges which are URIs referring to relationships. To provide a common understanding of RDF statements among different sources, the so-called RDF-schemas are typically used. They are expressed in RDFS[9], a language that allows for defining the semantics of RDF statements and therefore serves as an ontology language. One layer higher, the web ontology language OWL[10] extends RDFS with more advanced constructs for knowledge representation. This way, one is empowered to, for example, specify that two URIs refer to the same resource using the `owl:sameAs` predicate. In fact, FOAF, SIOC, and DOAP are all provided in form of RDFS or OWL ontologies. The standard technology to query RDF data sources is SPARQL[11] (SPARQL Protocol and RDF Query Language), a query language for RDF graphs. RDF data is typically exposed on the Web in form of Web services wrapping an RDF repository, the so-called SPARQL endpoints.

In the last years, Semantic Web standards started to get woven into the Social Web: more and more social data is stored and exposed by means of semantic technology and thus form the so-called Social Semantic Web [41]. Parts of this

[6] `http://www.foaf-project.org`

[7] `http://sioc-project.org`

[8] `http://trac.usefulinc.com/doap`

[9] `http://www.w3.org/TR/rdf-schema/`

[10] `http://www.w3.org/2004/OWL`

[11] `http://www.w3.org/TR/rdf-sparql-query`

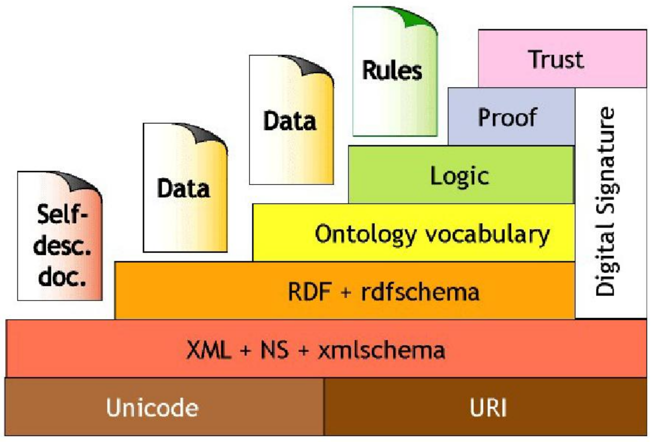


Figure 2.5: The Semantic Web Layer Tower [9] introduced by Tim Berners-Lee: upper layers add semantics to lower layers while reusing the formalisms of the layers below. For example, vocabularies are defined by means of RDF whereas deduction and proofs in turn exploit those vocabularies to handle implicit knowledge.

thesis will investigate how this fact can be exploited for policy-based privacy protection and behaviour control in applications acting on the Social Web. These co-called *Social Network applications* include Web platforms which directly act on the Social Web like Facebook, but also any other application based on Social Networks such as Skype. The principle of policy-based privacy control is a good candidate for privacy control on the Social Web. It is flexible enough to capture various situations and user preferences. On top of that, the social data semantically exposed on the Web can be exploited by a policy system thus incorporating arbitrary social information into the policy decision.

Chapter 3

Preference-enabled Policy Representation and Reasoning

Preference models are widely used for the representation of user intentions, be it for searching in databases or for the personalization of systems. In particular qualitative preferences which do not require the user to define numeric degrees of preference can be easily stated by users and still be exploited for automated decisions or option selection. Thus, preferences are a good candidate to be exploited in policy-based behaviour control: they offer a straightforward way to rank options that otherwise were equal alternatives. But preference models do not only allow for an a priori ranking of alternative reactions, they also support expressions ranking specific properties of alternatives which are generalized in order to do an a posteriori comparison among objects. Thus, expressing preferences in policy languages is a promising extension that eases a lot the process in encoding user intentions in formal languages.

This chapter describes an approach which incorporates preferences into the policy specification and reasoning process. Particular emphasis is placed on the expression and the evaluation of *partial* order preferences (cf. Section 2.2.1), not only because all the existing solutions are restricted to total order preferences but also because indifferences (which are only possible in partial orders) are pervasive in user-expressed preferences. In Section 3.1 in this chapter, a formal approach is described that incorporates preferences in a Trust Negotiation process. Here, preferences are exploited in order to automatically decide which of the possible next steps is the most preferred. This decision happens right after the policy is evaluated and the alternative steps are computed. The considerations about how preferences are evaluated in a Trust Negotiation directly lead to the elaborations and findings in the subsequent section: in Section 3.2, the problem of how such preferences can be generally encoded in a rule-based policy is tackled. Therefore, a more general view is taken and arbitrary Logic Programs are considered. To this end, DLPOD is introduced, a language for Logic Programs that allows to specify arbitrary partial order preferences on logical predicates inside a policy. To achieve this, an approach for encoding *total* order preferences in Logic Programs is extended towards partial order preferences.

The following contributions will be presented in this chapter:

- an approach for credential selection in automated Trust Negotiation based on qualitative partial order preferences featuring preference composition, preference amalgamation, and conditional preferences (first presented in [86]);
- an extension to Logic Programs with Ordered Disjunction, called DLPOD (Disjunctive Logic Programs with Ordered Disjunction) which supports partial order preferences among literals in rules' heads (first presented in [84]).

3.1 Exploiting Preferences for Policy-based Trust Negotiation

The process of Trust Negotiation is based on credential exchange which is guided by the policies. Policies, as it has been made clear in the former sections, are declarative statements which—in the context of Trust Negotiation—define conditions for requesters to access a resource or credential. However, in some cases, it is not exactly determined what credential to disclose; then, additional information has to be provided by the user indicating which option to chose. For example, the work by Yu et al. [141] allows users to define generic strategies that influence the message exchange of a negotiation beyond the actual policies which are protecting the resources. Yu's approach is thus giving the user the possibility to generally specify a negotiation's behaviour, for example by means of criteria when to stop a negotiation or when to disclose an unlocked credential (speeding up the negotiation with less or more cautious strategies). Still, the particular problem of personalization in terms of which successful negotiation path to choose among several alternatives has not been studied in [141].

Looking back at the example negotiation from Section 2.1.3, one can see that in order to register, Alice has at least two options: either she provides her passport credential or her digital ID. And in case the bookstore already disclosed a BBB credential, she is offered two options in order to register at the online book store. Generally speaking, there are situations where two or more reactions are possible in a Trust Negotiation.

Picking the right, i.e., the preferred action in a Trust Negotiation is similar to the personalization techniques known as preference queries which return for instance the optimal objects from a database or the most preferred good from a digital product catalog. In the situation of Trust Negotiation, the objects are the sets of possible negotiation steps and their attributes are the different credentials that either have to be disclosed in a step or not. In the following section, the requirements for a solution will be defined based on the negotiation scenario presented in Section 2.1.3.

3.1.1 Problem Definition and Requirements

By matching the policies of both, the book store and Alice, one can find all possible negotiation paths, that is, all the credential disclosure sets that will make the negotiation succeed. How to extract such a negotiation path is a standard process (cf. Section 2.1.3). The matching of the two example policies presented in Section 2.1.3 returns several possible negotiation paths and there exist 12 different credential disclosure sets (see Table 3.1) leading to a successful negotiation. Therefore, Alice has to select among 12 different possibilities that would all make the negotiation succeed. However, not all of them may be necessarily equally desirable to her. In fact, Alice has several preferences concerning the disclosure of her credentials: for example she prefers to disclose her ID card number instead of her passport number and she prefers to provide her bank account instead of paying via credit card. This information is not given by her policies. Generally speaking, a user may be interested in always disclosing the less sensitive credential in case there is a choice. Moreover, personal preferences for certain credentials may play a role as well. These preferences act orthogonal to the actual policies: policies specify that access to resources is granted if certain conditions are satisfied but not how to decide which credential to disclose in case only k out of n satisfied conditions are required. Alice's trust agent would have to ask Alice to decide which of all the 12 alternatives she prefers to disclose. And as soon as complex policies come into play, she may be easily overloaded with too many options. Furthermore, many of these options are already overruled by others so the user does not even need to consider them. Therefore, Alice's preferences shall be exploited in order to rule out suboptimal negotiations. The following requirements sum up what has to be taken into account when providing a solution to Alice.

Partial order preferences. It may be difficult for Alice to define a total order preference for all her credentials. First, it is time consuming, and second, indifferences shall be supported because it may be impossible to say whether a frequent-flyer card is more or less sensitive than a video club card. Moreover, it is useless to specify such a preference since it is unlikely that they will be given as an alternative to each other. Therefore, it should be possible to reason over Alice's preferences even if only a partial ordering among her credentials is available.

	1	2	3	4	5	6	7	8	9	10	11
	name	bdate	telephone	e-mail	postcode	id	passport	bank name	bank account	credit card	pin
S_1	×	×		×				×	×		
S_2	×	×		×						×	×
S_3	×	×			×			×	×		
S_4	×	×			×					×	×
S_5						×		×	×		
S_6						×				×	×
S_7							×	×	×		
S_8							×			×	×
S_9	×					×		×	×		
S_{10}	×					×				×	×
S_{11}				×		×		×	×		
S_{12}				×		×				×	×

Table 3.1: The 12 possible disclosure sets $S_1 - S_{12}$ for a successful negotiation between the book store and Alice. The crosses indicate if a credential is contained in the corresponding set.

Preferences among more than one credential. Generally, preferences among disclosure sets of credentials (and not only among single credentials) should be allowed, too. For instance, it could be preferred to disclose the e-mail address instead of the date of birth together with the postal code (since postal code and date of birth are considered a quasi-identifier [130]).

Conditional preferences. Contrarily to this preference, in case the date of birth is not disclosed, Alice may strongly prefer to disclose her postal code instead of her e-mail address. However, if she also has to disclose her date of birth, she would switch her preference and prefer to disclose her e-mail instead of her post code because the latter together with her date of birth is a quasi-identifier. Even more general, preferences may depend on other situational attributes such as the party the user is negotiating with. Therefore, a preference-based approach should allow for conditional preferences such as "This preference only holds if my date of birth is disclosed, too. In all other cases, I have the opposite preference.".

Qualitative preferences. It may be clear that Alice considers her ID card less sensitive than her passport. However, quantifying the sensitivity of a credential is difficult (e.g., $s_{id} = 10, s_{passport} = 11$ or $s_{id} = 10, s_{passport} = 51$), especially when dealing with a large number of credentials. Furthermore, the aggregation of this quantification for sets of credentials is even more difficult: calculating the cost of disclosing *two* (or more) credentials using arbitrary quantitative aggregation methods is difficult to understand by users (assigning sensitivity 11 or 51 to $s_{passport}$ may have a great difference later on). Therefore, *qualitative* preferences among credentials should be allowed.

Selecting the optimum. Finally, any solution provided to Alice has to meet a trivial but very important requirement, i.e., to reduce the number of negotiations by strictly following the users preferences: any procedure should ensure that no preferred alternative is ruled out and no suboptimal disclosure set should be contained in the selected alternatives.

After identifying the problem and the requirements for preferences in a policy-based Trust Negotiation, the following section will explain an extension to preference theory (as it has been detailed in Section 2.2) which will be needed to completely meet the scenario's requirements.

3.1.2 A Preference Model for Trust Negotiation

In what follows, the preference theory introduced so far will be extended in order to handle credential disclosure sets and to find the most preferred negotiations out of the set of all possible negotiations. For this, a model for credential disclosure will be described and a model for the various types of preferences involved. This model will be based on qualitative preferences defined over single credentials (such as "I prefer to give my bank account information. My credit card number would be the second choice.") or over sets of credentials (such as "Giving my e-mail is preferred to disclosing my postal code together with my date of birth.").

Modeling credential disclosure sets

Let $C = \{c_1, \ldots, c_n\}$ be the set of credentials a party of a negotiation owns. The set of credentials a party has to disclose during the whole negotiation in order to succeed is a subset of C. Following the representation in Table 3.1, a set of credentials is represented as a bit vector with n dimensions comprising one dimension for each single credential such that setting a bit i to 1 means that during the negotiation the credential c_i is disclosed.

Definition 3.1 (Credential Disclosure Vector). *Let S be a credential disclosure set over the set of credentials C. The Credential Disclosure Vector representing S is the bit vector $X = (x_1, \ldots, x_n)$ ($n = |C|$) such that $x_i = 1$ iff $c_i \in S$ and $x_i = 0$ otherwise.*

Example 3.1. In the scenario, the set of credentials Alice owns is $C = \{c_{name}, c_{bdate}, c_{phone}, c_{email}, c_{pcode}, c_{id}, c_{passport}, c_{bank\_name}, c_{bank\_account}, c_{cc}, c_{pin}\}$. Mapping this set into a vector allows one to easily represent the disclosure set S_1 from Table 3.1 as $(1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0)$. In the following, this order (as it is also depicted in Table 3.1) will be assumed for the following examples. $\diamond$

These bit vectors represent objects (credential disclosure sets) for which each attribute dimension is the credential name and only two possible values exist: either 0 (a certain credential is not disclosed) or 1 (a credential is disclosed). In the following, credential disclosure sets and their bit representation will be used interchangeably.

Modeling preferences

As it has been shown so far, preference relations act on two different levels: on the object level and on the attribute level (cf. Definitions 2.2 and 2.1). This section introduces preferences on both levels as well as other preference types needed for comparing disclosure sets.

Object level preferences. Preference relations on the object level act among disclosure sets or, more precisely, on their bit vectors. These preferences are computed out of attribute level preferences given by the user. Since object level preferences cannot be easily defined by the user, preferences on the attribute level are used to build up preferences on object level.

Attribute level preferences. Preferences on attribute level act among credentials, that is, whether the disclosure of one credential is preferred or not. Privacy plays a main role in Trust Negotiations and it may be assumed that a user always prefers "not to disclose" a credential in a negotiation. Therefore, in the running scenario the attribute level preference $0 \succ_i 1$ is assumed for each credential c_i. However, a user may want to specify the opposite preference for some credentials in order to force the negotiation to select a negotiation path in which a specific credential is disclosed.[1] Therefore, this theory allows for attribute level preferences in both directions.

The composition of the attribute level preferences allows to compare disclosure sets on the object level. But the two composition paradigms Pareto and lexicographic composition (see page 29 in Section 2.2.2) form the extreme cases of possible compositions: whereas a lexicographic order adheres to a strict ranking between the preferences, the Pareto composition assumes no order at all.

[1]This may be the case for vouchers or discount credentials that may allow the user to receive a discount or even a free purchase when performing a transaction.

However, for the application in Trust Negotiation both paradigms are problematic: by focusing on the highly preferred attributes the lexicographic order biases towards negotiations that will *not* disclose a very sensitive credential, even if they disclose all other credentials. Given the fact that the set of credentials disclosed should be kept rather small this is definitely not a desirable behaviour. The Pareto composition on the other hand is too careful: by considering the disclosure of each credential as equally problematic, a lot of incomparability between different negotiations is introduced and the user has to choose between loads of possible negotiations. In fact, the result sets of Pareto compositions are known to grow exponentially with the number of dimensions (here: the number of possible credentials) [20]. The solution to reduce the amount of incomparable disclosure sets is achieved by amalgamated preferences: it is possible for the user to specify a preference order over the attributes themselves and thus distinguish between more or less preferred (i.e., sensitive) credentials.

Amalgamated preferences. In the presented Trust Negotiation setting, amalgamated preferences connect two credentials with a preference relation. In Figure 2.4 on page 27, Alice's amalgamated preferences are represented in a graphical manner.[2] Amalgamated preferences in the considered setting represent the fact that a disclosure set can be more or less preferred depending on *which* credentials are contained in this disclosure set (and not only *whether* one credential is disclosed or not, which is considered in the comparison following the Pareto composition). In order to better understand the concept of amalgamation (given in Definition 2.5), a detailed example in the context of Trust Negotiation is provided as follows.

Example 3.2. Alice may state that a negotiation where she has to disclose her credit card and not her bank account is less preferred than a negotiation where she does not disclose her credit card but her bank account. Mind that these two disclosure sets are incomparable from the Pareto composition point of view because in the bank account dimension the first is better (it does not include the disclosure of the bank account and the other does) but in the credit card dimension the second is better (the second set does not contain the credit card). Hence, Alice relaxed her preference for not disclosing her bank account instead of disclosing it in favor of the fact that the credit card is not disclosed.

The formal specification of this amalgamated preference relation following Definition 2.5 looks as follows. The numbers of the different dimensions in μ rely on the order in Table 3.1 (as it is done for the vectors representing the credential

[2]The representation in Figure 2.4 is a way to present the amalgamated preferences of the scenario in a reader-friendly way. However, for amalgamated preferences over more than two dimensions this kind of representations may become more difficult to understand. The reader is pointed to [13] for further details about this issue.

disclosure sets) starting with 1. Hence, the bank account's dimension is 9, the credit card's dimension is 10, and the amalgamated preference relation is denoted as $\succ_{\{9,10\}}^{(1,0),(0,1)}$. This relation amalgamates the two dimensions 9 and 10 and allows to decide between two negotiations where in one the credit card is disclosed but the bank account is not and in the other the bank account is disclosed but the credit card is not; according to the ceteris paribus condition, all dimensions except the amalgamated ones 9 and 10 have to show equal values in both disclosure sets to be compared. $\diamond$

Conditional preferences. The notion of amalgamated preferences also allows for conditional preferences in the context of Trust Negotiation. In Figure 2.4, conditional preferences are depicted as dotted arrows with a condition attached. Alice's preference concerning post code and email depends on whether the date of birth is additionally disclosed or not. For each value of the condition, a new amalgamated preference is introduced as follows. For the case where date of birth is disclosed $\succ_{\{2,4,5\}}^{(1,1,0),(1,0,1)}$ is introduced and for the other case $\succ_{\{2,4,5\}}^{(0,0,1),(0,1,0)}$. This example solves the quasi-identifier problem presented in Section 3.1.1. However, conditions may be even more complex: they may be situational [74] and therefore include the external context of a negotiation. For example, it may play a role whom one is negotiating with (e.g., one prefers to disclose the bank account to the credit card number if the requester is the bank and vice versa otherwise). These kinds of conditions can easily be modeled with the presented framework as additional dimensions in the vectors to be compared.

Possible conflicts. As soon as one considers more than one preference in order to build up a concise knowledge base of all the user's preferences, one has to resolve possible conflicts between the given preferences. This is because a contradicting preference relation may lead to cycles in the object level preference and therefore does not allow for concise comparison anymore: finding the optimal object in a given set of objects becomes non-deterministic.

Example 3.3. One example could be that two amalgamations given by the user directly contradict, such as $\succ_{\{2,3\}}^{(1,0),(0,1)}$ and $\succ_{\{2,3\}}^{(0,1),(1,0)}$. $\diamond$

But even in cases where amalgamated preferences do not directly contradict, they may still conflict when considering a transitive chain as in the following example.

Example 3.4. Assume there already exists one amalgamated preference $\succ_{\{2,3\}}^{(1,0),(0,1)}$. Adding the amalgamation $\succ_{\{1,2,3\}}^{(1,0,1),(0,1,0)}$ leads to an indirect contradiction: $(0,1,0) \succ_{\{2,3\}}^{(1,0),(0,1)} (0,0,1) \succ_1 (1,0,1)$ holds but this directly contradicts the amalgamation to be added which states the opposite: $(1,0,1) \succ_{\{1,2,3\}}^{(1,0,1),(0,1,0)} (0,1,0)$. $\diamond$

In order to avoid possible conflicts in a set of preferences, a preference to be added to this set has to meet certain conditions:

Definition 3.2 (Consistent Preferences). *Let O be a set of objects and $P \subseteq O^2$ a preference relation on these objects. Let further P^{conv} be the converse relation wrt. to P such that $(x, y) \in P \leftrightarrow (y, x) \in P^{conv}$. A preference relation $S \subseteq O^2$ is called consistent wrt. P iff*

1. *$\forall x, y \in O : (x, y) \in S \rightarrow (y, x) \notin S$ and*
2. *$S \cap (P \cup P^{conv}) = \emptyset$.*

It will become obvious later (see Remark 3.2) that the first condition in this definition takes care of cases like the one in Example 3.3 and the second condition corresponds to Example 3.4.

Combining preferences transitively. Based on this condition, one is able to consistently add preferences to a knowledge base and incrementally build up one single preference relation which is called Incremented Preference Relation in the following. This relation includes the transitive closure of the preferences incrementally added:

Definition 3.3 (Incremented Preference Relation). *Let O be a set of objects, $P \subseteq O^2$ a relation on these objects, and $S \subseteq O^2$ the set of object pairs representing the preference to be added to P. Let further be S consistent wrt. P and let T be the transitive closure $T := (P \cup S)^+$. The Incremented Preference Relation of P incremented by S is defined as $P^* := \{(x, y) \in T | (y, x) \notin T\}$.*

3.1.3 Filtering out Non-Preferred Disclosure Sets

This section explains how the theoretical basis such as Pareto composition, amalgamated preferences, and incremented preferences is used to select the non-dominated objects, i.e., the most preferred disclosure sets according to the given preferences. At the end of this section the preference relation $\succ\!\succ$ will be defined which allows to compare any two disclosure sets according to a set of preferences given by a user. Based on this relation, a formal definition of the set of optimal disclosure sets will be given.

The following example shows how Pareto dominance helps to rule out non-preferred disclosure sets in the running scenario:

Example 3.5. Given the two disclosure sets S_6 and S_{10} from Table 3.1, $S_6 = (0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1)$ and $S_{10} = (1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1)$, and the set of attribute level preference relations over c_i $\{0 \succ_{c_{name}} 1, \ldots, 0 \succ_{c_{pin}} 1\}$ it is obvious to infer that $S_6 \succ S_{10}$ holds since $S_6 \succ_1 S_{10}$ and $S_6 =_i S_{10}$ for all $i \neq 1$. In any case, it is possible to automatically infer that a user always prefers to disclose only a subset of credentials, i.e., only ID card number and bank account. $\diamond$

Remark 3.1. *It is important to point out that the general preference of a subset of credentials only holds since the preference* $0 \succ_i 1$ *is assumed on attribute level, i.e., not disclosing a credential* c_i *is always preferred to disclosing it. Given these preferences over the binary value space for attributes, computation of Pareto domination is reduced to simple set containment checks.*

The following example shows how taking the amalgamated preferences into account eliminates additional dominated objects:

Example 3.6. Given the sets (from Table 3.1) $S_5 = (0,0,0,0,0,1,0,1,1,0,0)$ and $S_7 = (0,0,0,0,0,0,1,1,1,0,0)$, and the amalgamated preference $\succ^{(1,0),(0,1)}_{\{6,7\}}$ (that is, it is preferred to disclose the ID card number to the passport number) it is possible to infer that $S_5 \succ S_7$. Therefore, it is possible to automatically infer that the user would prefer to disclose her ID *and* bank information instead of disclosing her passport *and* bank information. $\diamond$

Pareto composition (in Example 3.5) and preference amalgamation (in Example 3.6) applied in isolation to filter out dominated disclosure sets is not sufficient in some cases—as it is shown in the following example.

Example 3.7. S_5 and S_6 cannot be compared with the mechanisms given so far: there is no way to combine Alice's amalgamated preference concerning bank name and credit card and her amalgamated preference concerning bank account and pin without utilizing a transitive application of two or more preferences. $\diamond$

This example motivates the exploitation of incremented preferences (see Definition 3.3) in order to further reduce the number of disclosure sets. According to Definition 2.5, only one amalgamated preference is applied at a time; for the remaining dimensions the ceteris paribus condition fires. However, it may be possible that several preferences—be it simple attribute level preferences or amalgamated ones—need to be applied over the same two disclosure sets in order to compare them. Therefore, the transitive combination of preferences is exploited. This combination is provided by the incremented preference relation. The following example shows how an incremented preference relation is used to compare the two disclosure sets from Example 3.7.

Example 3.8. In order to compare S_5 and S_6 one needs to combine two of Alice's preferences; for example her preference saying that bank name is preferred to credit card ($p_1 = \succ^{(1,0),(0,1)}_{\{8,10\}}$) and her preference saying that bank account is preferred to pin ($p_2 = \succ_{\{9,11\}}^{(1,0),(0,1)}$). To achieve this combination the definition of an incremental preference relation (see Definition 3.3) is applied in the following way. First, p_1 is added to an (initially empty) incremented preference P_0^*. From the definition of amalgamated preferences (Definition 2.5) and from the fact that P_0^* is empty, it is obvious that p_1 is consistent wrt. P_0^*. Adding p_1 yields the incremented preference $P_1^* = p_1$. In the second step, p_2 is added to P_1^* in order to construct the incremental preference P_2^*. Similarly to the first step, p_2 is consistent wrt. P_1^*. This is due to the ceteris paribus condition in the definition of amalgamated preferences: for any pair in p_1 the dimensions 9 and 10 are equal where for any pair in p_2 they are different. Therefore, no pair in p_1 contradicts a pair in p_2. By applying the transitive closure in order to construct P_2^* the pair (S_5, S_6) is introduced as an element of P_2^*. This is due to the transitive chain built from the following two pairs: p_1 contains the pair $(S_5, (0,0,0,0,0,1,0,0,1,1,0))$ and p_2 contains $((0,0,0,0,0,1,0,0,1,1,0), S_6)$. By transitively combining p_1 and p_2 to P_2^* one gets the desired relation $(S_5, S_6) \in P_2^*$. $\diamond$

This example forms a base case and provides evidences that the extension of an isolated application of the Pareto domination and the amalgamated preferences given by the user is needed. In order to provide *all possible combinations* of all given preferences, it is required to build up an incremented preference relation called Complete Preference Relation $\succ\!\succ$ which is defined as follows.

Definition 3.4 (Complete Preference Relation). *Let $\succ_P$ be the Pareto composition of attribute level preferences $\succ_1, \ldots, \succ_n$. Let further $P = \{p_1, \ldots, p_m\}$ be a set of amalgamated preferences. The Complete Preference Relation $\succ\!\succ$ is defined as the incremented preference relation of $\succ_P$ incremented by $\bigcup P$.*

Remark 3.2. *Because of the conditions in Definitions 3.2 and 3.3, this Definition covers two implicit requirements:*

1. *Each amalgamated preferences must not contradict another, i.e., $p_n \cap p_m^{conv}$ must be empty for all n, m. This case matches Example 3.3 and Condition 1 in Definition 3.2: it implies that the framework requires user-given amalgamated preferences to not directly contradict each other.*

2. *The union of the amalgamated preferences has to be consistent wrt. the Pareto composition. This restriction ensures that the incremented preference relation does not contain any cycles [12]. Instantiated for this thesis' scenario, this restriction requires that no amalgamated preference stated by the user contradicts the Pareto domination (i.e., each amalgamated preference has to be consistent wrt. the Pareto composition of the attribute level preferences). This requirement corresponds to Example 3.4 and to Condition 2 in Definition 3.2.*

Further, these two requirements imply that adding new preferences to the incremented preference relation is a monotonic process [12]: adding new preferences will never make former comparisons invalid, it will always add comparable pairs and never remove some. This is in particular helpful for an incremental preference elicitation process: after adding new preferences provided by the user it is not needed to recompute the whole set of optimal disclosure sets. Instead, one can simply remove those objects from the disclosure set which are now dominated.

The incremented preference relation forms the basis for the intended object level preference relation among disclosure sets. From its definition it is clear that it comprises the Pareto dominance relation as well all single amalgamated preference relations given by the user. It additionally contains all combinations of these base preferences. Therefore, it filters out all the disclosure sets which are not preferred:

Definition 3.5 (Optimal Disclosure Sets). *Let O be a set of disclosure sets and $\succ\!\succ$ a complete preference relation. The set of optimal disclosure sets $O_{\succ\succ}$ is defined as follows: $O_{\succ\succ} := \{o \in O \mid \nexists o' \in O : o' \succ\!\succ o\}$.*

In the following section the Trust Negotiation scenario from Section 2.1.3 will be revisited in detail and it will be shown how undesired disclosure sets are ruled out in the framework presented so far.

3.1.4 Revisiting the Scenario

In the following, the techniques and concepts defined in previous sections will be exemplified by applying them to the disclosure sets $S_1, \ldots, S_{12}$ which all yield a successful negotiation. This is to find out the optimal negotiations for Alice given these sets.

As it is shown above, the incremented preference relation contains the Pareto composed attribute level preferences as well as the amalgamated preferences. However, in order to see the improvement of each preference concept introduced in the previous section, the process of ruling out dominated disclosure sets is divided into three steps: it is shown

(A) how objects are ruled out by Pareto composition,

(B) how objects are ruled out by single amalgamated preferences, and

(C) how the transitive combination in the incremented preference relation rules out the remaining dominated objects.

Pareto composition. Using the attribute level preferences $0 \succ_i 1$ which always prefers not disclosing a credential (cf. considerations on page 42 in Section 3.1.2), Pareto composition can be applied to remove dominated sets. From Pareto domination one can conclude that disclosure set S_{10} can be removed since it is dominated by the set S_6 (all dimensions are equally good in S_6 and in S_{10} except dimension 1 (c_{name}) in which S_6 is preferred to S_{10}). This may be considered straightforward since S_{10} additionally requires Alice to disclose c_{name} which is an unnecessary disclosure. In addition, also S_9 can be removed since it is dominated by S_5. Furthermore, S_6 Pareto dominates S_{12} and S_5 Pareto dominates S_{11}. Hence, Pareto composition alone is already able to filter out four dominated disclosure sets, namely, S_9, S_{10}, S_{11}, and S_{12}.

Amalgamated preferences. In addition to Pareto composition, the amalgamated preferences specified by Alice are exploited to further reduce the number of disclosure sets. From Alice's preference for disclosing her ID card number instead of her passport number (amalgamated preference $\succ_{\{6,7\}}^{(1,0),(0,1)}$), one can conclude that S_7 is dominated by S_5 and that S_8 is dominated by S_6. Furthermore, Alice has an amalgamated preference over three dimensions because of her fear of being quasi-identified: $\succ_{\{2,4,5\}}^{(1,1,0),(1,0,1)}$. From this preference one can infer that S_2 dominates S_4 and S_1 dominates S_3. This way it is possible to remove yet another four dominated disclosure sets, namely S_3, S_4, S_7, and S_8.

Transitive combination of preferences. After having performed the previous steps, the remaining disclosure sets are S_1, S_2, S_5, and S_6. Among the inferred preferences, there is the preference transitively combined out of the preferences 'bank name is preferred to credit card' ($\succ_{\{8,10\}}^{(1,0),(0,1)}$) and 'bank account is preferred to pin' ($\succ_{\{9,11\}}^{(1,0),(0,1)}$). Applying both preferences transitively enables us to rule out S_2 because it is dominated by S_1 and S_6 because it is dominated by S_5 (as it is shown in Example 3.8).

The procedure described in this section is able to filter out ten non-preferred credential disclosure sets based on qualitative preferences, therefore facilitating Alice's interaction with her negotiation agent. However, two disclosure sets are still remaining: S_1 and S_5. Both are not comparable according to the specified preferences.

3.1.5 Implementation and Experiments

A prototype has been implemented computing preferred credential disclosure sets given a set of successful negotiation paths and a set of preferences specifying which credential's disclosure is preferred to another's. The user interface is web-based and the logical core of the approach is implemented in Prolog. Given a set of credentials a connected policy engine provides all disclosure sets yielding a successful negotiation. From this set the Prolog engine computes the non-dominated and therefore preferred disclosure sets. In case there exists one single non-dominated disclosure set, this one will be used automatically. Otherwise, the user has to choose one disclosure set.

The efficiency of the presented preference approach was tested along two dimensions:

- in terms of how many disclosure sets are ruled out and
- in terms of execution time.

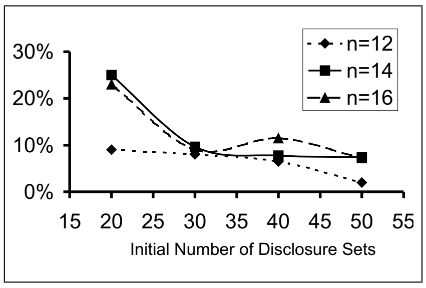
The first number is crucial since the approach is only effective if a considerable set of suboptimal negotiations is actually ruled out. The second number gives evidences about the applicability in a realistic scenario. Since there is no Trust Negotiation benchmark available which may serve for experiments, nor any available real data about disclosure sets (due to high sensitivity even if anonymized), generated data was used as follows.

In a Trust Negotiation, typically only one client credential of a certain type is needed. E.g., it is either required to disclose the passport number or the ID card number, rather than both. Therefore the set of client credentials C was partitioned into k disjoint subsets T called *credential types*. Further, it was assumed that for the success of a negotiation exactly one credential for each credential type has to be disclosed. Following this, it is obvious that each type should at least contain two credentials—otherwise this single credential will be contained in each disclosure set and will be disregarded in the comparison process. Further, for each credential type a totally ordered preference was assumed on the client side.

For each run of the experiment the following steps were performed:

1. randomly create k credential types $T_1, \ldots, T_k$ for the n credentials,
2. create a set of amalgamations such that for each type T_i the relation $\succ_{j,j+1}^{(1,0),(0,1)}$ is added for all $c_j, c_{j+1} \in T_i$,
3. create a set O of n disclosure sets such that each disclosure set contains one randomly chosen credential for each credential type,
4. rule out all dominated disclosure sets following the definition of $\succ\!\succ$ (Definition 3.4).

In the setting of the experiment parameters of the following dimensions were alternated:



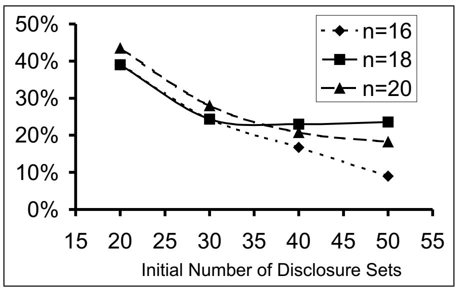


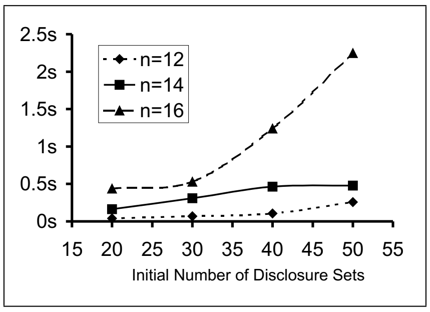
Figure 3.1: Optimal disclosure sets for $k = 5$ (left) and $k = 7$ (right). n is the number of credentials and k is the maximal number of credentials that can be disclosed during a negotiation.

Number of credentials n. The higher the number of credentials the higher the number of dimensions of the vectors to be compared. Increasing this value yields an increasing of incomparable pairs in the set of disclosure sets. For the experiments the number of credentials was chosen to be between 12 and 20.

Number of credential types k. This number actually determines how many credentials are definitely disclosed in each negotiation. Increasing this value also decreases the amount of suboptimal objects because the probability that an object is 'bad' in one credential type but 'good' in another becomes higher which leads to more incomparable pairs. In the experiments, 5 and 7 credential types were selected to cover negotiations where exactly 5 or 7 credentials *have* to be exchanged in each negotiation.

Number of disclosure sets. The number of disclosure sets represents how many different negotiation paths exist and would have been shown to the user for selection, in case no preference filtering was applied.

Several experiments were performed with varying values, 20 runs for each setting; see the graphs in Figure 3.1 for the results of some representative settings. It turned out that the average percentage of optimal disclosure sets is 18.3%. Hence, on average 81.7% of the alternatives a user has to chose from are ruled out. This is a huge improvement: instead of, e.g., 30 disclosure sets with obvious suboptimal alternatives only 6 sets are shown to the user. Moreover, removing the 24 dominated possibilities is not only important from a usability perspective—it is also suggestive from a privacy point of view: if the user selected one of the suboptimal negotiations, she would definitely disclose more sensitive information which she would not have to disclose to make the negotiation succeed.



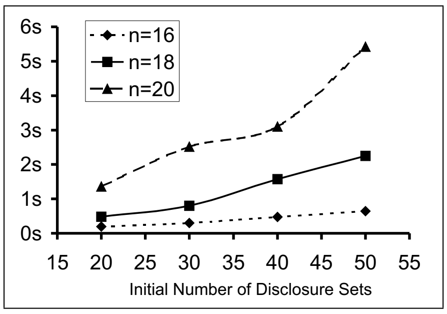


Figure 3.2: Computation Time for $k = 5$ (left) and $k = 7$ (right). n is the number of credentials and k is the maximal number of credentials that can be disclosed during a negotiation.

These results show that for a relatively big number of possible disclosure sets (a higher number will rarely occur in reality), the presented approach filters out a considerable subset of suboptimal alternatives and therefore reduces the work load for the user. Furthermore, it turned out that for the experimental setting the average computation time was about 2.3 seconds (see Figure 3.2). The implementation does not consider any of the numerous optimization strategies available for skyline computation (e.g., [106]) which would make the computation even faster.

Complexity

The complexity of qualitative preference composition is quadratic in the worst case. While this may be an issue in large database systems with million records, it is typically acceptable for the analysis of credential disclosure sets since the total number of successful negotiations is considerably small compared to databases. It is also important to note that it is only restricted to the preferences concerning the credentials in successful negotiation paths. The time the comparison of two objects takes depends on the amount of amalgamations given. However, this amount is also not too big and acts as a constant value that does not dramatically increase complexity.

Discussion

The modeling of Trust Negotiation in the presented experiments has several simplifications compared to a real scenario. Redundant policies as they are introduced in the book store scenario were not considered, hence, Pareto domination was never applied in the experiment which would even lead to a higher number of suboptimal disclosure sets.

Further, credential types are assumed to be independent. In real scenarios, as soon as one credential of a certain type is disclosed it is often not needed to disclose a credential of another type (e.g., as soon as the passport is provided it is not needed to additionally disclose name, address, or date of birth because this information is already contained in the passport credential). These kinds of dependencies were not considered in the experiment's setting.

Furthermore, the experiment does not reflect preferences among different credential types, which one can assume in a realistic setting and therefore more objects would be ruled out according to these preferences. Finally, the client's preferences are considered a total order for each credential type. As is has been motivated, this is also rarely the case since preferences are typically partially ordered.

The selection of credential disclosure sets in Trust Negotiations is a difficult task. Only allowing for total orders over the set of owned credentials or linear aggregation (as it is the case for existing approaches, cf. related work in Section 5.1) is undesirable. In this section an approach has been presented based on qualitative partial order preferences including preference composition, amalgamated preferences, and conditional preferences. It was further demonstrated with the help of the example scenario how this approach can be used to solve the problem of finding optimal disclosure sets. This solution eases the task of selecting among possibly many alternatives in a Trust Negotiation by exploiting the user's preferences. Although the preference framework is applied in the realm of Trust Negotiation, this preference scheme can be used in any other domain where objects need to be compared or selected based on preferences specified on the objects' attributes.

With this framework at hand, it is still an open question, how preference expressions can be placed inside a policy such that policy evaluation interferes with computing preferred solutions. The following section provides a logic framework that allows to incorporate preference expressions inside a policy language.

3.2 Partial Order Preferences for Rule-Based Policies

Representing semantic policies by means of rules is known as the most straightforward way of turning a human's intention of a system's behaviour into a computer-understandable set of instructions [36]. As far as the semantics of rules is concerned, policy languages—in particular policy languages supporting Trust Negotiation—apply the standard Logic Programming semantics for stratified programs where stable model semantics and well-founded semantics coincide [62].

To this end, in this section, the stable model semantics of Logic Programs will be extended to cater for partial order preferences to be expressed between literals in heads of rules. It is important to note that the application of this extension is not limited to the expression of policies but serves as a logical framework to express any kind of Logic Program with partial order preferences among literals. In the following, DLPOD (Disjunctive Logic Programs with Ordered Disjunction) will be presented. DLPOD is a Logic Programming language which extends the Answer Set paradigm with means to express *partial* order preferences. For this, the LPOD approach of Brewka et at. [44] (see page 33 in Section 2.2.3) will be combined with general disjunctions. Roughly speaking, DLPOD allows both operators in rule heads:

- ordered disjunctions indicated by the operator $\times$ as they are used in LPODs and
- normal disjunctions indicated by the operator $\vee$ known from standard Disjunctive Logic Programs (DLPs).

Thus, in DLPOD it is allowed to define rules like[3]

$$c_{name} \times (c_{telephone} \vee c_{pcode}) \leftarrow not\ c_{osc}. \tag{3.1}$$

By the combination of the two operators, an indifference can be introduced between the two options $c_{telephone}$ and c_{pcode}. This policy states, intuitively, in case the body of the rule is true (i.e., the OSC credential is not provided), the user prefers c_{name} to be true, that is, to be contained in the answer set and thus disclosed to a negotiation partner. If c_{name} can not be satisfied (e.g., another policy rule protects the disclosure of c_{name}), it is considered equal if either of the options $c_{telephone}$ and c_{pcode} are true. Consequently, both options, disclosing the telephone number and disclosing the postal code are equally preferred thus forming two answer sets. In the following, first a detailed definition of the syntax of DLPOD rules will be provided and second, their semantics will be defined.

3.2.1 Syntax and Semantics of DLPODs

The syntax of DLPODs straightforwardly extends Logic Programs with Ordered Disjunction from [44] by the common disjunction $\vee$ used in Disjunctive Logic Programming. To this end, the syntactic constructs allowed in the head of rules is extended as follows.

Definition 3.6 (Ordered Disjunctive Term). *An ordered disjunctive term is a (possibly nested) term of literals $C_1, \ldots, C_n$ connected by $\vee$ or $\times$. Such terms are recursively defined as follows.*

[3]In this section, the preferences and policies introduced as a scenario in Section 2.1.3 and Figure 2.4 are used as basis for examples. However, for the sake of overview, only parts of those policies and preferences are reused in this section.

- *Any literal l is an Ordered Disjunctive Term.*
- *If F and G are Ordered Disjunctive Terms, then $(F \times G)$ and $(F \vee G)$ are Ordered Disjunctive Terms as well.*

DLPOD programs are defined as extended Logic Programs allowing rules with Ordered Disjunctive Terms in the heads:

Definition 3.7 (Disj. Log. Program with Ordered Disjunction). *A Disjunctive Logic Program with Ordered Disjunction (DLPOD) P is a set of rules of the form*

$$r = Head_r \leftarrow Body_r.$$

where $Body_r = B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_k$ such that all B_i $(1 \leq i \leq k)$ are literals and $Head_r$ is an Ordered Disjunctive Term. Further

- *$Body^+(r)$ is defined as $\{B_1, \ldots, B_m\}$ and*
- *$Body^-(r)$ as $\{B_{m+1}, \ldots, B_k\}$.*

Defining the semantics of DLPODs follows a two steps approach. First, the potential optimal answer sets (or candidate answer sets) are defined. Subsequently, a preference relation is defined upon those candidate answer sets (acting as an object-level preference, cf. Definition 2.1) in order to identify the optimal answer sets. For the potential answer sets of DLPODs, a semantics based on split programs is provided in the following section.

Split programs of a DLPOD

The answer sets of a DLPOD are defined based on an extended notion of split programs as they are introduced in [44]. For defining split programs of a DLPOD, first the *Ordered Disjunctive Normal Form (ODNF)* of a rule is defined. Then, it is shown how to transform each rule's head into this normal form. Based on rules given in this normal form and on the definition of the *option* of such a rule, split programs of a DLPOD will be defined.

Definition 3.8 (Ordered Disjunctive Normal Form (ODNF)). *The Ordered Disjunctive Normal Form of an Ordered Disjunctive Term is*

$$\bigvee_{i=1}^{n} \bigtimes_{j=1}^{m_i} C_{i,j} = (C_{1,1} \times \ldots \times C_{1,m_1}) \vee \ldots \vee (C_{n,1} \times \ldots \times C_{n,m_n})$$

Further, $(C_{i,1} \times \ldots \times C_{i,k_i})$ is called the i-th Ordered Disjunct *of the ODNF and a rule r is defined to be in ODNF if $Head(r)$ is in ODNF.*

Nested DLPOD rules are treated arbitrarily as shorthand for DLPOD rules in ODNF. I.e., given an ordered disjunctive term S and subterms a, b and c of S the following rewriting rules can be used to expand a nested rule into ODNF:

$$a \times (b \vee c) \Rightarrow (a \times b) \vee (a \times c) \tag{3.2}$$

$$(a \vee b) \times c \Rightarrow (a \times c) \vee (b \times c) \tag{3.3}$$

$$(a \times b) \times c \Rightarrow a \times b \times c \tag{3.4}$$

$$a \times (b \times c) \Rightarrow a \times b \times c \tag{3.5}$$

Example 3.9. By exhaustive application of these rules, any rule in a program can be transformed into ODNF. For instance

$$c_{name} \times (c_{telephone} \vee c_{pcode}) \leftarrow not\ c_{osc}. \tag{3.6}$$

yields the following rule in ODNF:

$$(c_{name} \times c_{telephone}) \vee (c_{name} \times c_{pcode}) \leftarrow not\ c_{osc}. \tag{3.7}$$

◇

Using the rewriting rules (3.2)–(3.5), hereafter, the semantics of a DLPOD P will be defined in terms of rules in ODNF only.

In what follows the definition of the split programs of a DLPOD P will be given. Before that, the notion of a rule's *option* has to be defined, since, intuitively speaking, a split program denotes the combinations of all options of each rule:

Definition 3.9 (Option of a rule). *Let r be a DLPOD rule in ODNF:*

$$\bigvee_{i=1}^{n} \bigtimes_{j=1}^{m_i} C_{i,j} \leftarrow body.$$

where m_i is the number of literals in the i-th Ordered Disjunct of r. An option *of r is any rule of the form ($j_i \leq m_i$):*

$$\begin{aligned} C_{1,j_1} \vee C_{2,j_2} \vee \ldots \vee C_{n,j_n} \leftarrow\ & body, \\ & not\ C_{1,1}, not\ C_{1,2}, \ldots, not\ C_{1,j_1-1}, \\ & not\ C_{2,1}, not\ C_{2,2}, \ldots, not\ C_{2,j_2-1}, \\ & \ldots \\ & not\ C_{n,1}, not\ C_{i,2}, \ldots, not\ C_{n,j_n-1}. \end{aligned}$$

Example 3.10. The ODNF rule (3.7) has the following four options (for illustration purposes repeated atoms are not removed in the example):

$$\begin{aligned} c_{name} \vee c_{name} &\leftarrow not\ c_{osc}. \\ c_{name} \vee c_{pcode} &\leftarrow not\ c_{osc}, not\ c_{name}. \\ c_{telephone} \vee c_{name} &\leftarrow not\ c_{osc}, not\ c_{name}. \\ c_{telephone} \vee c_{pcode} &\leftarrow not\ c_{osc}, not\ c_{name}, not\ c_{name}. \end{aligned}$$

◇

Definition 3.10 (Split program of a DLPOD). *A split program P' of a DLPOD P is obtained by replacing each rule in P by one of its options.*

It is important to note that the split programs of DLPODs are *disjunctive* Logic Programs whereas split programs of classical LPODs (as defined in [44]) are normal Logic Programs which are not disjunctive.

Example 3.11. Given the DLPOD $P =$

$$\begin{aligned} c_{name} \times (c_{telephone} \vee c_{pcode}) &\leftarrow not\ c_{osc}. \\ c_{id} \vee c_{credit\_card} &\leftarrow c_{osc}. \end{aligned}$$

the following four programs are split programs of P :

1. $c_{telephone} \vee c_{pcode} \leftarrow not\ c_{osc}, not\ c_{name}.$
   $c_{id} \vee c_{credit\_card} \leftarrow c_{osc}.$
2. $c_{telephone} \vee c_{name} \leftarrow not\ c_{osc}, not\ c_{name}.$
   $c_{id} \vee c_{credit\_card} \leftarrow c_{osc}.$
3. $c_{name} \leftarrow not\ c_{osc}.$
   $c_{id} \vee c_{credit\_card} \leftarrow c_{osc}.$
4. $c_{name} \vee c_{pcode} \leftarrow not\ c_{osc}, not\ c_{name}.$
   $c_{id} \vee c_{credit\_card} \leftarrow c_{osc}.$

◇

To estimate upper bounds in the complexity attribution of DLPOD, it will be needed to have a notion for head-cycle free DLPODs. Analogously to Disjunctive Logic Programs, head-cycle-freeness [19] for DLPODs is defined as follows:

Definition 3.11 (Head-Cycle Free DLPODs). *Let a DLPOD's dependency graph be defined as in Definition 2.7. A DLPOD P is head-cycle free if its dependency graph does not contain directed cycles that go through two literals occurring in two ordered disjuncts C_i and C_j $(i \neq j)$ of the same rule head.*

The following observation can be easily verified:

Proposition 3.1. *Split programs of head-cycle free DLPODs are head-cycle free.*

Similar to LPODs, the possible optimal answer sets of a DLPOD are the answer sets of all split programs. In the following section it will be explained in detail which answer set is called optimal according to the original DLPOD.

Optimal candidate answer sets of a DLPOD

For the definition of the semantics of a DLPOD it has not yet been stated what is called a preferred answer set of a DLPOD. So far, the possible answer sets of a DLPOD have been defined as the answer sets of its split programs. This section will detail how to compare these possible answer sets in order to find the optimal ones; i.e., the most preferred answer sets according to the ordered and unordered disjunctions in the rules' heads. According to the definition of attribute and object-level preferences (see Definitions 2.1 and 2.2), the object-level preference among answer sets has to be deduced from the attribute level preferences acting on literal level expressed in the rules' heads.

First, the Satisfaction Degree Vector is introduced as a measure for how much an answer set satisfies a DLPOD rule:

Definition 3.12 (Satisfaction Degree Vector). *Let r be a DLPOD rule of the form*

$$r = \bigvee_{i=1}^{n} \bigtimes_{j=1}^{m_i} C_{i,j} \leftarrow A_1, ..., A_l, not\ B_1, ..., not\ B_k$$

and let S be a set of literals. The satisfaction degree vector *D of r in S is a vector of the form $D = (d_1, \ldots, d_n)$ representing degrees of satisfaction for each disjunct in r's head where each d_i is either a natural number or the constant ϵ. The dimensions of the Satisfaction Degree Vector are defined as follows:*

1. $D = (1, \ldots, 1)$ *if* $S \not\models r$ *or otherwise*

2. $d_i = \epsilon$ *if* $S \not\models C_{i,j}$ *for all* $1 \leq j \leq m_i$,

3. $d_i = min\{t | C_{i,t} \in S\}$.

The Satisfaction Degree Vector of r in S is denoted by $Deg_S(r)$.

Intuitively, in this definition, a penalty is assigned to each Ordered Disjunct thus representing how much the answer set satisfies it. For each rule, these penalties build up a vector of degree values with one dimension for each Ordered Disjunct. The degree ϵ is chosen for head disjuncts which do not overlap with S (cf. Condition *2*). With ϵ it is indicated that a particular disjunct does not tell anything about how much an answer set is preferred. Further, like in [44], the best satisfaction degree (i.e., the vector $(1, \ldots, 1)$) is assigned in case a rule's body is not satisfied (cf. Condition *1*): there is no reason to be dissatisfied if a rule does not apply for a particular answer set.

Example 3.12. Considering again the rule

$$c_{name} \times (c_{telephone} \vee c_{pcode}) \leftarrow not\ c_{osc}.$$

Since this rule has two Ordered Disjuncts, any Satisfaction Degree Vector has two dimensions. The sets of literals $\{c_{name}\}$ and $\{c_{osc}\}$ both satisfy this rule to degree $(1, 1)$ (applied Condition *3* and Condition *1*, respectively). The set $\{c_{telephone}\}$ satisfies r to degree $(2, \epsilon)$ and the set $\{c_{pcode}\}$ satisfies r to degree $(\epsilon, 2)$. ◇

Definition 3.13 (Preference acc. to a rule). *A set of literals S_1 is preferred to another set of literals S_2 according to a rule r (denoted $S_1 \succ_r S_2$) iff $Deg_{S_1}(r) = (d_1^1, \ldots, d_n^1)$ Pareto-dominates $Deg_{S_2}(r) = (d_1^2, \ldots, d_n^2)$. That is $\forall i\ d_i^1 \leq d_i^2 \wedge \exists i\ d_i^1 < d_i^2$.*

Intuitively, all dimensions in $Deg_{S_1}(r)$ are required to show a smaller or equal number than in $Deg_{S_2}(r)$ and in at least one dimension $Deg_{S_1}(r)$ has to show a strictly smaller number than $Deg_{S_2}(r)$ (cf. the principle of Pareto domination in Definition 2.4). The constant ϵ plays the role of a placeholder which is not comparable to any number except that it is equal to itself. As it will become obvious in Section 3.2.2, this ϵ provides the "incomparability" needed to capture partial orders.

Now, finally the preference notion is extended to a relation comparing sets of literals (answer sets) according to a whole DLPOD:

Definition 3.14 (Preference acc. to a program). *A set of literals S_1 is preferred to another set of literals S_2 according to a set of rules $R = \{r_1, \ldots, r_n\}$ (denoted $S_1 \succ S_2$) iff $\exists i(S_1 \succ_{r_i} S_2) \wedge \neg\exists j(S_2 \succ_{r_j} S_1)$.*

The conditions in both definitions follow again the fair principle of Pareto optimality (cf. Definition 2.4): an object is preferred if it is better or equal to another in all attributes (in the presented case in all Ordered Disjuncts or in all rules, respectively) and strictly better in at least one attribute. Finally, a preferred answer set of a program P is defined as follows:

Definition 3.15 (Preferred Answer Set). *Given a DLPOD P, its set of split programs $\mathcal{P}$, and an answer set S of a split program in $\mathcal{P}$. S is called a preferred answer set (of P) if there is no answer set S' of any split program in $\mathcal{P}$ for which $S' \succ S$ holds.*

3.2.2 Encoding Partial Order Preferences into DLPODs

As hinted already earlier in this chapter, DLPOD-programs extend the approach of preferences in Logic Programming towards *partial* order preference relations. This section details how to actually model partial order preference relations with Ordered Disjunctive Terms. For this, a transformation of a partial order of literals into a Disjunctive Normal Form is specified yielding a partial order preference statement in a rule's head.

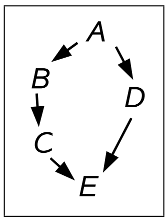
Definition 3.16 (Transformation of a Partial Order). *Given a Partial Order $<$ over a set of literals S and its corresponding covering relation $<_*$ (that is, $<_*$ contains the transitive reflexive reduction of $<$), the transformation P of $<$ into an Ordered Disjunctive Term is defined as: $P(<, S) = \bigvee_{j=1}^{n}(C_{j,1} \times \ldots \times C_{j,k_j})$ such that $(C_{j,1} \times \ldots \times C_{j,k_j}) \in P(<, S)$ iff*

- *$\forall i, j : C_{j,i} \in S$,*
- *$\forall j : (\neg\exists C : C <_* C_{j,1}) \wedge (\neg\exists C : C_{j,k_j} <_* C)$, and*
- *$\forall i < k_j, j : C_{j,i} <_* C_{j,i+1} \wedge (\neg\exists C : C_{j,i} <_* C <_* C_{j,i+1})$.*

Intuitively speaking, given a partial order preference relation represented by its Hasse-diagram [121, Chapter 5.4.2], for each possible path from an element with no incoming edges to an element with no outgoing edges, an Ordered Disjunct $(C_1 \times \ldots \times C_k)$ is created where C_1 is a node with no incoming edge, C_k is a node with no outgoing edge, and there is an edge between any pair C_i, C_{i+1}.

Example 3.13.

Given the preference relation $<$ over the set of literals $S = \{A, B, C, D, E\}$ as depicted in the Hasse diagram on the right hand side, the transformation $P(<, S)$ yields the following Ordered Disjunctive Term:


$$\begin{tikzcd} & A \arrow[dl] \arrow[dr] & \\ B \arrow[d] & & D \arrow[ddl] \\ C \arrow[dr] & & \\ & E & \end{tikzcd}$$

$$(A \times B \times C \times E) \vee (A \times D \times E).$$

◇

This transformation provides the means for modelling partial order preferences in DLPODs: now every partial order preference expressed for literals can be formulated as the head of a rule in a DLPOD.

3.2.3 Implementation

This section provides a theoretical basis for computing optimal answer sets of DLPODs. Based on these elaborations, a prototype has been implemented[4] delivering preferred answer sets provided a DLPOD. The principle behind the implementation is to construct a set of Disjunctive Logic Programs out of a given DLPOD in a way that both, the set of DLPs and the DLPOD have same semantics. The presented implementation adopts the work from Brewka et al. on the implementation of LPOD [43]; this extension is, however, not entirely straightforward. More precisely, in [43] the LPOD semantics is implemented on top of a standard solver for non-disjunctive Logic Programs based on the observation that each split program corresponds to guessing exactly *one degree* for each rule with an ordered disjunction. In contrast, in the present work, a *vector of degrees* needs to be guessed which results in a Disjunctive Logic Program.

As far as similarities are concerned, the two approaches DLPOD and [43] basically share the following procedure to compute a preferred answer set given a program P:

1. Guess a satisfaction degree vector for each rule (i.e., guessing a split) and compute the answer sets for this guess. This is encoded in a program called *generator* $G(P)$.

2. For each answer set S, check whether there is no split program which yields a better answer set than S. This is encoded in a program $T(P, S)$ called *tester*, which is called in an interleaved fashion for each answer set generated by $G(P)$. Whenever the tester does not find a better answer set than S, then S is a preferred answer set.

Regarding these two points, the implementation of DLPOD and LPOD follow the same principles. However, three major differences have to be pointed out. First, in order to generate all possible splits, the DLPOD approach needs to guess a satisfaction degree *vector* per rule (instead of a single degree *value*). Second, for DLPODs the answer sets for each split needs to be generated, and a split program is—as opposed to LPODs—a *Disjunctive* Logic Program. Third, the tester program which establishes whether a better answer set can be found needs to be modified.

Before adapting the formal definitions of Brewka et al.'s generator and tester, two lemmata are provided. The first lemma states that one can replace a head symbol h in a disjunctive rule of a program P with a new symbol h' by adding some extra rules without changing the semantics of P:

[4]see `http://www.L3S.de/~kaerger/DLPOD/`

Lemma 3.1 (Ground head atom replacement)**.** *Let P be a a Disjunctive Logic Program and $r = h_1 \vee \ldots \vee h_i \vee \ldots \vee h_n \leftarrow Body_r$. a rule in P such that h_i is ground. Let further $P' = P \setminus r \cup \{h_1 \vee \ldots \vee h'_i \vee \ldots \vee h_n \leftarrow Body.\ h'_i \leftarrow h_i.\ h_i \leftarrow h'_i.\}$ such that h'_i does not occur in P. Then S is an answer set of the Disjunctive Logic Program P iff $S' = S \cup \{h'_i \mid h_i \in S\}$ is an answer set of P'.*

Similarly, it is true that a part of the body of r can essentially be "outsourced" to an external rule, as stated in the following lemma:

Lemma 3.2 (Body replacement)**.** *Let $r = Head \leftarrow Body_1, Body_2$. be a rule in a Disjunctive Logic Program P, and let further*

$$P' = P \setminus r \cup \{Head \leftarrow b', Body_2.b' \leftarrow Body_1.\}$$

such that b' does not occur in P. Then S is an answer set of P if and only if $S' = S \cup \{b' \mid Body_1 \text{ true in } S\}$ is an answer set of P'.

The *generator* program is now defined using these lemmata. The following definition makes use of the cardinality constraint notation $L\{l_1, \ldots l_n\}U$ (see Section 2.2.3) .

Definition 3.17 (Generator Program, adapts [43, Def. 10])**.** *Let r be the rule of a DLPOD of the form*

$$\begin{array}{ll} H_{1,1} \times \ldots \times H_{1,m_1} \vee & \\ \vdots & \\ H_{i,1} \times \ldots \times H_{i,m_i} \vee & \leftarrow Body_r. \\ \vdots & \\ H_{n,1} \times \ldots \times H_{n,m_n} & \end{array}$$

Then the transformation $G(r)$ is defined as the following set of rules:

(a) $\{\ 1\{c_{r,i}(1), \ldots, c_{r,i}(m_i)\}1 \leftarrow\ Body_r. \mid 1 \le i \le n\}$
(b) $\cup\ \{\ h_{r,1} \vee \ldots \vee h_{r,n} \leftarrow b_{r,1}, \ldots, b_{r,n}, Body_r.\}$
(c) $\cup\ \{\ h_{r,i} \leftarrow H_{i,j}, c_{r,i}(j).\ H_{i,j} \leftarrow h_{r,i}, c_{r,i}(j). \mid 1 \le i \le n, 1 \le j \le m_i\}$
(d) $\cup\ \{\ b_{r,i} \leftarrow c_{r,i}(j), not\ H_{i,1}, \ldots, not\ H_{i,j-1}. \mid 1 \le i \le n, 1 \le j \le m_i\}$
(e) $\cup\ \{\ \leftarrow not\ H_{1,1}, \ldots, not\ H_{1,m_1}, \ldots,$
$\quad not\ H_{i-1,1}, \ldots, not\ H_{i-1,m_{i-1}},$
$\quad not\ H_{i,1}, \ldots, not\ H_{i,j-1},$
$\quad H_{i,j}, not\ c_{r,i}(j),$
$\quad not\ H_{i+1,1}, \ldots, not\ H_{i+1,m_{i+1}}, \ldots$
$\quad not\ H_{n,1}, \ldots, not\ H_{n,m_n}, Body_r. \mid\ 1 \le i \le n, 1 \le j \le m_i\}$

Finally, the transformation $G(P)$ of a complete DLPOD is the union of all its transformed rules:

$$G(P) = \bigcup \{G(r) | r \in P\}.$$

Here, the auxiliary predicates $c_{r,i}$, $h_{r,i}$, $b_{r,i}$ ($1 \leq i \leq n$) stand for "choice", "head", and "body" auxiliary symbols. Whereas $c_{r,i}$ plays the role of modeling the choice of an actual degree vector, the $h_{r,i}$ and $b_{r,i}$ predicates are auxiliary symbols used according to Lemmas 3.1 and 3.2 for a particular choice. The Rules (a) are guessing a particular choice option forming a split. Using this choice, the Rules (b)–(e) represent the actual rules in the split program for the particular choice, by using Lemma 3.1 in Rules (c) and Lemma 3.2 in Rules (d). Finally, the Rules (e) ensure that—in case all other ordered disjuncts $k \neq i$ are false—$H_{i,j}$ has to be added if no better literal $H_{i,l}$ in disjunct i with $l < j$ is already in the model.[5]

Example 3.14. Considering the following rule $r = (A \times B) \vee (C \times D) \leftarrow Body$. the transformation $G(r)$ looks as follows:

(a) $1\{c_{r,1}(1), c_{r,1}(2)\}1 \leftarrow Body.$
$1\{c_{r,2}(1), c_{r,2}(2)\}1 \leftarrow Body.$
(b) $h_{r,1} \vee h_{r,2} \leftarrow b_{r,1}, b_{r,2}, Body.$
(c) $h_{r,1} \leftarrow A, c_{r,1}(1).\ A \leftarrow h_{r,1}, c_{r,1}(1).$
$h_{r,1} \leftarrow B, c_{r,1}(2).\ B \leftarrow h_{r,1}, c_{r,1}(2).$
$h_{r,2} \leftarrow C, c_{r,2}(1).\ C \leftarrow h_{r,2}, c_{r,2}(1).$
$h_{r,2} \leftarrow D, c_{r,2}(2).\ D \leftarrow h_{r,2}, c_{r,2}(2).$
(d) $b_{r,1} \leftarrow c_{r,1}(1).$
$b_{r,1} \leftarrow c_{r,1}(2), not\ A.$
$b_{r,2} \leftarrow c_{r,2}(1).$
$b_{r,2} \leftarrow c_{r,2}(2), not\ C.$
(e) $\leftarrow A, not\ c_{r,1}(1), not\ B, not\ C, not\ D, Body.$
$\leftarrow not\ A, B, not\ c_{r,1}(2), not\ C, not\ D, Body.$
$\leftarrow not\ A, not\ B, C, not\ c_{r,2}(1), not\ D, Body.$
$\leftarrow not\ A, not\ B, not\ C, D, not\ c_{r,2}(2), Body.$

$\diamond$

Proposition 3.2. *Let P be a DLPOD. Then*

(i) $G(P)$ is polynomial in the size of P and

(ii) S is an answer set of $G(P)$ if and only if $S \cap Lit(P)$ is a potential answer set of P.

[5]For the interested reader, Rules (a) roughly correspond to the rule in Equation (8) in [43], Rules (b)–(d) to Rule (4) in [43], and finally Rules (e) correspond to Rule (5) in [43].

Proof 3.1. Proposition *(i)* is obvious: in the generator's Rules (a)–(e), for each rule r in the original program only a constant number of rules are generated.

The proof for *(ii)* is similar to the analogous Proposition 2 in [43] with an additional application of Lemma 3.1 and 3.2. Intuitively, each guess of the $c_{r,i}(j)$ in the Rules (a) yields a split program in the sense that each rule not belonging to that particular guess is projected away by putting $c_{r,i}(j)$ in the bodies of the Rules (c) and (d). By the Lemmas 3.1 and 3.2 now, Rule (b) exactly corresponds to the guess rule in the split program corresponding to the guess modeled in Rule (a). □

Each answer set S of $G(P)$ is subsequently tested by a *tester* program $T(P, S)$ for whether it is a preferred answer set. The tester program is generated for each potential answer set S and only if the tester has no answer set, S can be considered optimal since no better, i.e., no more preferred answer set can be constructed.

Definition 3.18 (Tester Program). *Let P be a DLPOD and S be a set of literals. The tester program checking whether there is a better answer set than S is defined as follows:*

$$
\begin{aligned}
T(P,S) = G(P)\\
&\cup \{O_{i,j}. \mid c_{i,j} \in S\} \cup \{rule(r). \mid r \in P\}\\
&\cup \{better(r) \leftarrow rule(r), O_{i,j}, c_{i,k}. \mid j > k, r \in P, 1 \le i \le n, 1 \le k, j \le m_i\}\\
&\cup \{worse(r) \leftarrow rule(r), O_{i,j}, c_{i,k}. \mid j < k, r \in P, 1 \le i \le n, 1 \le j, k \le m_i\}\\
&\cup \{betterRule(R) \leftarrow better(R), not\ worse(R).\\
&\quad worseRule(R) \leftarrow worse(R), not\ better(R).\\
&\quad worseSet \leftarrow worseRule(R).\\
&\quad betterSet \leftarrow betterRule(R), not\ worseSet.\\
&\quad \leftarrow not\ betterSet.
\end{aligned}
$$

Intuitively, the predicate $better(r)$ fires if there is a dimension i in r's satisfaction degree vector according to S such that $T(P, S)$ found an answer set S' with a satisfaction degree vector that is better in position i. Conversely, $worse(r)$ fires if a dimension i can be found where S' is worse. It is worth noting that it is not required to encode ϵ in the tester, since the rules defining $better(r)$ and $worse(r)$, respectively, are only constructed for comparable and unequal options (here, pairs of literals are compared occurring in the same disjunct of the same rule but at difference positions). Next, $S' \succ_r S$ (expressed by $betterRule(r)$) holds if there is a dimension where S' is equal or better and there is no dimension where S' is worse. Analogously, $worseRule(r)$ determines rules r such that $S \succ_r S'$ (cf. Definition 3.13). The two remaining rules and the final constraint ensure that the answer set S' only "survives" if it is better in some rule and not worse in any other rule. Thus, only those answer sets $S' \succ S$ pass (cf. Definition 3.14). Finally, if no such answer set S' is returned by the tester, S is optimal. From those observations, the following proposition can be stated.

Proposition 3.3. *Let S be an answer set of $G(P)$. If $T(P, S)$ does not have any consistent answer sets, then S is an optimal answer set of P.*

It is further noted that LPODs are just a special case of DLPODs:

Proposition 3.4. *LPODs are a special case of DLPODs and the preferred answer set of an LPOD computed by $G(P)$ and $T(P, S)$ correspond 1-to-1 to the preferred answer sets computed by the generator and tester presented in [43].*

Proof 3.2. This is easy to see by the correspondence of $G(P)$ and $T(P, S)$ 3.2, i.e., the answer sets of the generator and tester programs outlined in [43] only differ by the auxiliary symbols $h_{r,i}$ and $b_{r,i}$ which are introduced according to both lemmata. $\square$

Based on this result, DLPODs can easily be implemented using a standard solver for Disjunctive Logic Programming. A prototype DLPOD solver has been implemented in the course of this thesis; it is based on DLV [94], a Disjunctive Logic Programming system implementing the stable model semantics. The DLPOD implementation is available for download at
`http://www.L3S.de/~kaerger/DLPOD/`.

3.2.4 Complexity Results for DLPODs

In the following, some complexity results for DLPODs are given which partly derive from lifting respective results from normal LPODs to the disjunctive case of DLPODs.

Considering the complexity of finding an optimal answer set for DLPODs one can observe the following. First, it is easy to see that determining whether an optimal answer set exists is not more difficult than determining whether any answer set exists because if there is an answer set, there is also at least one optimal one. Thus Theorem 1 from [43] can be straightforwardly lifted by the Σ_2^p-completeness of disjunctive Logic Programs [64].

Theorem 3.1. *Deciding whether a DLPOD P has an optimal answer set is Σ_2^p-complete.*

Secondly, the same lifting to the second level of the polynomial hierarchy also works for checking whether S is optimal.

Theorem 3.2. *Deciding whether an answer set S of a DLPOD is optimal is Π_2^p-complete.*

Proof 3.3. This proof is analog to the membership proof of Theorem 2 in [43].

Π_2^p-hardness follows because (in variation of the proof for co-NP-hardness for the non-disjunctive LPOD case, see [43, Proof of Theorem 2]) it is possible to use, instead of a reduction of SAT, a variation of the standard disjunctive encoding of QSAT with two quantifier alternations into ASP, see e.g. [65]. □

Theorem 3.3. *Given a DLPOD P and a literal $l \in Lit(P)$, deciding whether there exists an optimal answer set S such that $l \in S$ is in Σ_3^p.*

Proof 3.4. First, note that the algorithm from [43] can, with slight modifications, be used to solve exactly this decision problem. Namely, one needs to simply add to the outer $G(P)$ computation the constraint "$\leftarrow$ *not l*." thus invalidating answer sets that do not contain l in the initial guess to G(P). Obviously, this modification yields an algorithm which is in the complexity class $\Sigma_2^{p\Sigma_2^p}$. This indeed boils down to $\Sigma_3^p = \mathsf{NP}^{\Sigma_2^p}$ because instead of $\Sigma_2^{p\Sigma_2^p} = (\mathsf{NP}^{\mathsf{NP}})^{\Sigma_2^p}$ one can simply use the outer Σ_2^p oracle also to compute the inner NP oracle calls. □

It is important to note that head-cycle free DLPODs preserve better computational properties than general DLPODs. Actually, all example rules and policies in this thesis are head-cycle free and thus fall in this class of programs. For this reason it is worth to explicitly state the following theorem.

Theorem 3.4. *Given a head-cycle free DLPOD P and a literal $l \in Lit(P)$, deciding whether there exists an optimal answer set S such that $l \in S$ is Σ_2^p-complete.*

Proof 3.5. Hardness follows immediately from hardness of this problem for non-disjunctive LPODs because any head-cycle free DLPOD can be transformed into an LPOD with the same answer sets (cf. [19, 63]). As for membership, it has been stated already in Proposition 3.1 that each split program of a head-cycle free program is again head-cycle free. Thus, one can observe that guessing a split and checking whether an answer set S exists such that $l \in S$ is in NP and, likewise, checking non-existence of a better answer set is in co-NP. Consequently, since a co-NP and an NP problem are combined in an interleaved fashion, the overall problem is in Σ_2^p. □

3.2.5 Partial Order Preferences under the Stable Model Semantics

To the best of the author's knowledge, none of the existing approaches to preference handling in Logic Programs allows for general *partial* order preference expressions on attribute level.[6] This section describes related work and potential other ways to encode partial orders in Answer Set Programming. More detailed elaborations on general approaches for preference expressions in policy handling will be given later in Section 5.1.

When discussing partial order encoding of preferences under the answer set semantics, it is worth referring to the work presented in [25] and [103]. There, LPODs are used as a basis for the policy language PPDL (Preference Policy Definition Language) which is describing the behaviour of a network node and therefore allowing for preference definitions between possible actions a node can perform. This approach allows to model only a subset of partial order preferences by ordered disjunctions of disjunctions. This way, levels are assigned to elements of distinct branches of the partial order (see [25, Section 4.4]) and hence not all partial order preferences are covered. For instance, the one described in Example 3.13: there is no unique level assignment that keeps B and D as well as C and D incomparable. However, the semantics of partial orders requires both pairs to be incomparable.

In the following, further discussions on potential ways to encode subsets of partial order preferences will be given in the form of counterexample for some naive encodings.

First, one conceivable approach would be to introduce auxiliary predicates for any set of literals one is indifferent about. That way, the example $c_{name} \times (c_{telephone} \vee c_{pcode})$. would be encoded by two rules:

$$\begin{aligned} c_{name} &\times aux. \\ 1\{c_{telephone}, c_{pcode}\}1 &\leftarrow aux. \end{aligned}$$

However, by this modeling one cannot capture partial orders similar to Example 3.13: would aux model the choice between B and D or between C and D? In both cases the modelling is incorrect since either D dominates C or B dominates D.

Second, modeling the paths of the partial order from Example 3.13 could be realized by several unconnected ordered disjunctions, e.g.,

$$\begin{aligned} A \times B \times C \times E. \\ A \times D \times E. \end{aligned}$$

However, this does not lead to the expected behaviour either: assuming one would add a constraint "$\leftarrow A.$" to the program, this encoding would produce a single

[6]Of course, object level preferences (in contrast to attribute level preferences), such as the relation $\succ$ from Definition 3.14, can perfectly form a partial order preference with existing approaches such as LPOD.

optimal answer set $\{B, D\}$, instead of the two expected optimal answer sets $\{B\}$ and $\{D\}$.

One might now try to exclude the literals from the respective other paths by adding the literals with a default negation to the bodies, e.g.,

$$\begin{aligned}
&\leftarrow A. \\
A \times B \times C \times E &\leftarrow \mathit{not}\ D. \\
A \times D \times E &\leftarrow \mathit{not}\ B, \mathit{not}\ C.
\end{aligned}$$

This encoding though may cause the exclusion of valid paths. For example, adding the fact "C." prevents the second path rule from being satisfied and returns $\{C, B\}$ as the only optimal answer set, although one would also expect $\{C, D\}$ as answer set.

Finally, encoding all the paths of a partial order into a cardinality constraint of an LPOD delivers the same candidate answer sets as a DLPOD encoding but fails to correctly compare those answer sets. For example, the partial order preference defined by the DLPOD "$(A \times B) \vee (C \times D)$." could be simulated by the following LPOD with cardinality constraints:

$$\begin{aligned}
&1\{aux_1, aux_2\}1. \\
A \times B &\leftarrow aux_1. \\
C \times D &\leftarrow aux_2.
\end{aligned}$$

However, if the constraint "$\leftarrow A$." was added to the program, according to the LPOD semantics the answer set $\{B, aux_1\}$ are dominated by $\{C, aux_2\}$ although B and C would be incomparable in the presented partial order. Hence, this LPOD would return $\{C, aux_2\}$ as the single optimal answer set excluding $\{B, aux_1\}$ which is expected to be optimal as well.

It is further important to note that this work focuses on a Pareto semantic-based preference notion. In [44] two other preference notions are introduced, namely cardinality-preferred and inclusion-preferred. However, in [44] both preference notions are proven to be not general enough (see the motivation for Definition 10 and 11 in [44]). Moreover, these two preference definitions are based on counting and hence do not reflect the qualitative nature of partial order preferences.

Chapter 4

Reactive Policies

Semantic Policies are declarative behaviour descriptions for automated systems. In the past, different policy languages have been described as a powerful means to express a system's behaviour by defining statements about how the system must behave given different conditions and situations. Existing policy frameworks allow for the evaluation of complex conditions including external dependencies such as location of the requester, the time of the request, ontological definitions of concepts, etc. Typically, a request or requester has to fulfill these conditions in order to let the request being executed (e.g., access is granted). With the growing dynamics of the Web the need for a reactive control based on changing and evolving situations arises. Thus, as it will become clear later in this chapter, an advanced policy language shall support *reactivity.* Roughly speaking, it is important to not only define the condition of whether something is true, but to include the *event* that triggers the evaluation of this condition as well as its *reaction* in the definition of the policy. This specification of policies is a shift from classical, purely binary grant-or-deny semantics to a more detect-and-react manner of policy-based behaviour control. Among the existing powerful semantic policy languages, none includes semantics for reactivity (see Section 5.2 for detailed elaborations on this particular statement). This chapter explains how the traditional policy model has to be extended to so-called *reactive policies.*

Reactive policies—in contrast to the standard policies—allow to

1. define events triggering the evaluation of conditions and

2. actions that are taken as reactions for such an evaluation.

Reactive policies make it possible to declaratively define the reactions to a specific situation or event within one single language while at the same time relying on well-established security and trust mechanisms. This way, reactive policies reflect the dynamics of the Web: updates in the knowledge stored on some peer are turned into reactions in the real world, thus closing the gap between semantic knowledge and behaviour on the Web as well as ensuring trusted and policy-compliant communication. The requirement for *reactive* semantic policies, based on reactive rules, has been formulated already in several publications [35, 34, 6, 81], however, a unified policy language that combines reactive behaviour control with advanced trust establishing techniques was not developed up to now.

To this end, in the following section, a policy language is presented for defining declarative, exact, and detailed behaviour descriptions by means of Event-Condition-Action rules (ECA rules [136]). The presented language combines ideas from reactive languages on the Web [7] or on database systems [136] with advanced policy reasoning and Trust Negotiations in a way that reactivity is based on secure evidences. To achieve this, a policy language is formally defined that features an extension to the formal Trust Negotiation process [51, 35] and is based on an interplay between negotiations and reactive rule evaluation. As negotiation model, a standard message exchange framework will be adopted which was presented in [51, 35].

After this language is introduced, Section 4.2 will further illustrate how the presented language can be used in a Social Web scenario. This scenario will show nicely how the language's reactivity features can be exploited in combination with the management of distributed systems, such as automated peer interaction and gathering of distributed information from the Web. Subsequently, the language's implementation is described and its use in the application SPoX [82]. In SPoX, reactive policies are enforced for the Social Network and communication tool Skype.

The following contributions will be presented in this chapter:

- a formal framework for expressing and enforcing reactive policies with support for Trust Negotiation (partially published in [30, 6, 81, 5]);
- an approach for privacy protection on the Social Web exploiting reactivity and social data (first presented in [83, 87]);
- SPoX (Skype Policy Extension), an implementation of the formal framework and its application to Skype (presented in [82]).

4.1 A Framework for Reactive Policies

In this section, a policy framework will be defined which combines reactive behaviour control with policy evaluation and policy-based Trust Negotiation. The framework consists of a policy language and a protocol for exchanging Trust Negotiation messages based on policy evaluations. Before the framework is formally defined, an informal description of the policy language's syntax and the evaluation procedure is presented followed by a motivating example illustrating the basic principles of the approach.

At a glance, in the presented policy language, policies have the form

ON *Event* **IF** *Condition* **DO** *Action*

and are interpreted according to the standard Event-Condition-Action rule semantics [136]: in case *Event* occurs and, at the same time, *Condition* is evaluated to true, the action *Action* is performed.

Events are any kind of change in the environment that is propagated by exploiting the infrastructure of the Internet such as "a flight is rescheduled", "an auction will end in 10 minutes", or "a money transaction is completed".

Conditions on the one hand, make use of information exposed on the Web such as "is the requester listed in my FOAF profile?" or "is there an alternative flight connection available?" and, on the other hand, they will allow for security checks that may, for example, only be carried out by a Trust Negotiation, such as "is the flight change notification coming from an authorized peer belonging to the airline?" or "is the requester a citizen of the city Hannover?". To prove such properties, typically the exchange of signed credentials is required which may only be carried out given an appropriate level of trust. Finally, the

Actions will turn the knowledge into real world changes such as "rebook the flight according to my preferences", "redirect the call to my mobile phone", or "dispatch the goods".[1]

An example policy expressed in the presented language is given in Figure 4.1. There, it is stated in a reactive rule that phone calls from students are automatically accepted on Wednesday morning. If a call comes in, in order to let the ECA Rule (4.1) apply, the predicate $isStudent(\_)$ has to be proven which requires the predicate $credential(\_,\_)$ to be true. The intuition behind this predicate is, that it is true if the peer given in the first argument provided a credential (which in turn will be bound to the second argument). In such a case, the fact $credential(peer, cred)$ would be added to the state or knowledge of the system. Typically, in an initial state, when a call comes in, $credential(\_,\_)$ is not true. Thus, the policy owner's agent will initiate a counter-request asking for a credential. The goal of the procedure is to make $credential(\_,\_)$ true, i.e., adding it as a fact to the system's state. Then, the reply to the initiator (i.e., the counter-request) of the call comprises two parts: the request for proving $isStudent(\_)$ and the policy itself that tells the other peer how to prove $isStudent(\_)$. The idea behind this reply is that the policy owner asks the caller about evidences helping to prove $isStudent(\_)$. From the policy that is sent along with the response,

[1]The reader is referred to [5] where more examples are given for the use of reactive policies.

$$\begin{aligned}&\texttt{ON callArrives(Time,Call,Caller)}\\&\texttt{IF isStudent(Caller),}\\&\quad\texttt{isWednesdayMorning(Time)}\\&\texttt{DO letThrough(Call).}\end{aligned}\tag{4.1}$$

$$\begin{aligned}\texttt{isStudent(Person)} \leftarrow\ &\texttt{credential(Person,Credential),}\\&\texttt{Credential.issuer} =' \texttt{uni} - \texttt{hannover}',\\&\texttt{Credential.type} =' \texttt{studentid}'.\end{aligned}\tag{4.2}$$

$$\begin{aligned}\texttt{knowsPassword(Person)} \leftarrow\ &\texttt{declaration(Person,Password),}\\&(\texttt{Password} =' \texttt{let\_me\_in}').\end{aligned}\tag{4.3}$$

$$\texttt{allow(letThrough(Call)).}\tag{4.4}$$

$$\texttt{callArrives(X,Y,Z)} \Leftarrow \texttt{callArrivesOnSkype(X,Y,Z).}\tag{4.5}$$

$$\begin{aligned}\texttt{letThrough(Call)} \rightarrow\ &\texttt{showNotification(Call),}\\&\texttt{activateHeadSet(),}\\&\texttt{passToSkype(Call).}\end{aligned}\tag{4.6}$$

Figure 4.1: An example policy with a reactive policy rule that automatically accepts student's calls on Wednesday morning.

the caller learns that a student credential is required. If available, the caller's agent sends back the credential and the negotiation is successful. Of course, the credential itself can again be protected by another policy (e.g., the calling peer may disclose a student credential only to university employees) which would lead to a request back forming a multi-step negotiation. Finally, if the negotiation is successful and it is Wednesday morning, the action part of Rule (4.1) is executed. The execution of actions needs to undergo a policy check again since automated re-actions have to be checked to ensure execution safety (as it will be discussed later in this section). Hence, for each action a that is going to be executed, the predicate $allow(a)$ has to be proven. In this example, accepting the call is always allowed (see the empty body in Rule (4.4)).

The language will now be defined in detail, first starting with its syntax and subsequently providing its semantics.

4.1.1 Syntax

In the following, the syntax of the reactive policy language is defined. For the sake of completeness, the EBNF[2] of the language definition is given in Figure 4.2.

[2]extended Backus Naur Form, see [76]

The language is composed of three different types of rules:

- reactive rules in ECA form (e.g., Rule (4.1) in Figure 4.1),
- definition rules to define complex events (Rule (4.5)) and complex actions (Rule (4.6)),
- implication rules (Rules (4.2–4.4)) which form the declarative part of the policy and are processed like standard Logic Programming rules.

The detailed syntax is defined as follows.

Definition 4.1 (SYNTAX). *A policy P is a set of definition rules, reactive rules and inference rules. A definition rule is either an event definition or an action definition. Let E_a, E_c (atomic and complex events, respectively), A_a, A_c (atomic and complex actions, respectively) be sets of atoms and let e_a,e_c,a_a,a_c be respective elements of those sets.*

The set of events E over E_a and E_c is the set of atoms e of the form $e ::= e_a \mid e_1 \bigtriangledown e_2 \mid e_c$ where e_1, e_2 are arbitrary elements of E. Given $e \in E$, an event definition is any expression of the form "$e_c \Leftarrow e.$".

A set of actions A over A_a and A_c is the set of atoms of the form $a ::= a_a \mid a_1, a_2 \mid a_c$ where a_1, a_2 are arbitrary elements of A. Given $a \in A$, an action definition is any expression of the form "$a_c \rightarrow a.$".

Let $Cond$ be a set of atoms. A reactive rule r is any rule of the form "`ON` e `IF` $c_1, \ldots, c_m, not\ c_{m+1}, \ldots, not\ c_n$ `DO` $a.$" with $c_1, \ldots, c_n \in Cond$.

An implication rule is of the form "$head \leftarrow c_1, \ldots, c_m, not\ c_{m+1}, \ldots, not\ c_n.$" with $head ::= c_0 \mid allow(a)$ and c_i $(0 \leq i \leq n) \in Cond$, $a \in A_a \cup A_c$.

4.1.2 Semantics

In this section, first, a procedural description of the policy evaluation process is given. Subsequently, this description will be detailed by a formal definition of the semantics.

Procedurally, the evaluation of a reactive policy happens as follows:

1. If an event occurs, the event expressions of all reactive rules are evaluated to determine if the event expression matches the event happened. All reactive rules whose event part applies are called *pending*.
2. For all pending reactive rules r all the evidences from the condition part are collected which are not known from the current state but can be proven by other peers (for example, the evidence stating that the origin of a signal is a student and thus can provide a student ID credential).

```
Policy              ::= Rule | Rule Policy
Rule                ::= ECARule
                        | EventDefinition
                        | ActionDefinition
                        | ImplicationRule
ECARule             ::= 'ON' EventExpression
                        'IF' ConditionExpression
                        'DO' ActionExpression '.'

EventDefinition     ::= EventAtom '⇐' EventExpression '.'
EventExpression     ::= EventAtom
                        | EventExpression '▽' EventExpression
EventAtom           ::= EventName {'(' ArgumentList?  ')'}?

ActionDefinition    ::= ActionAtom '→' ActionExpression '.'
ActionExpression    ::= ActionAtom
                        | ActionExpression ',' ActionExpression
ActionAtom          ::= ActionName {'(' ArgumentList?  ')'}?

ConditionExpression ::= BodyLiteral | BodyLiteral ',' ConditionExpression
BodyLiteral         ::= 'not'?  Literal
Literal             ::= PredicateName {'(' ArgumentList ')'}?

ImplicationRule     ::= Head '.'  | Head '←' ConditionExpression '.'
Head                ::= Literal | 'allow(' Literal ')'

ArgumentList        ::= Term | Term ',' ArgumentList
Term                ::= Constant | Variable
```

Figure 4.2: Syntax of the reactive policy language in EBNF.

3. Those requests together with the corresponding policy are sent to the parties defined in the evidence's peer variable and thus initiate a negotiation.

4. If the condition of r is not true but there are remaining evidences that can be proven by other peers, the negotiation goes on.

5. If the condition of r evaluates to true, its variable results are bound to the action part, it is checked if the action execution is authorized (i.e., $allow(action)$ is true) and the actions in question are executed.

As stated above, the presented approach relies on the negotiation model presented in [51, 35]. For the sake of completeness, two fundamental principles of this model, namely *state predicates* and *policy filtering* are detailed and recalled in the following.

State predicates

A distinguished subset of predicates, called *state predicates*, may change their extension dynamically (as a consequence of message exchanges, updates, etc.). State predicates comprise `credential`, `declaration`, the `in` predicate, as well as predicates that define the attributes of compound objects like credentials and declarations (like the attribute `issuer` of a credential in Rule (4.2)) .

Policies can be based on a variety of properties provided by other peers. Some of them can be encoded in digital credentials; called strong evidence as its validity and authenticity can be certified by means of cryptographic techniques. On the opposite side of the spectrum (weak evidence) unsigned declarations are found such as those commonly issued in copyright agreements just by clicking a button in a pop-up window [32]. The state predicates supported in the presented framework are adopted from [35] and their descriptions are provided as follows.

`credential`: The state predicate `credential` appearing in an atom part of a condition represents that some peer needs to provide a credential in order to make this atom true.

`declaration`: The predicate `declaration` is analogous but it does not require a signed certificate rather than a semi-structured object to be sent by another peer. Such objects can be strings when, for example, a password is required by another peer as for the predicate `knowsPassword(_)` in Rule (4.3). Here, the other peer is requested to send a string which will be compared to the correct password. Other examples are retrieving a peer's address or company. However, a declaration—as opposed to a credential—does not provide means to prove that a given object is certified, e.g., it does not include evidences about the correctness of the given address.

`in`: The `in`-predicate allows to consider external data sources as given facts in the state of the knowledge base. With this predicate, ground facts do not have to be present explicitly but can be retrieved on demand from external sources during the reasoning process. Later, in Section 4.2 this predicate is used to incorporate social data from the Web into the reasoning process. However, it can also be used to interact with arbitrary systems if, for example, the policy framework requires to reason on legacy data. For the `in`-predicate the syntax introduced in [129] is adopted (cf. Rule (4.9) on page 88 later in this thesis).

Compound objects: For accessing attributes of compound objects (for example the student ID credential in Figure 4.1), a FLORA-like syntax is supported as in [32], as follows. One can express by $X.attr : v$ the fact that X has an attribute $attr$ with value v (which is handled internally as syntactic sugar for $attr(X, v)$).

Conceptually, in the logical model, state predicates are defined by sets of ground facts, while their actual implementation may be ad hoc or cast into the system on evaluation time. For simplicity, it is assumed that policy rules do not change during negotiations. On the contrary, the dynamics of the system is reflected in the state predicates whose validity may change.

In contrast to traditional non-reactive Trust Negotiation approaches, in the reactive case, peer interactions are triggered by a variety of events instead of requests only (cf. Section 2.1.3). For example, in the standard negotiation model described in [35, 51], the other peer (there expressed by the meta-term *peer*) is in general clearly identified by the requester in the negotiation's context. Conversely, in the reactive approach, sources for policy evaluations may not only be requests with a single requester peer. Therefore, binary state predicates are used for representing credentials and declarations, where the first argument explicitly mentions the peer in charge of providing missing evidence and the second argument denotes the evidence itself. For example, a peer may try to make a condition $credential(p, c)$ true by asking peer p for the credential c. Accordingly, any variable occurring as the first argument of `credential` or `declaration` in a rule r's body will be called *peer variable* of r.

Provisional predicates are state predicates that are proven by another peer—typically declarations or credentials. Although the presented policy language implements negation as failure in general, negation of provisional predicates is prohibited. As it has been stated in [35], this restriction ensures that policies are monotonic, that is, as more credentials are released the set of permissions does not decrease. Moreover, the restriction on negations makes the implication rules build a stratified program; therefore negation as failure has a clear, PTIME computable semantics that can be equivalently formulated as the least model semantics, the well-founded semantics or the stable model semantics [15].

Sending and filtering policies

As described in the example above, the negotiation principle implies that a request for evidences from another peer comprises the policies whose evaluation requires those evidences. This is needed in order to provide the peer receiving the request with reasons why the request is sent (otherwise, in Figure 4.1, a calling student would not be able to understand the benefit of providing her student credential). Another reason is that the policy rules implicitly contain all possible options (e.g., sets of credentials or declarations to disclose) to fulfill them. Thus, the policy itself is a far more compact representation of the options making a negotiation succeed. In summary, sending the policy along with the negotiation messages has the advantages that [32]:

- user agents can see at once multiple alternative ways of fulfilling a policy and select those that more closely match the user's preferences (cf. Chapter 3);

- this can be done without paying the price of combinatorial explosions of compound requests, which may decrease performance because of an increasing number of messages and/or size of messages

However, policies are sensitive resources and it may be necessary to hide parts of them [35]. That policies are sensitive becomes obvious in example Policy Rule (4.3): since the password itself is part of the policy, sending a non-filtered policy would unintentionally disclose the password. Moreover, not all parts of the policies are important for the receiver of a request which is another motivation for filtering. The standard strategy here is to disclose only those parts of the policy which are necessary for the current negotiation state. In the following, a policy which is sent along with a negotiation message will be called filtered policy and will be referred to as a function *FilteredPol* turning a peer's policy into a filtered policy suited for the particular negotiation step. The mechanism how to specify and decide if a policy rule needs to be filtered is described in detail in [35] and is adopted for the presented approach. In the following, this work will focus on the interplay between negotiation message exchange and reactivity rather than on the creation of the message's content. The mechanisms how a filtered policy is used to select a "promising" set of credentials from a portfolio are discussed in detail in Chapter 3.

In what follows, the semantics of the reactive policy language will be presented by starting with the event, condition, and action parts and subsequently by defining the message exchange and event handling in a negotiation.

Events

An event triggers the evaluation of a rule if it matches the rule's event part as follows:

Definition 4.2 *An event expression e is true for a given atomic event e_A and a reactive policy P, denoted $e \vdash_P e_A$, if one of the following conditions is true*

1. *e is an atomic event and there is a substitution σ such that $e\sigma = e_A$*

2. *e is an event expression of the form $e_1 \bigtriangledown e_2$ and there is a substitution σ such that $e_1\sigma \vdash_P e_A \vee e_2\sigma \vdash_P e_A$ (disjunctive connection of events)*

3. *there is a substitution σ, an event definition $e_c \Leftarrow e_{def}. \in P$, and $e_c\sigma = e$ as well as $e \vdash_P e_{def}$ holds.*

Condition evaluation

In the following, it is defined what satisfies a condition part of a reactive rule and what message exchanges and negotiations are triggered during the evaluation.[3]

Definition 4.3 (State atom, context, support). *A* state atom *(respectively* state literal*) is an atom (respectively literal) whose predicate is a state predicate. A* goal *is a set of ground atoms and a* context *is a set of ground state atoms.*[4] *A* support *of a goal G from a set of implication rules P is a goal G_n containing state literals only and occurring as the last step in a complete, finite SLD-derivation $G, G_1, G_2, \ldots, G_n$. With a slight abuse of notation, goals will be identified with the* set *of literals constituting the goal (instead of a sequence of literals).*

Contexts will be used to express the set of facts that are true for the current state of the negotiation (sometimes also called negotiation state). Supports will be used to denote the set of predicates that are collected from the policy and that are not true (yet). Those predicates are candidates for requests to other peers in order to collect evidences to prove the initial goal. For example, if a student Alice calls the peer owning the policy in Figure 4.1 on a Wednesday 11AM, the peer's context contains the fact $isWednesdayMorning('11AM')$, the support for the goal $isStudent(alice)$ contains $credential(alice, Credential)$ with $Credential$ being a variable and thus not instantiated. As soon as Alice provides a credential c, the instantiated fact $credential(alice, c)$ will be added to the policy owner's context.

In general, a support is not always a context because a support is not necessarily ground but may still contain unbound variables. Roughly speaking, the support of a goal is the set of literals that are to be verified in order to prove the goal. A goal can only be verified, if the literals in its support hold in the context. Hence, starting from the support of G, the problem of deriving G from the context is reduced to pattern matching against the current context. This is formalized by the following lemma describing the validity of a goal G given a policy and a context.

Lemma 4.1. *Let P be a set of implication rules of a policy and let C be a context. Then $P \cup C \models G$ iff there exists a support S of G from P and a substitution σ such that $S\sigma \subseteq C$.*

[3]In this section, the reader is assumed to be familiar with SLD-derivations and the most general unifier (*mgu*) [15].

[4]Note that a context is a Herbrand model containing only state atoms.

Resulting actions

Now, since it is stated clearly how an event part and a condition part of a reactive rule is satisfied, the language's semantics is defined by stating which actions are executed and which messages are exchanged given

- a certain policy,
- a local negotiation state, and
- a history of events and exchanged messages.

The formal framework consists of the following components:

- a (global) *history* (of message/signal exchanges) $\mathcal{H}$;
- a mapping *Pol* associating each peer p with its policy $Pol(p)$;
- a mapping *Ctx* that for each peer p and time t returns a context $Ctx(p, t)$ (called the *negotiation state*);
- a *multiset* of *pending rules* $Pending(p, t)$ for each peer p and time point t;
- a set of *executed actions* $Exe(p, t)$ for each peer p and time point t.

The history $\mathcal{H}$ is a finite set of *messages* $\langle p_1, p_2, M, t\rangle$, where p_1 is the sender, p_2 is the recipient, M is the message content, and $t \in \mathbb{N}$ represents a time point.

Beyond the agents potentially participating in negotiations, the set of peers also comprises some special peers called *environmental peers* that represent the system and the environment in which the system is situated. This contrasts with traditional Trust Negotiation models where only peers involved in a negotiation can trigger a negotiation. Here, the environmental peers act as transmitters of events which account for specific events called *signals*: events that are not requests or disclosures generated by an agent in the system. Examples for signals are notifications about environmental changes, updates in a database, an arriving call (as in Figure 4.1), or the change of the online status of a peer.

In summary, the set of possible messages comprises:

- *signals* where M is a ground event and p_1 is an environmental peer;
- *requests* where $M = \langle G, FilteredPol\rangle$; intuitively, p_1 asks p_2 for evidence that can help in proving G from the filtered policy *FilteredPol*;
- *disclosures* where M is a context consisting of a logical description of a set of credentials and declarations.

Message exchange

The system's behaviour is further constrained due to the interplay of messages and events as follows:

Pending and executing rules. At any time, a *signal* may activate a reactive rule thus making the rule pending; i.e., its condition part needs to be evaluated as a goal against the set of implication rules (see Lemma 4.1). This has several effects, including the start of a negotiation. Formally, for all signals $\langle p_1, p_2, e, t\rangle \in \mathcal{H}$ (where e is an event) and for all pending rules $R = (\texttt{ON}\ e'\ \texttt{IF}\ G\ \texttt{DO}\ a)$ in $\mathit{Pol}(p_2)$ such that $e \vdash_{\mathit{Pol}(p_2)} e_A$ and $\mathit{mgu}(e_A, e') = \sigma$, if $G\sigma$ has at least one support from $\mathit{Pol}(p_2)$ then it holds that:

- p_2 has to request for evidences that are to be provided by other peers. Or, more formally, there have to be requests for all peers p_3 occurring in some support of $G\sigma$ from $\mathit{Pol}(p_2)$. Further, $\mathcal{H}$ must contain a request $\langle p_2, p_3, m, t'\rangle$, with $t' < t$, $m = \langle G\sigma, \mathit{FilteredPol}\rangle$, and *FilteredPol* is a filtering of $\mathit{Pol}(p_2)$ satisfying the faithfulness condition below. To identify correctly the recipients p_3, the first argument of all `credential` and `declaration` atoms must be ground in all supports (guaranteed by the call-safeness conditions discussed later in this section);
- the set of pending rules is cumulative, that is

  $$\mathit{Pending}(p_2, t) := \mathit{Pending}(p_2, t-1) \cup \{R\sigma\}$$

  for all R that are pending at time t. It may be noted that in a real system, pending rules may not always be simply accumulated: they may be eliminated when the corresponding negotiations terminate or a timeout occurs. However, this conceptual simplification is made here for the sake of simplicity.

If $G\sigma$ does not have any support, that is, if the condition part of R is not fulfilled, then the event e does not cause the execution of the pending rule R.

Faithfulness. Filtered policies always correctly approximate the real policy in the following sense: for all requests $\langle p_1, p_2, \langle G, \mathit{FilteredPol}\rangle, t\rangle \in \mathcal{H}$, and for all contexts C, it holds that

$$\mathit{FilteredPol} \cup C \models G \text{ implies } \mathit{Pol}(p_1) \cup C \models G\,.$$

Note that the converse is not required to hold: not all conclusions that are valid in $\mathit{Pol}(p_1) \cup C$ also hold in the filtered version $\mathit{FilteredPol} \cup C$. This is due to the fact that policy filtering may remove relevant information from $\mathit{Pol}(p_1)$ during filtering either for confidentiality or efficiency (see page 76 in Section 4.1.2 for details on policy filtering).

Disclosure safety. Each disclosure during a negotiation process must be authorized by the local policy. Formally, for all $\langle p_1, p_2, C, t\rangle \in \mathcal{H}$ (where C is a context), and for all atoms $P(p_1, e) \in C$ where $P \in \{\texttt{credential}, \texttt{declaration}\}$,

$$\mathit{Pol}(p_1) \cup \mathit{Ctx}(p_1, t) \models \mathit{allow}(\mathit{release}(e))\,.$$

Relevance. No evidence should be disclosed without need, that is, any disclosed evidence should be relevant to some previous request. This notion of relevance is justified by Lemma 4.1: since all the proofs of a goal G depend on some support, all disclosed evidence are required to occur in some of those supports. Formally, for all disclosures $\langle p_1, p_2, C, t\rangle \in \mathcal{H}$ (where C is a context), and for all atoms $P(p_1, e) \in C$ where $P \in \{\texttt{credential}, \texttt{declaration}\}$, there must exist a time point $t' < t$ and a request $\langle p_2, p_1, \langle G, \mathit{FilteredPol}\rangle, t'\rangle \in \mathcal{H}$ such that $P(p_1, e)$ belongs to a support of G from *FilteredPol*.

Evidence acquisition. The context is cumulative as well, i.e., facts are never removed from the context: $\mathit{Ctx}(p, t) := \mathit{Ctx}(p, t-1) \cup \bigcup\{C \mid \langle p_1, p, C, t\rangle \in \mathcal{H}\}$. This is again a conceptual simplification: in real systems the elements of a context are first cryptographically verified, and—as a consequence—some may be discarded and thus removed from the set of facts in the context; moreover, context elements may be eliminated when the corresponding credential expires. Considerations in that direction have been presented for example in [93]. However, such issues will be relevant for efficiency discussions and are out of the scope of this work.

Execution safety. $\mathit{Exe}(p, t)$ is the set of all actions that are executed as a result of a pending rule's successful evaluation. Formally, $\mathit{Exe}(p, t)$ is the set of all actions a such that for some reactive rules $R = (\texttt{ON}\ e\ \texttt{IF}\ G\ \texttt{DO}\ a.)$ in $\mathit{Pending}(p, t)$, the following two conditions hold

1. $\mathit{Pol}(p) \cup \mathit{Ctx}(p, t) \models G$
2. $\mathit{Pol}(p) \cup \mathit{Ctx}(p, t) \models \mathit{allow}(\mathit{execute}(a))$.

All other reactive rules remain pending. Accordingly, $\mathit{Pending}(p, t+1)$ is the multiset of all rules in $\mathit{Pending}(p, t)$ whose condition is not true (yet), or formally, all rules such that either G or $\mathit{allow}(\mathit{execute}(a))$ is not a logical consequence of $\mathit{Pol}(p) \cup \mathit{Ctx}(p, t)$.

Call safeness. As it has been stated before, the peer responsible for providing a state predicate has to be instantiated (grounded) in some argument of the atom. For defining this formally, in the following, call safeness is introduced based on a notion of call typing.

A *call typing* is a mapping $\tau : \mathsf{Pred} \to \wp(\mathbb{N})$ that associates each predicate symbol p to a set of argument indexes. Intuitively, if $\tau(p) = \{1, 3\}$ then the first and third argument of p needs to be ground in each call to p. As stated before, the first argument of specific provisional predicates is required to be ground in order to identify the party responsible for the provision; formally:

$$\tau(\texttt{credential}) = \tau(\texttt{declaration}) = \{1\}\,.$$

The operator τ is extended to atoms in the natural way: if $\tau(p) = \{i_1, \ldots, i_k\}$ then for all atoms $A = p(t_1, \ldots, t_n)$ let $\tau(A) = \{t_{i_1}, \ldots, t_{i_k}\}$.

An implication rule R with head A is *call safe* iff for all atoms B occurring in the body of R and for all variables X occurring in B, $X \in \tau(B)$ implies $X \in \tau(A)$. Moreover, a goal G is *call safe* iff for all atoms A occurring in G, $\tau(A)$ is ground.

Proposition 4.1. *If a goal G is call safe, then all of its supports from a call safe program are call safe, too. In particular, the first arguments of all* `credential` *and* `declaration` *predicates in the support are ground.*

A *reactive* rule `ON` e `IF` G `DO` a is *call safe* iff for all atoms B occurring in G and for all variables $X \in \tau(B)$, X occurs in e.

This guarantees that the most general unifier of e and the occurred event makes G call safe and, consequently, all `credential` and `declaration` predicates in G are associated to specific peers. This condition ensures that at any stage during a negotiation, the receivers of request messages are determined.

In summary, this formal framework models the message exchange needed for Trust Negotiation and the flow of events and reactions in a single setting. In the next section, it will be shown how this framework can be exploited to improve behaviour control and privacy protection on the Social Semantic Web.

4.2 Reactive Policies on the Social Semantic Web

In the last years, the way people use the Web changed from pure consumption of content to an active participation in the generation of content. With the advent of Web 2.0 technology the Web became more and more a communication infrastructure rather than a collection of Web pages. Blogs, networking platforms, and social software emerged and are currently one of the most frequently used ways of communication. However, the well-established privacy preserving techniques that were developed for the conservative way of communication cannot be directly applied to this new way of information exchange. For example, personal data is ubiquitously disclosed in social platforms and it is difficult to control who is going to access whose data. Resources, such as pictures or videos are published and shared; but access to those resources needs to be restricted carefully. Furthermore, the new opportunities of communications do not only raise issues for privacy protection but also issues of protection from undesired messages and filtering of communication flow. More precisely, following the ideas from [131], the user's attention itself is to be considered a particular resource that needs to be protected against the overload of undesired messages [109].

The problem of access control or behaviour control is typically addressed in Social Network applications by offering their users ways to express simple preferences or policies about which notification they are interested in and which user or message is allowed to approach them in certain ways and contexts. Such privacy preferences are typically limited in several respects:

- First of all, the notion of events is not reflected in current privacy settings. Although there are fine-grained options available for tuning who is allowed to see the details of one's user profile or to access files uploaded to a platform, the fact that being approached by other people is a privacy issue is not reflected.

- Second, Social Network applications suffer from a so-called walled garden: information available in one social platform is not available in another. Thus, privacy settings cannot refer to properties of users or objects stored in a different platforms, hence, although it was stated that friends are allowed to see the profile on one platform, a user who is a friend on a different platform may not be allowed to see it only because she is not redundantly defined to be a friend on *both* platforms.

- Third, the fact that users are registered at different platforms requires means for platform independent authentication, e.g., by means of strong evidences; otherwise, it is impossible to identify users across the borders of social network applications.

This section describes the Social Web as a particular environment where reactive policies can be put in place to improve the user experience in both, privacy protection and system behaviour control. To this end, two aspects are considered:

1. how reactivity helps to guide the behaviour of Social Web applications to provide a privacy-aware and trust-enabled communication and data protection;

2. how social data semantically exposed on the Web can serve as input to policy evaluation processes thus improving privacy protection capabilities;

3. how Trust Negotiation can be exploited to establish authentication and authorization crossing the borders of Social Web platforms.

In the following, first the problems and requirements will be outlined by the hand of a set of example policies. Subsequently, it is detailed how the presented approach provides the intended features and finally, a prototype will be described showing how Skype can be enhanced by fine-grained and security-driven behaviour control. There, the policy framework presented in the previous section is acting in order to enforce policies on Skype such as the one given in Figure 4.1: calls, notifications, and chat messages are either accepted or initiated only if certain conditions are fulfilled.

4.2.1 Problem Definition and Requirements

As sketched above, the limitations in the current privacy settings on the Social Web are manifold [50]. To showcase how reactive policies are particularly helpful to overcome these limitations, the following set of simple policies will be used throughout this section. These policies are controlling the flow of messages and the disclosure of information in the context of Social Network applications and make obvious the scenario which will be addressed with the concept of reactive policies.

(A) Do not accept Skype calls unless the caller is either in my Skype contact list or listed as friend in one of my Social Network profiles. For all other people calling me, deny the call and open a chat connection instead.

(B) Show notifications about emails arriving and contacts going online on Skype only if the origin is either in my family group on Flickr or in my family category on Skype. Never show such notifications if my computer is in presentation mode.

(C) Do not allow wall posts on my Facebook profile that contain a word that is listed in a specified collection of bad words.

(D) Automatically accept friend requests on any platform if I ever wrote a paper with the requester or if the requester's Web site is a blog I regularly comment on.

(E) Only people who attended the International Semantic Web Conference are allowed to comment on this blog and to see pictures tagged with `ISWC`.

(F) Forward tweets from Twitter to my mobile phone if I am offline and the author of the tweet is listed in my FOAF profile.

(G) Do not accept calls by students unless they are calling during my consultation hours.

(H) Show this blog entry only to friends who are students at the University of Hannover.

The Social Web as a walled garden

Stating for one Social Network application that only friends are allowed to perform a certain action in one's profile does not cover people defined as friends on another application. Skype, as used in example Policy (A), is only aware of contacts explicitly listed in Skype. Consequently, it may happen that a call from someone who is a friend on Facebook or listed in one's FOAF profile is blocked. This only happens because one did not explicitly (and redundantly) add the caller as a Skype contact as well.

But when looking at the example policies, privacy preferences may perfectly be based on data scattered over the Social Web. And, moreover, it becomes obvious that not only data stored in the Social Networks is required for privacy decisions. In Policy (D) data about publications and in Policy (F) data about conferences attendees is used. Generally speaking, any kind of social data that can be gathered from the Web shall be considered for policy decisions. Recent studies on interruptions in mobile phone usage actually show that disturbance of messages heavily depends on the social context of their origin [131, 47]. For example, one may not mind to be approached by people one follows on Twitter; or accept friendship requests from people who regularly post in one's blog (see Policy (D)).

As a matter of fact, all this information about social contacts and context is available on the Web: with whom one discusses in forums (e.g., represented in SIOC), who one knows (e.g., represented in FOAF profiles), with whom one is working, be it on publications (e.g., listed on the computer science bibliography Web site DBLP[5]) or on software projects (e.g., DOAP). Unfortunately, current solutions for privacy preferences do not make use of this data [71] and thus suffer from the "walled garden" of Social Networks [40, 87].

Reactivity on the Social Web

Communication in Social Network applications such as Twitter, Skype, and Facebook plays an increasing role. Such applications typically offer plenty of ways to communicate among their members, examples are chat messages, wall posts, and

[5]accessible via SPARQL Endpoints, such as `dblp.L3S.de/d2r/`

micro blogging. Moreover, in addition to such messages, a lot of communication happens via notifications. For example, in Skype the change of a contact's online status is shown in a small notification pop-up. Other kinds of notifications include friendship requests, notifications about being tagged in a picture, etc. In any case, independently of whether a message comes from a single user or if it is a notification generated by the Social Network application, in order to fight this information overload [109], users typically want to canalize the way such messages approach them and may want a system to automatically reject messages, automatically accept or deny friendship request, etc. However, such advanced reactivity-driven privacy preferences are not easily expressible in current Social Web applications although they are purely reactive: events occur and they typically require (semi-)automatic handling by the users (or the application doing it on their behalf).

4.2.2 Policies Acting on the Social Web

As stated before, reactive policies extend the classical grant-or-deny-access policies by the notion of events and (re-)actions. This extension fits the event-driven nature of Social Networks [71] that includes the reaction to interactions and notifications. Here, the classical approach of allowing or denying access does not apply anymore since communication attempts, tagging people on pictures, or posting on forums requires more advanced control including event observation and reaction triggering. The policy framework presented in the previous sections is able to express simple yet powerful reactive policies reflecting this situation as shown in the following example.

Example 4.1. Recalling Scenario (A): a reactive policy that changes a call into a chat based on a local contact list or based on information gathered from a Social Network may look as follows:

$$\begin{aligned}&\texttt{ON callComesIn(User,Call)}\\&\texttt{IF not isInMyContactList(User),}\\&\quad\texttt{not isMySocialNetworkFriend(User)}\\&\texttt{DO denyCall(Call),sendChatMessage('Hello ...',User).}\end{aligned} \tag{4.7}$$

◇

Guiding and controlling the behaviour of social software requires conditions for policy decisions to be based on the information which is available on social software applications. Moreover, as most people use more than one social platform bearing their social data, the immense volume of data available, if combined, can lead to more fine-grained policies. This data consists of personal data and

Social Network information on the one hand and, on the other hand, it consists of user-generated content, like bookmarks, tags, reviews, photos, and blogposts (the so-called object-centered Social Network [40]). Furthermore, with the advent of the Semantic Web, and the Linked Data movement[6] in particular, this social data can be linked to more general concepts revealing information like what is a blog post about, in which country was a picture taken, etc.

Semantic Web technologies provide standards and models to make social information easily accessible. Information about users is described in standard formats like RDF, which allows to deal with data from different sources in a uniform way. In addition, due to the emergence of new Web technologies in the era of Web 2.0, many Social Web applications open their platforms for external developers by providing an API in order to access the platform's data.

In the policy evaluation process all this data, either extracted from proprietary Social Platforms or exposed on the Semantic Web, is not isolated anymore. Once the data is retrieved, a policy reasoner can combine the retrieved data from multiple sources of the Social and Semantic Web to evaluate advanced, fine-grained and user-adjusted policies. This is based on the observation that relationships among people are not only extracted from explicitly mentioned links in the network but "citizens form relationships and self-organize into communities around shared interests" [40] thus following the principle of *augmented Social Networks* [78].

For example, looking at the policies (A)–(H), a reaction to a message or to an event in the Social Network may depend on social data about the network, be it if someone is a FOAF friend (as used in Scenario (A)), a student (Scenario (G)), attended an event (E) , or co-wrote a paper (D). Consequently, the condition part of a reactive policy needs to retrieve such data in order to contribute to the decision about whether to execute the action or not. By interleaving reactive rules with implication rules, one is able to combine the definition of reactive behaviour with the declarative definition of policies. As shown in the following example, social data can be combined in order to support complex policy decisions.

Example 4.2. Implication rules allow to further define what is meant by the predicate `isMySocialNetworkFriend(_)` that was used in Policy (4.7) in the previous example as follows:

$$\begin{aligned}
&\texttt{isMySocialNetworkFriend(Person)} \leftarrow \texttt{iAmFollowingOnTwitter(Person)}.\\
&\texttt{isMySocialNetworkFriend(Person)} \leftarrow \texttt{isMyDBLPCoAuthor(Person)}.\\
&\texttt{isMySocialNetworkFriend(Person)} \leftarrow \texttt{isFacebookFriend(Person)}.
\end{aligned} \tag{4.8}$$

In a similar way, one may not only define the concept "MySocialNetwork-Friend" but also "FOAF friend of a FOAF friend of mine" or "person living

[6]`http://linkeddata.org`

in my city", etc. Looking back at Scenario (D), one may, for example, define the concepts of "interest-sharers" [40] similar to the concept "MySocialNetwork-Friend" in Policy (4.8) in the following way: from the Skype profile of a caller, one can retrieve her Web site. Based on SIOC data exposed on this Web site one can find out if it is a blog one regularly comments on [29]. The second condition in Scenario (D) can be determined by a SPARQL query to DBLP that delivers all published papers of a person. Based on this, it is possible to decide if a caller ever was a co-author of the person to be called. ◇

Looking back at Example 4.1 and at Policy (4.8), in order to evaluate the predicate $iAmFollowingOnTwitter(\cdot)$, it has to be connected to the actual Twitter API. For this the `in`-predicate (see page 74 in Section 4.1.2 on state predicates) is exploited to connect to Twitter during the policy evaluation process and to retrieve details about users, e.g., if one is another user's so-called *follower*[7] on Twitter. Following this approach, Policy (4.8) has to be extended with an implication rule as follows.

$$\begin{aligned}
&\mathtt{iAmFollowingOnTwitter(Person)} \leftarrow \\
&\qquad \mathtt{in([Friend], TwitterWrapper : getPeopleIAmFollowing()),} \\
&\qquad \mathtt{Person = Friend.}
\end{aligned} \tag{4.9}$$

For the evaluation of the in-predicate, a wrapper is required to be implemented that offers a method *getPeopleIAmFollowing()*. The results of this method are bound to the variable `Friend` and, internally, for each result r_i a fact is added to the policy of the form

$$\mathtt{in([r_i], TwitterWrapper : getPeopleIAmFollowing()).}$$

More details on such wrappers will be given in the following section. However, in the implementation presented later, the translation and replacement of such predicates is done automatically. Moreover, the creation of Logic Programming-style policies is kept away from users; instead, they are supported by means of a graphical policy editor.

The problem of identifying users across different social platforms is delivered by the framework's handling of strong evidences. This approach assumes each user owning credentials which are certifying her name on a certain platform. This way, one can, for example, define an implication rule for the predicate $isMyDBLPCoAuthor(\_)$ as follows.

[7]A user A is called a follower of another user B if A subscribed to regularly receive the micro blogging messages of B.

$$\begin{aligned}&\mathtt{isMyDBLPCoAuthor(Person)} \leftarrow\\ &\qquad \mathtt{credential(Person, C)},\\ &\qquad \mathtt{C.dblpIdentifier} = \mathtt{Dblp\_Id},\\ &\qquad \mathtt{in([\,], DBLPWrapper : isMyCoAuthor(Dblp\_Id))}.\end{aligned} \tag{4.10}$$

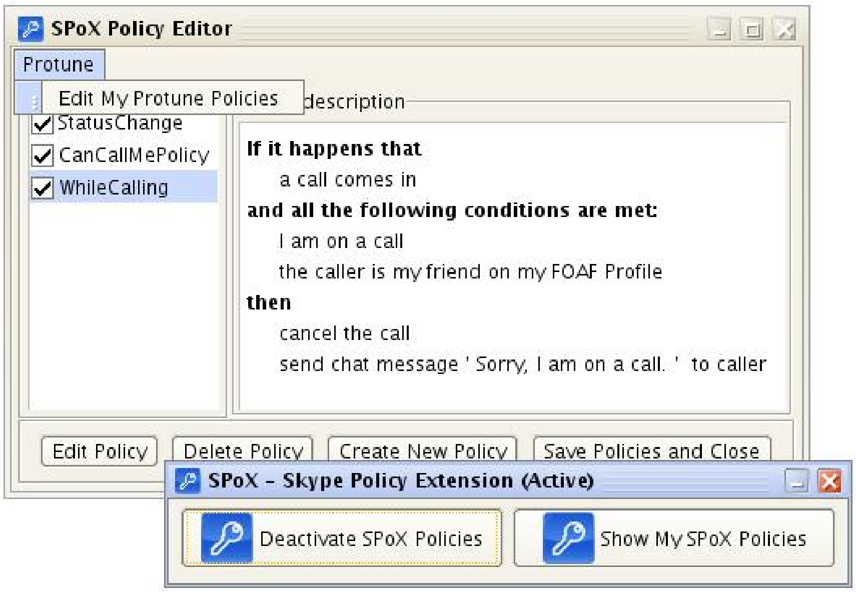
This rule shows how a credential can be used to exactly determine a user's identifier on DBLP. As explained in Section 4.1.2, the state predicate `credential` will trigger a negotiation message telling another peer (to be specified in the predicate's first argument) that a credential is required. Given this credential is disclosed to the policy owner, it is added to the negotiation state and subsequently its attribute `dblpIdentifier` is used to query the DBLPWrapper for co-authorship.

The concept of reactive policies has been realized and implemented in the context of the Social Web as described in the following section.

4.2.3 SPoX—a Skype Policy Extension

The presented approach of controlling the behaviour of Social Network applications by exploiting social data is promising to be applied to the Social Network and communication tool Skype. First, Skype's privacy policies controlling who is allowed to do what are very limited: a contact of a Skype user can belong to three classes only: a user's friend, a foreigner, or a person who was blocked by the user. For example, one cannot set up filtering rules based on attributes of contacts such as which category does a contact belong to (see scenario Policy (B)). Moreover, the way a user is allowed to approach another cannot be filtered. For example, either both, chats and calls, are blocked or none of both (see scenario Policy (A)). Second, Skype offers an API which eases the retrieval of social data about other users on the one hand and, on the other hand, it allows to easily influence Skype's behaviour. Third, Skype offers a channel separated from chat messages and call streams, that allows to transfer data to other peers. This is particularly helpful in order to exchange negotiation messages since no additional channel has to be set up between two peers.

This section presents SPoX, a prototype which applies the idea of reactive policies to Skype. With policies formulated for SPoX a user can state, for example, that only friends on Twitter and Flickr or people listed in the user's FOAF profile are allowed to call and calls from all other users are automatically blocked and turned into a chat (cf. Policy (A) and see the policy in Figure 4.5). Another example policy allowed in SPoX is the filtering of status change notifications:



If it happens that
a call comes in
and all the following conditions are met:
I am on a call
the caller is my friend on my FOAF Profile
then
cancel the call
send chat message ' Sorry, I am on a call. ' to caller

Figure 4.3: The control panel (bottom) and the policy panel with an overview of the current SPoX policies (top).

during working hours one may only want to be notified about colleagues going online and not about family members or friends.

SPoX policies can be expressed based on such semantic Social Network information as well as social data about other users exposed on the Web or accessible via APIs of proprietary Social Network applications. Further, context information like local time, day of the week, or online status can be used for policy specification. Based on this information, SPoX automatically reacts to events in Skype's Social Network according to user-defined reactive policies.

Architecture

SPoX[8] [82] is implemented as a stand alone program which connects to the Skype client running on the local system (see Figure 4.4). It consists of a reactive policy engine influencing the behaviour of the local Skype client. SPoX' policy engine is an implementation of the framework presented in Section 4.1. The engine's core is a reactive extension of the (non-reactive) policy engine Protune [32, 35]. Further, the engine features wrappers to access and reason on Semantic Web data (via SPARQL endpoints or RDF files) and social data (via the proprietary Social Platform APIs of Flickr, Skype, and Twitter) and to trigger actions at the local Skype client.

[8]`http://www.L3S.de/~kaerger/SPoX`

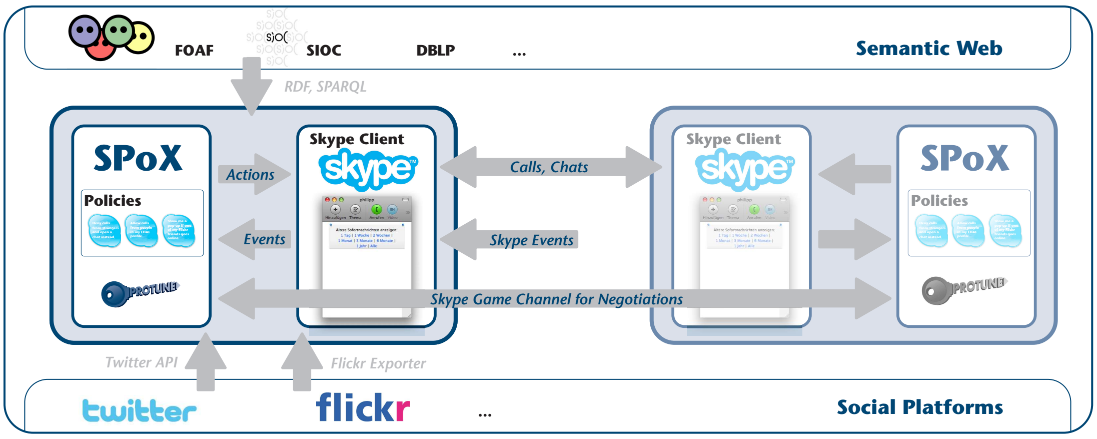


Figure 4.4: The architecture of SPoX: two peers are negotiating and utilizing the negotiation context retrieved from the Social Web and the Semantic Web. Negotiations are carried out exploiting Skype's game channel.

The connection from the policy engine to the Skype client is achieved via the the open Skype Java API[9], a Java implementation representing the Skype Client API. This connection

1. observes events from Skype and passes them to the policy engine,
2. uses the Skype-inherent application channel[10] for transferring negotiation messages and
3. performs actions in Skype, such as cancelling a call, changing the mood message, or sending a chat message.

From a GUI perspective, SPoX is equipped with a

- graphical policy editor (see Figure 4.5), with a
- policy panel providing an overview of all policies for modifying and activating single policies (see Figure 4.3 top), and with a
- control panel for activating or de-activating SPoX (see Figure 4.3 bottom).

Gathering Social and Semantic Web data in SPoX

To allow social data to be incorporated into the policy reasoning process of SPoX, the policy engine features wrappers for several data sources. Logically, the gathering of this data is triggered whenever the evaluation has to decide if an `in`-predicate (see Section 4.1.2) is true or not.

[9]`http://developer.skype.com/wiki/Java_API`
[10]This channel is typically used for game information, see `http://skype.easybits.com/`.

Semantic Web data. In order to incorporate social data exposed on the Web into the policy decision process of SPoX, a general wrapper was developed which queries SPARQL endpoints. This wrapper is used to incorporate co-authorship data from DBLP's Semantic Web endpoint. To this end, the SPARQL endpoint mirroring the DBLP dataset[11] is queried. A second Semantic Web wrapper is offered which accesses arbitrary RDF files. This wrapper is used to check friendship relations stored in FOAF profiles.

Proprietary Social Web data. Further, Social Web data which is not semantically exposed on the Web per se but accessible via social platform APIs is made available to SPoX by the following wrappers.

Twitter a wrapper accessing the API of Twitter[12] in order to determine if a user is a follower of another user on Twitter;

Skype a wrapper for gathering personal data from Skype, for example if somebody is listed as a contact or if a user was blocked;

OpenSocial a wrapper extending SPoX for gathering social data from any OpenSocial[13] platform has been added;

Flickr information about a user's contacts on Flickr are retrieved in form of SPARQL queries to FOAF profiles which are created on the fly by a Flickr RDF exporter [14].

Defining reactive policies in SPoX

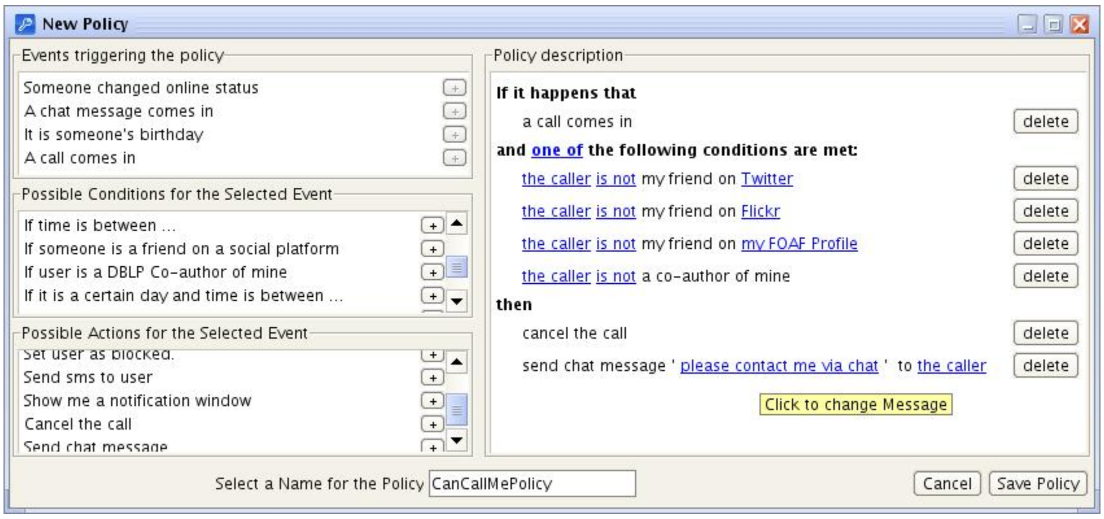
When launching SPoX the user is presented with a small control panel for activating or de-activating SPoX and for opening the policy panel (see Figure 4.3). In this panel, policies can be activated and de-activated by means of the check boxes on the left, and they can be deleted or modified. Most prominently, SPoX comes along with an easy-to-use reactive policy editor (see Figure 4.5). It is inspired by the design of classical e-mail filters in mail clients like Outlook or Thunderbird. On the left-hand side, events, conditions, and actions can be selected and added to the policy that is compiled on the right. The underlined words can be modified in a pop-up window which appears when the links on corresponding words are clicked. Conditions and actions can be applied either to the initiator of the events (named "the caller" in Figure 4.5) or to an arbitrary Skype user. The conditions can be changed between conjunctive and disjunctive

[11]see `dblp.L3S.de/d2r/`

[12]see `apiwiki.twitter.com`

[13]see `www.opensocial.org`

[14]see `apassant.net/home/2007/12/flickrdf`



New Policy

Events triggering the policy

- Someone changed online status
- A chat message comes in
- It is someone's birthday
- A call comes in

Possible Conditions for the Selected Event

- If time is between ...
- If someone is a friend on a social platform
- If user is a DBLP Co-author of mine
- If it is a certain day and time is between ...

Possible Actions for the Selected Event

- Set user as blocked.
- Send sms to user
- Show me a notification window
- Cancel the call
- Send chat message

Policy description

If it happens that
a call comes in — delete
and one of the following conditions are met:
the caller is not my friend on Twitter — delete
the caller is not my friend on Flickr — delete
the caller is not my friend on my FOAF Profile — delete
the caller is not a co-author of mine — delete
then
cancel the call — delete
send chat message ' please contact me via chat ' to the caller — delete
Click to change Message

Select a Name for the Policy: CanCallMePolicy

Cancel | Save Policy

Figure 4.5: The SPoX policy editor for setting up a new policies. The interface is similar to message filters known from e-mail clients.

connection (by clicking on "one of" respectively "all") and conditions can be negated (by clicking on "is not" respectively "is").

Not all conditions one may impose on a person approaching via Skype can be expressed by such a user interface. Therefore, SPoX supports the definition of arbitrary predicates which can be used in conditions. This way, arbitrary concepts like "interest-sharer" can become part of the user-defined policies. This way, new concepts can be defined and exploited for decisions SPoX has to take. This has been done in Policy (4.8) where the concept *isMySocialNetworkFriend* is further defined by implication rules. SPoX allows to add such predicate definitions to the user's policy via the link "Edit my Protune policies" (see Figure 4.3) that opens the Protune policy editor. This editor allows the user to add more fine-grained policies and concept definitions by means of implication rules. In a SPoX policy's condition, one can refer to these predicates by means of a special condition where predicates can be freely typed. The semantics implemented in SPoX evaluates the policies in the order they are listed, and, Skype behaves as if there were no policies unless a policy contradicts.

Trust Negotiation within SPoX

The Trust Negotiation protocol presented in Section 4.1 is used in SPoX for two specific tasks, for strong authentication with digital certificates and for retrieving additional information from other peers.

Digitally signed credentials are used in SPoX for strong authentication by exploiting Trust Negotiation as follows. In Policy (H), for example, a call to a user's Skype client requires the caller to prove (by means of a digitally signed credential) that she is a student of a particular university. This example exactly corresponds to the implication rule in Figure 4.1. In SPoX, such a policy is evaluated as follows.

Assuming the policy engine facing an implication rule of the following form:

$$\begin{aligned}\texttt{isStudent}(\texttt{Person}) \leftarrow & \texttt{credential}(\texttt{Person}, \texttt{Credential}), \\ & \texttt{Credential.issuer} =' \texttt{uni} - \texttt{hannover}', \\ & \texttt{Credential.type} =' \texttt{studentid}'.\end{aligned} \tag{4.11}$$

Given an event which requires the evaluation of the predicate $isStudent(\_)$, the SPoX client informs the requester (which is bound to the variable $Person$) that the performance of the `credential` predicate is required. If the requester's conditions to disclose the credential are fulfilled, the credential is sent (if not, a second step of the negotiation is triggered). Consequently, if the attributes of the transmitted credential fulfill the last two atoms in Policy 4.11, the SPoX client will let Skype accept the call. In SPoX, negotiation messages and credentials are well transferred using the Skype game channel, so no separate communication channel is needed.

The concept of declarations is exploited in SPoX to link users across different Social Web platforms. In Policy (A), for example, one may state in SPoX that only Flickr friends are allowed to call. In order to find out if a caller is a Flickr friend, a state predicate $flickrName(Peer, Name)$ is used that, according to the semantics definition and Lemma 4.1, leads to a request to the call initiator. The SPoX client on the caller's peer will receive the request for the declaration and, if provided by the caller, will send back the instantiated fact thus disclosing her Flickr name. Of course, disclosing the Flickr name can again be protected by a policy and would lead to a multistep negotiation. It has to be noted that for such a declaration there is no easy way to verify if a requester's Flickr name is actually the one provided. However, there are technologies such as OpenID that provide solutions in this case (see Chapter 6 where this discussion is revisited).

In general, the application of strong authentication via data exchange and Trust Negotiation, as it is needed for an advanced policy control in Skype, is provided by an interplay of triggering events and negotiation message exchange. The formal model ensures that any action taken in Skype fulfills the given policies, either by adding facts gathered from the Semantic Web to the negotiation state or by exchanging evidences thus mutually and automatically establishing trust among Skype peers.

Chapter 5

Related Work

The research presented in this thesis was carried out in a context which touches manifold related research areas; most prominently, the field of formal policy specification and reasoning which has been introduced in detail in Section 2.1. This chapter provides an overview and a review of the state of the art on research approaches related to the present thesis. Further, a comparison of those existing approaches to the present work will be given thus justifying this thesis' introduction of new methodologies; this includes the following two areas:

Policies and Preferences There are several approaches introducing some notion of preferences for policy specification and the according reasoning process. Besides that, a number of rule-based languages (independent of their use as policy languages) support preferences in their semantics. It needs to be made explicit, what justifies the introduction of new methodologies as done in this thesis. However, as it will be shown later, existing approaches do either not completely support partial order preferences or do not fit the needs of advanced policy handling or Trust Negotiation.

Policies and Reactivity Reactive behaviour control in distributed systems is a well-studied area which applies techniques related to the present work. Moreover, supporting reactivity in policies has been done—similarly to the present work—for some policy languages. However, providing a well-defined, semantic understanding of reactivity as a semantic policy language's first-order citizen has not been done so far. And, on top of that, an interleaved evaluation of reactive rules and negotiation messages is a new topic.

In the following, both directions will be reviewed in detail.

5.1 Policies and Preferences

Preference models nicely capture potentially complex user intentions. Therefore, extending policy representation techniques with such models is considered suggestive and attracted a lot of research work. This section gives a general overview on research where this combination has been investigated and it further provides a comparison of these approaches to the one taken in the present thesis. Subsequently, in the Sections 5.1.1 and 5.1.2, particular emphasis will be given first on the use of preferences for automated agent interaction and negotiations and, second, on the use of preferences to extend the expressiveness of rule-based languages.

Platform for Privacy Preferences (P3P). The most prominent approach which appears to combine privacy policies with preferences is P3P, the Platform for Privacy Preferences [54]. P3P was developed by the World Wide Web consortium (W3C). It is a protocol that allows Web sites or Web services to explicitly define for what purpose data is stored that is exposed by visitors of their sites. The goal is to give the user more control over the data she is disclosing by browsing the Web. This is achieved by providing users the possibility to define their preferences concerning their data. For example, such a preference may be that a user's name shall only be stored for certain purposes (e.g., for customer care but not for advertisements of interested third party companies) or only for a specific time span. To express these preferences, the W3C proposed a language called APPEL [92] which was proven to be error prone even for simple privacy preferences in [3]. On top of that, in [3], an alternative language based on XPath was presented.

Compared to the present thesis, P3P shares only the term *preference* and the concept of privacy protection. In contrast to the policy systems considered in this thesis, P3P does not enforce policies: it solely compares the statements provided by a Web site host with a user's preferences. Whether the disclosed data is handled as pretended by the P3P statement cannot be checked after the disclosure (see also page 19 on policy enforcement). The second difference lies in P3P's interpretation of the term *preference*. This interpretation is opposed to the preferences-enabled policies as described in Chapter 3. P3P does not support an ordering of alternatives; instead, preferences are rather simple static configurations comprising no information about a user's options.

Preference Policy Description Language (PPDL). In [25] a policy language is introduced that combines policy enforcement with actual preferences among potential steps in a policy evaluation. PPDL straightforwardly extends PDL, the Policy Description Language, that was already introduced in 1999 by Lobo and Chomicki [98] (see also page 101 in Section 5.2.2, where the reactivity of PDL is reviewed). In PPDL preferences are applied in the context of policy-driven network control. The authors introduce the extension to PDL in order to allow for preference definitions between possible actions preventing hazardous network situations which may be generated by a policy. However, compared to the work presented in Chapter 3, PPDL is restricted in several terms.

First, preferences in PPDL cannot be expressed among all possible predicates: they are only allowed to be defined among actions in the so-called consistency monitors. That is, only in case of hazardous network situations, these policies are evaluated and the most preferred action is blocked in order to prevent the hazard. If this blocking is not possible (i.e., the most preferred action has to be performed), the second option is chosen [25], and so on.

The second restriction is PPDL's limitation to total order preferences. As stated already in Section 3.2.5, PPDL focuses solely on total orders. It provides a leveling approach for partial orders but this leveling does not work for all sets of partial order preferences.

5.1.1 Preferences in Negotiations

Policy-based Trust Negotiation is a process influenced by various parameters. It is not only the partners' policies which need to be aligned, it is also the strategies the partners follow. This strategy influences the negotiation including whether one is rather unassertive in disclosing credentials or more interested in a successful negotiation without caring too much about a potential loss of privacy. An overview how such general dimensions can be modeled and how they influence the negotiation has been given in [141]. The particular dimension that takes into account a preference order among credentials involved in a negotiation in order to decide which negotiation path to follow is not provided in [141]. In the following, research work will be reviewed that incorporates preferences into the negotiation process in order to ease an automated decision if the policies alone do not give enough evidences to decide for a unique next negotiation step.

A quantitative approach for comparing negotiation paths has been described in [48]. In this approach numerical weights are attached to credentials as well as to policies. Then, accumulated costs of negotiation paths are compared in order to decide which path to follow, that is, which credential to disclose. However, this approach requires to assign numbers to all credentials which is rather complex and unintuitive for users [88] compared to a qualitative partial order. Moreover, assuming the linear aggregation of numerical weights as a measurement for the composition of objects is undesirable. Looking back at the negotiation scenario in Section 2.1.3, for cases like the quasi-identifier, it may even yield a lack of security if weights are just aggregated. Such cases could only be avoided if weights would also be assigned to combinations of credentials, thus letting disclosure sets containing only one of the two quasi-identifier credentials be more preferred than a set containing both. In general, conditional preferences as they are allowed in this thesis' approach are not included in [48]. Finally, using weights in [48] implies the preference order to be a total order which is too restrictive since it may not be possible to express a preference between every two credentials (cf. for instance [66]). Similar approaches have been presented in

- [139] where a point-based trust management model is introduced enabling the users to quantitatively distinguish the sensitivity of their credentials by assigning a certain amount of points to each credential; and in
- [99] where preference elicitation for negotiating agents and case studies concerning how to acquire knowledge about the user's preferences are described.

Due to the quantitativeness of the sensitivity values used, both approaches basically share the same drawbacks as the quantitative one presented in [48].

5.1.2 Preferences in Logic Programming

Preferences have been introduced to formal knowledge representation frameworks since 1980 already, when preferences were applied in Circumscription or Default Logic. Later, the emphasis was shifted to Logic Programs and Extended Logic Programs under the Answer Set Semantics [58].

The most notable distinction to be drawn among the countless approaches for preference expressions in Logic Programs (see [58] for an overview) is according to the objects, preference relations are imposed on. The majority considers preferences among rules. This is practical wherever it is important for the knowledge to cover which rules to apply in preference to another. In policy languages, these sorts of preferences have been applied in order to solve conflicts among possibly contradicting policy rules; for example, if one rule states that access is denied and another states that access should be granted. A prominent solution for this problem is to always prefer a policy not granting access to a policy granting it ("denial takes precedence" [8, Section 2.2]), as it is applied, for example, in the TAB system [22] or in the Hierarchical Temporal Authorization Model [24].

Only the minority of logical frameworks with preferences actually allows preferences among literals. Besides LPOD which was used as a basis in this thesis, the common approaches for preferences on atom or literal level instead of rule level are

1. Sakama and Inoue 2000 [116] for extended Logic Programs under the Answer Set semantics,
2. Pradhan and Minker [112] for definite Logic Programs (i.e., Logic Programs without negation),

3. Lifschitz [96] for circumscription,
4. Inoue and Sakama 1999 [75] for abduction (according to [58] this is equivalent to Sakama and Inoue 2000), and
5. Brewka 2004 [42] for extended Logic Programs under the Answer Set semantics.

However, for the approaches 1–4, it holds that preferences are applied statically. That is, the knowledge model consists of two disjunct parts, the rules and the preferences and no reference can me made from the preference knowledge to the world knowledge. This makes it impossible to capture conditional preferences (cf. Section 2.2.1) where specific preference relations only hold given certain states in the world. The only alternative approach to LPOD was presented by Brewka in [42]. There, a more general notion for preferences in extended Logic Programs is given where the generation of answer sets and the evaluation of preference expressions is not amalgamated anymore (as it is in LPOD). [42] presents a logical framework that actually covers LPODs. However, as for all approaches mentioned so far in this thesis, partial order preferences are not covered in [42].

5.2 Policies and Reactivity

The concept of reactive policies lies in the intersection of two research areas, namely *advanced policy reasoning and trust management* on the one side and *reactive rules on the Web* on the other. In the following, first, approaches to reactivity on the Web will be detailed and subsequently, general policy languages will be reviewed that support a certain notion of reactivity.

5.2.1 Reactive Rules on the Web

In order to specify the reactive behaviour of systems acting on the Web, the concept of Reactivity on the (Semantic) Web has been introduced [73, 11, 101, 7]. Adding declarative, reactive rules to enrich systems with reactivity features is not at all a new issue. In fact, reactive rules have been extensively studied in the 90s in the area of database systems, in particular in the field of Active Databases [136]. However, the differences in the requirements and the new challenges posed by the Web, and in particular by the Semantic Web, justified the proposal of several specific new approaches. In fact, unlike in databases, reactivity on the Semantic Web must cope with a highly distributed and heterogeneous environment, where heterogeneity is present both in data and languages. Moreover, actions and events are not anymore restricted to the database domain only (e.g., updates of a field) but have to include far more general actions and events which have to be made available to and handled by a Semantic Web reactivity framework.

The most promising approach to represent reactivity on the Web is using Event-Condition-Action rules (ECA rules) as it has been stated in [45]. These rules provide a suitable common model for reactivity, with modularization into clean concepts, with a well-defined information flow, and a clear and well understood semantics. However, in early approaches, ECA rule languages resembled the triggers developed for active databases, where events were always related to the modification of data (e.g., updates in XML or RDF documents stored on the Web).

A first attempt to shift database techniques for reactivity to a Web context was Active XQuery [38] which used the same syntax and switches as standard SQL triggers [7]. A later work worth to be mentioned was done by Wood et al. who explored ways to express reactive rules acting on homogeneous data; but, unlike the former reactive database languages, they do not supervise edits in database tables but on XML data [11] (and later also on RDF [111]). Wood's XML-ECA language is based on XPATH expressions for both, event and condition part of the rules. These expressions are evaluated on XML data in order to trigger rules or to check if a rule's condition part holds. The action part of a rule is represented as an XQuery which is in charge of modifying XML data accordingly. Rules are triggered as soon as a data edit matches the XPATH expression in a rule's event part. This data is then bound to a variable `$delta` which can be reused in the condition and action part of the rule. The XML-ECA language can be directly applied to RDF data if it is represented in XML. However, [111] also presents an ECA language acting on sets of triples, i.e., acting on graphs. There, the event part contains a path pattern of the graph (using the language RDFPath), the condition part is a condition on the graph (potentially referring again to the `$delta` variable), and the action part is a set of instructions adding or deleting triples. This approach was shifted to a distributed setting by means of the language RDFTL (RDF triggering language) [107]. RDFTL allows to define ECA rules on RDF data in a P2P environment.

Soon it was realized that the Web required more general forms of events than simple data edits. On the Web, an event should be the (volatile) perception of an external occurrence that may be a change of data but may also be a manifestation of an executed action or simply an incoming message. Hence, several advanced languages and frameworks have been developed to represent and evaluate reactive rules in a Web context. For example, in order to cater for complex events which are formed by combining simple events, more advanced reactive rule languages have been defined for database systems, e.g., SnoopIB [2], as well as for the Web, e.g., XChange$^{\text{EQ}}$ [46, 10], the ruleCore Markup Language (rCML) [118], and Reaction RuleML [108]. A further step towards more complex events has been carried out in [114] where *quantitative* temporal event statements are used in a rule's event part. These statements incorporate quantitative ranges of time that allow to model real-time constraints, for example, timeouts and duration. On top of that, rule evaluation frameworks have been developed such as r^3 [4] and MARS [18], accounting for the Web's heterogeneous environment as well as for complex events.

All these approaches, however, share the strategy of reactive policies presented in Section 4.1, that is, they provide reasoners for reactive rules. But a general difference is that the present work adds a level of trust on top of reactive rule reasoning by advanced policy reasoning and trust handling. This additional level features an explicit handling of credentials, interactions among agents, contextual disclosure of information, and the exchange of policies and evidences. In contrast, the approaches to reactivity on the Web provide general-purpose frameworks which meet at most basic requirements for reactive policies such as declarativity or reactivity. However, these are mechanisms provided in order to enable a general-purpose reasoner for reactive rules but they cannot be directly applied for behaviour control of automated agents: they miss many features to support trusted interaction as they are delivered by the work presented in this thesis.

5.2.2 Policy Languages with Reactivity

Supporting reactive behaviour description in a policy language or in policy-based systems is not a completely new field. In this section, policy languages and policy-based systems are reviewed that support a certain level of reactivity or incorporate temporal notions, events, or action control. On top of that, these approaches are compared to the work carried out in the present thesis.

The Policy Description Language (PDL)

The policy description language (PDL) was already introduced at the end of the last century [98]. PDL is a formal policy language based on ECA rules. It combines three other formal languages, namely

1. the composite temporal event language by Motakis and Zaniolo [104] to define event expressions,
2. the *state language* of Geffner and Bonet [68] from the area of AI planning which is suited to express constraints concerning the system's state in order to trigger a rule,
3. the action description language $\mathcal{A}$ by Gelfond and Lifschitz [70, Section 2] to express the effects of a rule's evaluation.

PDL's semantics if defined as a transition function that is a mapping of a series of events into a set of actions. The major motivation for PDL is its application to network control, thus acting as a network policy language (cf. Section 2.1.1). PDL's complexity as well as an implementation by means of a simple Prolog program is given in [98]. PDL is a policy language that features a well-defined formal semantics including definition of events and actions; this makes it a close relative of the language proposed in this thesis' Section 4.1. However, since PDL does not support agent interaction required for trust establishment via negotiations, it addresses scenarios of low-level behaviour control for network peers rather than high-level trust establishment with message and credential exchange required for the scenarios addressed in this thesis. Another difference is the use of variables in PDL: it restricts the policy author to only use constants in a policy's action part thus PDL does not allow for variable bindings crossing events, condition, and action expressions. This feature is essential for policies where actions to be taken depend either on the triggered event or on the result of a condition evaluation. As an example, see Policy 4.7 on page 86 where the action "sending a chat message" can only be executed if the variable *User* is bound to the actual caller triggering the event *callComesIn*.

Ponder and obligation policies

Ponder [56] and its successor Ponder2 [133] are policy languages for pervasive systems. Both distinct two kinds of policies, authorization policies and obligation policies. Authorization policies express standard, non-reactive access control conditions, like which actor is allowed to perform which action on certain objects. Obligation policies, in contrast, specify actions that must be performed when certain events occur. Obligation policies are supported by several policy frameworks and are typically expressed by means of reactive behaviour descriptions. Due to this fact, Ponder and obligation policies in general deserve a deeper analysis in the context of this thesis.

In general, obligation policies are policies specifying the actions that must or most not be performed by components or subjects within a system when certain events occur. They provide the ability to respond to changing circumstances [56], for example, deactivating an account after three consecutive login fails. It is remarkable that these actions are considered to happen *inside* the system in contrast to general reactive policies which allow arbitrary actions to be executed.

Other policy approaches employ an even weaker notion of obligation policies. As examples, SPL [115] as well as the policy model presented in [77], do not even consider the policy system to be in charge of executing the obligated action. There, the only duty of the policy system is to observe if the action is eventually executed by a subject in charge. And in case it is not, some granted rights may be revoked [115] (see [77, Example 14]).

The general relation between obligation policies and the approach of reactive policies presented in this thesis is as follows. Reactive policies can well be used to express obligation policies. Obligations are primarily intended to detect security violations and to inform manager components about the steps to be performed in the system as a consequence to the violation. Still, the means to express obligation policies in Ponder do not suffice for general reactive policies including credential exchange. The main drawback of Ponder is that it does not belong to the category of semantic policy languages: no well-defined semantics for the Ponder language is given. For example, a formal specification of the event and action operators available can not be found. On top of that, similar to PDL, Ponder does not support automated message exchange and agent interaction which is needed for Trust Negotiation.

Protune and provisional actions

Protune is a policy framework with a Logic Programming-inspired policy language [32, 35] and a protocol for exchanging credentials in order to support agent negotiation and automated trust establishment. Of all non-reactive policy languages, Protune shares to most features with the idea of reactive policies. It supports strong and lightweight evidences and employs the same negotiation mechanisms as the approach presented in this thesis (cf. Section 4.1.2 and [51]). Moreover, the principle of *provisional actions* [35] that is exploited in Protune suggests a certain level of action control and a step towards reactivity. However, in the following, the most important differences between Protune and the approach presented in this thesis will be given. Focus will be directed towards the questions why Protune is not able to capture reactivity and, in particular, why actions and events are not first-order citizens in Protune.

Actions in Protune. Similar to the presented approach, Protune supports so-called *provisional actions* which are part of the condition evaluation rather than reactions of such evaluations. In Protune's policy language, these actions are represented by provisional predicates providing means to reason on activities required by a negotiation partner in order to establish trust, such as sending a credential, providing a password (cf. Section 4.1.2, where provisional predicates are introduced as a special case of state predicates). These actions are not reactions in the sense of reactive policies. They rather contribute to the policy evaluation process and capture the idea of cooperative policy evaluation: facing a request the policy owner sends back information about the provisional actions that are required to be executed by the requesting peer. If the requester executes those actions, it contributes to a successful policy evaluation. In contrast, reactive policies as they are presented in this thesis, support reactivity by means of actions being executed *after* a successful negotiation. However, it is worth noting that the reactive policies from Section 4 also have provisional actions since the Trust Negotiation principles there are used in Protune as well.

Using Protune alone to specify reactions to policy evaluations is not possible due to the following reasons. Provisional actions are not suitable to model reactions: they are part of the evaluation process and their execution is not deterministic but depend on the implementation of the policy reasoner. Protune's actual reactions to requests are limited to the acceptance or the denial of a request. Unlike reactive policies, in Protune a call can only be accepted or not but it can not be, for example, put on the answering machine or into a delay loop as alternative reactions. Moreover, reactive policies also allow the request for an access to a resource to be granted only to several extents, like, for example, access is granted but with charging a specific cost or only for a certain time interval. On top of that, defining complex actions in form of action definition rules (see Definition 4.1) is not allowed in Protune.

Events in Protune. Protune basically offers only one single event, namely the arrival of a negotiation message, be it a request or a negotiation reply. However, simple single atomic events could be simulated in Protune by defining a specific request for each event. This requires an additional component that triggers requests in case the according event happens. Although such a simulation could be easily achieved, it would not support reasoning on events including variable bindings or the evaluation of complex events as they are supported in reactive policies by event operators and event definition rules (see Definition 4.1). Reactive policies are designed to directly and pro-actively react to changes in the world such as rebooking flight tickets if a flight is cancelled or the expiration of a credential triggers the re-negotiation of access. In Protune, such behaviour cannot be defined.

Further related approaches with reactivity

Combining notions of events and actions with policies have been considered in other work as well. Some more approaches strongly related in this context will be sketched in the following.

TRBAC - Temporal Role-Based Access Control. TRBAC [23] is an extension to the classical role-based access control model. It additionally supports periodic role enabling and disabling and temporal dependencies among such actions. Although TRBAC is not a reactive policy language per se, it is still relevant in the context of reactive policies since it provides an access control mechanism that allows for event-based constraints.

Classical role-based access control (RBAC) has well-recognized advantages, most prominently, it nicely captures the organizational structures of most institutions. However, RBAC models do not reflect the temporal dimension of such structures which may require roles to expire, to be activated or deactivated upon certain conditions. To this end, TRBAC supports so-called *role triggers* which may be either executed immediately, or deferred by an explicitly specified time span. Compared to this thesis' approach, TRBAC acts on a lower level: reactivity is not used to express reactions to requests or general situations but serves as means for managing roles where conditions are temporal periodic expressions and reactions are always either the activation or the deactivation of a role. The actual access control enforcement in TRBAC is still delivered by simple role-based policies not allowing for credential handling.

Event- and Time-based Policy Model. Another approach along the lines of TRBAC is presented in [77] where a policy model is introduced that allows for the dynamic change of policies either over time or on the occurrence of events. This model exploits an advanced temporal logic for specification of time dependent policy validity, to define, for example, when a policy or its parts become valid. Although this way policies are combined with a reactive behaviour, reactivity is not part of the policies themselves rather than acting on top of them. Additionally, as opposed to reactive policies, reactions are restricted to the validation or deactivation of policies or parts thereof.

Security Policy Language (SPL). An approach that also suffers from the problem that reactions to events are limited is SPL [115]. SPL is an event driven policy language that supports access control as well as obligation policies [55]. In SPL, an event is considered an access request to some resource. SPL's semantics is based on an event monitor that, for each event, decides to either ignore, allow, or disallow the event [55]. SPL's obligation policies are implemented as additional conditions to the access control constraining events happening in the future [115]. Consequently, these obligation policies are different from the ones used in Ponder (cf. page 102): they are not obligations for managers or agents who are obliged to execute specific actions on the occurrence of system events [55], but they are used to keep the system in a stable state by withdrawing actions if obligations are not fulfilled. In summary, the notion of events used in SPL do not suffice to make it a reactive policy language in the sense of Chapter 4.

Chapter 6

Conclusions and Outlook

In this thesis, limitations of current policy-based privacy protection and behaviour control solutions are analyzed and addressed. In particular, research results in the area of formal policy languages and automated trust establishment are presented, extended, and applied to realistic scenarios such as the Social Web. The main contributions of this thesis are as follows:

1. *Development and implementation of a Trust Negotiation model supporting qualitative partial order preferences among credentials.* Influencing the behaviour of a policy-based system featuring Trust Negotiation becomes complex as soon as user-given preferences have to be considered. In order to not interfere with the negotiation process, the approach presented in this thesis supports arbitrary preference statements among credentials. These statements help to guide the automated negotiation in a way that user preferences are considered. Hence, only relevant and most-preferred steps are taken by the negotiation agent. Experiments show that preference handling does not add considerably to the complexity and performance of the negotiation process, in particular compared to bothersome interventions forcing the user to manually select her preferred option during the negotiation.

2. *Definition of a Logic Programming language based on stable model semantics and Answer Set Programming supporting qualitative partial order preferences.* Logic Programming languages are one of the most popular ways to represent policies. Still, possibilities to encode preferences in Logic Programs are limited to several extents, most prominently the ability to express *partial order* preferences. With DLPOD, this thesis presents an extension to the Logic Programming paradigm Answer Set Programming. DLPOD basically forms a careful union of Disjunctive Logic Programs and LPODs (Logic Programs with Ordered Disjunction) in order to model partial order preference expressions in non-monotonic reasoning. It is shown how to

transform a DLPOD into an interleaved Disjunctive Logic Program which allows normal answer set solvers to interpret DLPODs in order to compute preferred answer sets. This transformation is the basis for the presented DLPOD implementation. This implementation exploits a Disjunctive Logic Program reasoner and computes answer sets of any given DLPOD.

3. *Definition of a reactive policy language supporting Trust Negotiation.* Reactive behaviour control with policy features or, contrariwise, policy languages featuring reactivity have not been extensively considered in policy research so far. In this thesis, a declarative policy language with well-defined semantics is presented which constitutes a union of reactive rule reasoning and policy enforcement. By establishing an interplay between message exchange (for Trust Negotiation and credential exchange) and event handling, this language acts as a powerful means for basing behaviour control on secured evidences. To this end, a well-established negotiation scheme was extended and combined with ECA rule reasoning and was further extended to support multiple peers.

4. *Development of a framework supporting reactive policies crossing platforms on the Social Web.* The walled garden of social platforms makes it difficult to refer to several platform's information for policy decisions. This thesis presents a framework which makes use of a reactive policy language and supports the consideration and seamless integration of distributed social data for policy decisions. This framework allows the definition of behaviour control by using reactive rules which, among others, gives the user the opportunity to provide fine-grained privacy policies, to rely on strong authentication, to reuse social data, and to let the framework automatically guide the information flow by means of reactive filtering rules.

5. *Integration of reactive policies into Skype for supporting inter-platform privacy protection and behaviour control.* Skype, as one of the most popular IP-based communication tools [17], provides only a few predefined options for privacy protection and call filtering. In the context of this thesis, the Skype extension SPoX has been developed. By exploiting the principles of reactive policies and Trust Negotiation, SPoX users are offered the possibility to freely define ECA rules for filtering disturbing Skype calls or rejecting chat requests while relying either on social information gathered from the Web, on strong authentication, or on arbitrary context information.

In the following, future directions to continue the research carried out in the context of this thesis as well as to exploit the presented results are pointed out.

Since formal policy languages are tools which are rather tailored for computer experts, in-depth investigation towards human-computer interaction in general and computer-aided policy authoring in particular is an important next step. First achievements in this context have been made in the combination of policies and natural language handling. One direction is to allow users to author policies in natural languages or, more feasible, in a subset of natural language such as controlled natural language[1]. This has been done in [53, 57], where controlled natural English is allowed as policy language and a policy editor helps the user by showing the possible applicable next terms while typing in a policy. A second direction in the intersection of natural language handling and policies is to provide explanations for the outcomes of policy reasoning, as it has been done in [37, 80]. The motivation of this work is to make the reason for a denial of access transparent to the user. Since policies are declarative, they can not only serve as basis for reasoning in order to make a decision; they can in addition be exploited to show a natural language proof tree explaining how the decision was actually reached.

Both directions are worth investigating for reactive policies as well. Although supporting the authoring of reactive policies is already provided by SPoX' policy editor, a general translation of natural language policies into reactive policies is a desired feature and subject to future research. Similarly, generating explanations from reactive policies in order to better communicate reactions automatically executed by the policy-driven system is a future research field building on this thesis. Such a system may adopt ideas from [37] for the condition part of reactive policies, however, the verbalization of event detection and action triggering has to be orchestrated accordingly.

The use of Trust Negotiation techniques relies on the pervasiveness of digital signed credentials and online certificates. Today, such authentication mechanisms are commonly used in company intranets but did not yet reach full worldwide acceptance among online service providers and consumers. Most prominently and important for the context of this thesis, the use of certificates is not yet common for Social Web platforms. In order to fully exploit the potentials of this thesis, such an authentication infrastructure is needed since bridging the walled garden of the Social Web for policy reasoning requires a way of identifying the requester and to further prove her identity. As an example, determining if a caller on Skype is a friend in one's FOAF profile or on Facebook requires information about the caller's identity on other platforms. Future work in this direction may combine upcoming authentication standards with Social Web technology and to further exploit those standards for policy reasoning. Potential solutions are OpenID[2] which may be exploited to identify persons across Social Network boundaries. Alternatively, FOAF+SSL [128] can be used to authenticate requesters using SSL certificates and FOAF networks. FOAF+SSL enables a server to authenticate a client given a simple URL which can then be used directly for policy-based authorization.

[1]see `http://sites.google.com/site/controllednaturallanguage/` for an overview

[2]`http://openid.net/`

Bibliography

[1] Fabian Abel, Eelco Herder, Philipp Kärger, Daniel Olmedilla, and Wolf Siberski. Exploiting preference queries for searching learning resources. In *2nd European Conference on Technology Enhanced Learning (EC-TEL)*, volume 4753 of *Lecture Notes in Computer Science*, pages 143–157, Crete, Greece, Sep 2007. Springer.

[2] Raman Adaikkalavan and Sharma Chakravarthy. Snoopib: Interval-based event specification and detection for active databases. *Data Knowl. Eng.*, 59(1):139–165, 2006.

[3] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. An xpath-based preference language for p3p. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 629–639, New York, NY, USA, 2003. ACM.

[4] José Júlio Alferes and Ricardo Amador. r^3- A foundational ontology for reactive rules. In *ODBASE'07, LNCS 4803*. Springer.

[5] José Júlio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. Towards reactive semantic web policies—motivation scenario and implementation details. Technical report, L3S Research Center, 2008.

[6] José Júlio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. Towards reactive semantic web policies: Advanced agent control for the semantic web. In *Poster and Demo Session ISWC 2008, Karlsruhe, Germany.* CEUR Workshop Proceedings, October 2008.

[7] José Júlio Alferes and Wolfgang May. Evolution and reactivity for the web. In *Reasoning Web*, pages 134–172, 2005.

[8] Grigoris Antoniou, Matteo Baldoni, Piero A. Bonatti, Wolfgang Nejdl, and Daniel Olmedilla. Rule-based policy specification. In Ting Yu and Sushil Jajodia, editors, *Secure Data Management in Decentralized Systems*, volume 33 of *Advances in Information Security*. Springer, 2007.

[9] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, 2. edition, 2008.

[10] James Bailey, François Bry, Michael Eckert, and Paula-Lavinia Patranjan. Flavours of xchange, a rule-based reactive language for the (semantic) web. In *RuleML*, pages 187–192, 2005.

[11] James Bailey, Alexandra Poulovassilis, and Peter T. Wood. An event-condition-action language for xml. In *WWW*, pages 486–495, 2002.

[12] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. Incremental trade-off management for preference based queries. *International Journal of Computer Science and Applications (IJCSA)*, 4(1), 2007.

[13] Wolf-Tilo Balke, Ulrich Güntzer, and Christoph Lofi. User interaction support for incremental refinement of preference-based queries. In *RCIS*, pages 209–220, 2007.

[14] Wolf-Tilo Balke, Ulrich Güntzer, and Jason Xin Zheng. Efficient distributed skylining for web information systems. In *EDBT*, Heraklion, Greece, 2004.

[15] Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.

[16] Moritz Y. Becker and Peter Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *POLICY'04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 159–168, 2004.

[17] Stephan Beckert. International phone traffic growth slows, while skype accelerates. *TeleGeography's CommsUpdate*, January 2010.

[18] Erik Behrends, Oliver Fritzen, Wolfgang May, and Daniel Schubert. An eca engine for deploying heterogeneous component languages in the semantic web. In *EDBT Workshops*, pages 887–898, 2006.

[19] Rachel Ben-Eliyahu and Rina Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.

[20] Jon Louis Bentley, H. T. Kung, Mario Schkolnick, and Clark D. Thompson. On the average number of maxima in a set of vectors and applications. *Journal of the ACM (JACM)*, 25(4), 1978.

[21] Tim Berners-Lee, Jim Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001.

[22] Elisa Bertino, Claudio Bettini, Elena Ferrari, and Pierangela Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Trans. Database Syst.*, 23(3):231–285, 1998.

[23] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[24] Elisa Bertino, Piero Andrea Bonatti, Elena Ferrari, and Maria Luisa Sapino. Temporal authorization bases: from specification to integration. *J. Comput. Secur.*, 8(4):309–353, 2000.

[25] Elisa Bertino, Alessandra Mileo, and Alessandro Provetti. PDL with preferences. In *POLICY 2005*, Los Alamitos,USA. IEEE Computer Society.

[26] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In *Security Protocols, 6th International Workshop*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63, Cambridge, April, 1998. Springer.

[27] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[28] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance checking in the policymaker trust management system. In *Financial Cryptography, Second International Conference*, volume 1465 of *Lecture Notes in Computer Science*, pages 254–274, Anguilla, British West Indies, February 1998. Springer.

[29] Uldis Bojars, John G. Breslin, Aidan Finn, and Stefan Decker. Using the semantic web for linking and reusing data across web 2.0 communities. *Web Semant.*, 6(1):21–28, 2008.

[30] Piero Bonatti, Philipp Kärger, and Daniel Olmedilla. Reactive policies for the semantic web. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece*, June 2010.

[31] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 134–143, New York, NY, USA, 2000. ACM.

[32] Piero A. Bonatti, Juri L. De Coi, Daniel Olmedilla, and Luigi Sauro. A rule-based trust negotiation system. *IEEE Transactions on Knowledge and Data Engineering, to appear*, 2010.

[33] Piero A. Bonatti, Juri Luca De Coi, Daniel Olmedilla, and Luigi Sauro. Rule-based policy representations and reasoning. In Francois Bry and Jan Maluszynski, editors, *Semantic Techniques for the Web. The REWERSE Perspective*, volume 5500 of *Lecture Notes in Computer Science*, pages 201–232. Springer, 2009.

[34] Piero A. Bonatti, Claudiu Duma, Norbert Fuchs, Wolfgang Nejdl, Daniel Olmedilla, Joachim Peer, and Nahid Shahmehri. Semantic web policies - a discussion of requirements and research issues. In *3rd European Semantic Web Conference (ESWC)*, volume 4011 of *Lecture Notes in Computer Science*, Budva, Montenegro, June 2006. Springer.

[35] Piero A. Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE Policies for Distributed Systems and Networks (POLICY 2005)*, pages 14–23, Stockholm, Sweden, June 2005. IEEE Computer Society.

[36] Piero A. Bonatti and Daniel Olmedilla. Rule-based policy representation and reasoning for the semantic web. In *Reasoning Web, Third International Summer School 2007*, volume 4636 of *Lecture Notes in Computer Science*, pages 240–268, Dresden, Germany, September 2007. Springer.

[37] Piero A. Bonatti, Daniel Olmedilla, and Joachim Peer. Advanced policy explanations on the web. In *17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 200–204, Riva del Garda, Italy, Aug-Sep 2006. IOS Press.

[38] Angela Bonifati and Stefano Paraboschi. Active xquery. In *Web Dynamics*, pages 249–274. 2004.

[39] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *International Conference on Data Engineering*, Heidelberg, Germany, 2001.

[40] John Breslin and Stefan Decker. The future of social networks on the internet: The need for semantics. *IEEE Internet Computing*, 11(6):86–90, 2007.

[41] John G. Breslin, Alexandre Passant, and Stefan Decker. *The Social Semantic Web.* Springer, ISBN: 978-3-642-01171-9, 2010.

[42] Gerhard Brewka. Complex preferences for answer set optimization. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004.* AAAI Press.

[43] Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 444–455, London, UK, 2002. Springer-Verlag.

[44] Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence*, 20:335–357(23), May 2004.

[45] François Bry and Michael Eckert. Twelve theses on reactive rules for the web. In *EDBT Workshop Reactivity on the Web, Munich, Germany, 2006*, pages 842–854, 2006.

[46] François Bry and Michael Eckert. Rule-based composite event queries: The language xchange$^{\mathrm{eq}}$ and its semantics. In *RR*, pages 16–30, 2007.

[47] Scott W. Campbell and Tracy C. Russo. *The social construction of mobile telephony: an application of the social influence model to perceptions and uses of mobile phones within personal communication networks.* Communication Monographs 7 (4), 317-334, 2003.

[48] Weifeng Chen, Lori Clarke, Jim Kurose, and Don Towsley. Optimizing costsensitive trust-negotiation protocols. In *Annual Joint Conference of the IEEE Computer and Communications Societies*, 2005.

[49] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[50] Juri L. De Coi, Philipp Kärger, Daniel Olmedilla, and Sergej Zerr. Semantic web policies for security, trust management and privacy in social networks. In *Workshop on Privacy and Protection in Web-based Social Networks in conjunction with the 12th International Conference on Artificial Intelligence & Law (ICAIL)*, Barcelona, Spain, June 2009.

[51] Juri L. De Coi and Daniel Olmedilla. A flexible policy-driven trust negotiation model. In *IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Silicon Valley, CA, USA, November 2007.

[52] Juri L. De Coi and Daniel Olmedilla. A review of trust management, security and privacy policy languages. In *International Conference on Security and Cryptography (SECRYPT 2008)*. INSTICC Press, July 2008.

[53] Juri Luca De Coi, Philipp Kärger, Daniel Olmedilla, and Sergej Zerr. Using natural language policies for privacy control in social platforms. In *ESWC Workshop on Trust and Privacy on the Social and Semantic Web (SPOT 2009)*, 2009.

[54] Lorrie Faith Cranor and Lawrence Lessig. *Web Privacy with P3P*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[55] Nicodemos Damianou, Arosha K Bandara, Morris Sloman, and Emil C Lupu. A survey of policy specification approaches. Technical report, Department of Computing at Imperial College of Science Technology and Medicine, 2002.

[56] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *POLICY*, pages 18–38, 2001.

[57] Juri Luca De Coi, Peter Fankhauser, Tobias Kuhn, Wolfgang Nejdl, and Daniel Olmedilla. Controlled natural language policies. In *In: Proceedings of the 16th ACM Conference on Computer and Communications Security, CCS 2009, Chicago, IL, USA*, 2009.

[58] James P. Delgrande, Torsten Schaub, Hans Tompits, and Kewen Wang. Towards a classification of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20:308–334, 2003.

[59] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. Aggregate functions in disjunctive logic programming: Semantics, complexity, and implementation in dlv. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico*, 2003.

[60] Sven Döring, Timotheus Preisinger, and Markus Endres. Advanced preference query processing for e-commerce. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, New York, NY, USA, 2008. ACM.

[61] Matthew Duckham and Lars Kulik. Location privacy and location-aware computing. *Dynamic & Mobile GIS: Investigating Change in Space and Time. CRC Press, Boca Raton*, page 3551, 2006.

[62] Phan Minh Dung. On the relations between stable and well-founded semantics of logic programs. *Theoretical Computer Science*, 105(1):7 – 25, 1992.

[63] Thomas Eiter and Michael Fink. Uniform equivalence of logic programs under the stable model semantics. In *International Conference on Logic Programming*, pages 224–238, 2003.

[64] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.

[65] Thomas Eiter and Axel Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *Theory and Practice of Logic Programming (TPLP)*, 6(1-2):23–60, 2006.

[66] Peter Fishburn. Preference structures and their numerical representations. *Theoretical Computer Science*, 217:359–383, 1999.

[67] Rita Gavriloaie, Wolfgang Nejdl, Daniel Olmedilla, Kent E. Seamons, and Marianne Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *ESWS 2004*, Heraklion, Crete, Greece. Springer.

[68] Hector Geffner and Blai Bonet. High-level planning and control with incomplete information using pomdps. AAAI Technical Report WS-98-02, 1998.

[69] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[70] Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[71] Tyrone Grandison and E. Michael Maximilien. Towards privacy propagation in the social web. In *Workshop on Web 2.0 Security and Privacy at the 2008 IEEE Symposium on Security and Privacy. Oakland, California, USA, 18-21 May 2008.*

[72] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *IEEE Communications Surveys and Tutorials*, 3(4), 2000.

[73] Torsten Grust, Hagen Höpfner, Arantza Illarramendi, Stefan Jablonski, Marco Mesiti, Sascha Müller, Paula-Lavinia Patranjan, Kai-Uwe Sattler, Myra Spiliopoulou, and Jef Wijsen, editors. *EDBT Workshops, Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers*. Springer.

[74] Stefan Holland and Werner Kießling. Situated preferences and preference repositories for personalized database applications. In *ER*, pages 511–523, 2004.

[75] Katsumi Inoue and Chiaki Sakama. Abducing priorities to derive intended conclusions. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI99)*, pages 44–49. Morgan Kaufmann Publishers, 1999.

[76] ISO/IEC 14977:1996(E). *The standard of extended BNF.* ISO, Geneva, Switzerland.

[77] Helge Janicke, Antonio Cau, François Siewe, Hussein Zedan, and Kevin Jones. A compositional event & time-based policy model. In *POLICY*, pages 173–182, 2006.

[78] Ken Jordan, Jan Hauser, and Steven Foster. The augmented social network: Building identity and trust into the next-generation internet. *First Monday*, 8, 2003.

[79] Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In *ISWC 2003, Sanibel Island, FL, USA, 2003*, Lecture Notes in Computer Science. Springer, 2003.

[80] Lalana Kagal, Chris Hanson, and Daniel J. Weitzner. Using dependency tracking to provide explanations for policy management. In *9th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2008), June 2008, Palisades, New York, USA*, 2008.

[81] Philipp Kärger. Advanced semantic web policies: Evolution reactivities, and priorities. In *Doctoral Consortium at the 7th International Semantic Web Conference, Karlsruhe, Germany*, Lecture Notes in Computer Science. Springer, 2008.

[82] Philipp Kärger, Emily Kigel, and VenkatRam Yadav Jaltar. Spox: combining reactive semantic web policies and social semantic data to control the behaviour of skype. In *ISWC, Demo Session, Washington, DC, USA*, October 2009.

[83] Philipp Kärger, Emily Kigel, and Daniel Olmedilla. Reactivity and social data: Keys to drive decisions in social network applications. In *Second ISWC Workshop on Social Data on the Web (SDoW2009).*

[84] Philipp Kärger, Nuno Lopes, Daniel Olmedilla, and Axel Polleres. Towards logic programs with ordered and unordered disjunction. In *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2008), 24th International Conference on Logic Programming (ICLP 2008)*, Udine, Italy, 12 2008.

[85] Philipp Kärger, Daniel Olmedilla, Fabian Abel, Eelco Herder, and Wolf Siberski. What do you prefer? using preferences to enhance learning technology. *IEEE Transactions on Learning Technologies*, 1(1), 2008.

[86] Philipp Kärger, Daniel Olmedilla, and Wolf-Tilo Balke. Exploiting preferences for minimal credential disclosure in policy-driven trust negotiations. In *VLDB Workshop on Secure Data Management (SDM)*, Lecture Notes in Computer Science, Auckland, New Zealand, August 2008. Springer.

[87] Philipp Kärger and Wolf Siberski. Guarding a walled garden - semantic privacy preferences for the social web. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece*, June 2010.

[88] Ralph Keeney and Howard Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* Wiley, 1976.

[89] Werner Kießling. Foundations of preferences in database systems. In *International Conference on Very Large Data Bases*, Hong Kong, China, 2002.

[90] Josef Kolbitsch and Hermann A. Maurer. The transformation of the web: How emerging communities shape the information we consume. *J. UCS*, 12(2):187–213, 2006.

[91] H. T. Kung, Fabrizio Luccio, and Franco P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM (JACM)*, 22(4), 1975.

[92] Marc Langheinrich, Lorrie Cranor, and Massimo Marchiori. Appel: A p3p preference exchange language. W3C Working Draft, April 2002.

[93] Adam J. Lee and Marianne Winslett. Enforcing safety and consistency constraints in policy-based authorization systems. *ACM Trans. Inf. Syst. Secur.*, 12(2), 2008.

[94] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[95] Jiangtao Li and Ninghui Li. Oacerts: Oblivious attribute certificates. *IEEE Trans. Dependable Sec. Comput.*, 3(4), 2006.

[96] Vladimir Lifschitz. Closed-world databases and circumscription. *Artif. Intell.*, 27(2):229–235, 1985.

[97] John W. Lloyd. *Foundations of logic programming.* Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[98] Jorge Lobo, Randeep Bhatia, and Shamim Naqvi. A policy description language. In *In Proc. of AAAI*, pages 291–298, 1999.

[99] Xudong Luo, Nicholas R. Jennings, and Nigel Shadbolt. Knowledge-based acquisition of tradeoff preferences for negotiating agents. In *International Conference on Electronic Commerce*, New York, NY, USA, 2003. ACM Press.

[100] Frank Manola and Eric Miller, editors. *RDF Primer.* W3C Recommendation. World Wide Web Consortium, February 2004.

[101] Wolfgang May, José Júlio Alferes, and François Bry. Towards generic query, update, and event languages for the semantic web. In *PPSWR*, pages 19–33, 2004.

[102] Michael McGeachie and Jon Doyle. Efficient utility functions for ceteris paribus preferences. In *Conference on Artificial Intelligence and Conference on Innovative Applications of Artificial Intelligence*, Edmonton, Canada, 2002.

[103] Alessandra Mileo and Torsten Schaub. Qualitative constraint enforcement in advanced policy specification. In *9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertaint,(ECSQARU), Hammamet, Tunisia*, pages 695–706, 2007.

[104] Iakovos Motakis and Carlo Zaniolo. Temporal aggregation in active database rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, USA*, 1997.

[105] Ilkka Niemelä. Language extensions and software engineering for ASP. Technical report, European Working group on Answer Set Programming, 2005.

[106] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *ACM SIGMOD*, San Diego, CA, USA, 2003.

[107] George Papamarkos, Alexandra Poulovassilis, and Peter T. Wood. Event-condition-action rules on rdf metadata in p2p environments. *Computer Networks*, 50(10):1513–1532, 2006.

[108] A. Paschke, A. Kozlenkov, H. Boley, S. Tabet, M. Kifer, and M. Dean. *Reaction RuleML.* RuleML Initiative, http://ibis.in.tum.de/research/ReactionRuleML/, 2007.

[109] Alexandre Passant, Philipp Kärger, Michael Hausenblas, Daniel Olmedilla, Axel Polleres, and Stefan Decker. Enabling trust and privacy on the social web. In *W3C Workshop on the Future of Social Networking*, Barcelona, Spain, January 2009.

[110] Jon Peterson. A presence-based geopriv location object format. Internet Engineering Task Force, RFC 4119, December 2005.

[111] Alexandra Poulovassilis, George Papamarkos, and Peter T. Wood. Event-condition-action rule languages for the semantic web. In *Reactivity on the Web, EDBT Workshop*, pages 855–864, 2006.

[112] Shekhar Pradhan and Jack Minker. Using priorities to combine knowledge bases. *Int. J. Cooperative Inf. Syst.*, 5(2&3), 1996.

[113] Teodor C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.

[114] Ying Qiao, Hongan Wang, Kang Zhong, and Xiang Li. Visual event-condition-action rules with temporal events. In *Eighth Real-Time Linux Workshop, Lanzhou*, 2006.

[115] Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies and complex constraints. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2001, San Diego, California, USA. The Internet Society 2001.*

[116] Chiaki Sakama and Katsumi Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artif. Intell.*, 123(1-2):185–222, 2000.

[117] Kent E. Seamons, Marianne Winslett, Ting Yu, Bryan Smith, Evan Child, Jared Jacobson, Hyrum Mills, and Lina Yu. Requirements for policy languages for trust negotiation. In *POLICY*, pages 68–79, 2002.

[118] Marco Seiriö and Mikael Berndtsson. Design and implementation of an eca rule markup language. In *RuleML*, pages 98–112, 2005.

[119] Oshani Seneviratne, Lalana Kagal, and Tim Berners-Lee. Policy-aware content reuse on the web. In *8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009*, pages 553–568.

[120] Patrik Simons. Extending the smodels system with cardinality and weight constraints. In *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer Academic Publishers, 2000.

[121] Steven Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica.* Addison-Wesley, 1990.

[122] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[123] Morris Sloman and Emil Lupu. Security and management policy specification. *IEEE Network*, 16(2):10–19, March 2002.

[124] Tran Cao Son and Jorge Lobo. Reasoning about policies using logic programs. In *Proceedings of the 1st Intl. Answer Set Programming Workshop, Stanford*, 2001.

[125] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.

[126] Daniel F. Sterne. On the buzzword 'security policy'. In *Research in Security and Privacy, 1991. Proceedings., 1991 IEEE Computer Society Symposium on*, pages 219–230, May 1991.

[127] G.N. Stone, B. Lundy, and G.G. Xie. Network policy languages: a survey and a new approach. *Network, IEEE*, 15(1):10–21, Jan/Feb 2001.

[128] Henry Story, Bruno Harbulot, Ian Jacobi, and Mike Jones. FOAF+SSL: RESTful Authentication for the Social Web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009), Heraklion, Crete. CEUR-WS.org/Vol-447*, 2009.

[129] V. S. Subrahmanian, Piero A. Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert B. Ross. *Heterogenous Active Agents.* MIT Press, 2000.

[130] Latanya Sweeney. Guaranteeing anonymity when sharing medical data, the datafly system. *Journal of the American Medical Informatics Association*, 1997.

[131] Alessandra Toninelli, Deepali Khushraj, Ora Lassila, and Rebecca Montanari. Towards socially aware mobile phones. In *First Workshop on Social Data on the Web (SDoW)*, 2008.

[132] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In *International Semantic Web Conference*, pages 419–437, 2003.

[133] Kevin P. Twidle, Emil Lupu, Naranker Dulay, and Morris Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *POLICY*, pages 245–246, 2008.

[134] Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Maggie R. Breedy, Larry Bunch, Matt Johnson, Shriniwas Kulkarni, and James Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, page 93, 2003.

[135] Claudia Wagner and Enrico Motta. Data Republishing on the Social Semantic Web. In *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009), Heraklion, Crete. CEUR-WS.org/Vol-447*, 2009.

[136] Jennifer Widom and Stefano Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing.* Morgan Kaufmann, 1996.

[137] William H. Winsborough and Ninghui Li. Automated trust negotiation. In *In DARPA Information Survivability Conference and Exposition, volume I*, pages 88–102. IEEE Press, 2000.

[138] William H. Winsborough and Ninghui Li. Safety in automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 9(3):352–390, 2006.

[139] Danfeng Yao, Keith Frikken, Mike Atallah, and Roberto Tamassia. Point-based trust: Define how much privacy is worth. In *International Conference on Information and Communications Security (ICICS '06). Springer. North Carolina, USA*, 2006.

[140] Ting Yu, Ninghui Li, and Annie I. Antón. A formal semantics for p3p. In *SWS '04: Proceedings of the 2004 workshop on Secure web service*, pages 1–8, New York, NY, USA, 2004. ACM.

[141] Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *CCS*, 2001.

Publications

1. Philipp Kärger and Wolf Siberski. *Guarding a Walled Garden - Semantic Privacy Preferences for the Social Web.* In 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece, June 2010.

2. Piero Bonatti, Philipp Kärger, and Daniel Olmedilla. *Reactive Policies for the Semantic Web.* In 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece, June 2010.

3. Philipp Kärger, Daniel Olmedilla, Alexandre Passant, Axel Polleres (eds.). *Proceedings of the Second Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2010)*, Heraklion, Greece, May 31st, 2010, CEUR Workshop Proceedings, ISSN 1613-0073, online CEUR-WS.org/Vol-447/.

4. Philipp Kärger, Emily Kigel, and Daniel Olmedilla. *Reactivity and Social Data: Keys to Drive Decisions in Social Network Applications.* In 2nd International ISWC Workshop on Social Data on the Web (SDoW2009), Washington, DC, USA, October, 2009.

5. Philipp Kärger, Emily Kigel, and VenkatRam Yadav Jaltar. *SPoX: Combining Reactive Semantic Web Policies and Social Semantic Data to Control the Behaviour of Skype.* In International Semantic Web Conference (ISWC 2009), Poster and Demo Session, Washington, DC, USA, October, 2009.

6. Juri L. De Coi, Philipp Kärger, Daniel Olmedilla, and Sergej Zerr. *Semantic Web policies for Security, Trust Management and Privacy in Social Networks.* In Workshop on Privacy and Protection in Web-based Social Networks in conjunction with the 12th International Conference on Artificial Intelligence and Law (ICAIL), Barcelona, Spain, June 2009.

7. Michael Hausenblas, Philipp Kärger, Daniel Olmedilla, Alexandre Passant, Axel Polleres (eds.). *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*, Heraklion, Greece, June 1, 2009, CEUR Workshop Proceedings, ISSN 1613-0073, online CEUR-WS.org/Vol-447/.

8. Juri Luca De Coi, Philipp Kärger, Daniel Olmedilla, Sergej Zerr. *Using Natural Language Policies for Privacy Control in Social Platforms.* In: Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009), Heraklion, Greece, June 1, 2009, CEUR Workshop Proceedings, ISSN 1613-0073.

9. Alexandre Passant, Philipp Kärger, Michael Hausenblas, Daniel Olmedilla, Axel Polleres, and Stefan Decker. *Enabling Trust and Privacy on the Social Web.* In W3C Workshop on the Future of Social Networking, Barcelona, Spain, January 2009.

10. Philipp Kärger, Nuno Lopez, Axel Polleres, and Daniel Olmedilla. *Towards Logic Programs with Ordered and Unordered Disjunction.* In ICLP Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), Udine, Italy, December 2008.

11. Jose Julio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. *Towards Reactive Semantic Web Policies: Advanced Agent Control for the Semantic Web.* In Poster and Demo Session of the 7th International Semantic Web conference (ISWC 2008), Karlsruhe, Germany, October 2008.

12. Jose Julio Alferes, Ricardo Amador, Philipp Kärger, and Daniel Olmedilla. *Towards Reactive Semantic Web Policies—Motivation Scenario and Implementation Details.* Technical Report, L3S Research Center, Hannover, Germany, October 2008.

13. Philipp Kärger, Daniel Olmedilla, and Wolf-Tilo Balke. *Exploiting Preferences for Minimal Credential Disclosure in Policy-driven Trust Negotiations.* In VLDB Workshop on Secure Data Management (SDM), Lecture Notes in Computer Science, Auckland, New Zealand, August 2008. Springer.

14. Philipp Kärger. *Advanced Semantic Web Policies: Evolution, Reactivity, Priority.* PhD Symposium of the 7th International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, October 2008.

15. Philipp Kärger, Daniel Olmedilla, Fabian Abel, Eelco Herder, and Wolf Siberski. *What do you prefer? Using Preferences to Enhance Learning Technology.* IEEE Journal for Transactions on Learning Technologies, 1(1), 2008.

16. Juri L. De Coi, Philipp Kärger, Arne W. Koesling, and Daniel Olmedilla. *Control your e-learning environment: Exploiting Policies in an Open Infrastructure for Lifelong Learning.* IEEE Transactions on Learning Technologies, 1(1), 2008.

17. Eelco Herder and Philipp Kärger. *Hybrid Personalization for Recommendations.* Proc. of the 16th Workshop on Adaptivity and User Modeling in Interactive Systems (ABIS), 2008, Würzburg, Germany.

18. Juri Luca De Coi, Philipp Kärger, Arne W. Koesling, Daniel Olmedilla. *Exploiting Policies in an Open Infrastructure for Lifelong Learning.* Creating New Learning Experiences on a Global Scale, Second European Conference on Technology Enhanced Learning, EC-TEL 2007, Crete, Greece, September 17-20, 2007.

19. Fabian Abel, Eelco Herder, Philipp Kärger, Daniel Olmedilla, Wolf Siberski. *Exploiting Preference Queries for Searching Learning Resources.* In: Creating New Learning Experiences on a Global Scale, Second European Conference on Technology Enhanced Learning, EC-TEL 2007, Crete, Greece, September 17-20, 2007.

20. Elena Demidova, Philipp Kärger, Daniel Olmedilla, Stefaan Ternier, Erik Duval, Michele Dicerto, Carlos Mendez, Krassen Stefanov. *Services for Knowledge Resource Sharing & Management in an Open Source Infrastructure for Lifelong Competence Development.* In: Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies (ICALT), July 2007, Niigata, Japan.